Diseño de Comportamiento Emergente en Arquitecturas Adaptativas con Tecnologías del Acuerdo

Tesis presentada por

José Santiago Pérez Sotelo

Departamento de Ciencias de la Computación, Arquitectura de Computadores,
Lenguajes y Sistemas Informáticos y Estadística e Investigación Operativa

para la obtención del título de

Doctor en Informática

Dirigida por

Carlos E. Cuesta Quintero
Dirk Sascha Ossowski

Universidad Rey Juan Carlos
Octubre 2015

D. Carlos E. Cuesta Quintero, Profesor Contratado Doctor de la Universidad Rey Juan Carlos, con DNI 09331980Y, y **D. Sascha Ossowski**, Profesor Catedrático de la Universidad Rey Juan Carlos, con NIE X1805919M,

CERTIFICAN: Que D. José Santiago Pérez Sotelo, Master en Tecnologías de la Información y Sistemas Informáticos, ha realizado en el Departamento de Ciencias de la Computación, Arquitectura de Computadores, Lenguajes y Sistemas Informáticos, Estadística e Investigación Operativa, bajo su dirección, el trabajo correspondiente a la tesis doctoral titulada:

Diseño de Comportamiento Emergente en Arquitecturas Adaptativas con Tecnologías del Acuerdo

Revisado el presente trabajo, estiman que puede ser presentado al tribunal que ha de juzgarlo. Y para que conste a efecto de lo establecido en la normativa reguladora del tercer ciclo de la Universidad Rey Juan Carlos, autorizan su presentación.

Móstoles, Octubre de 2015

Fdo. Carlos E. Cuesta Quintero

Agradecimientos

Una tesis doctoral puede parecer un trabajo individual y en solitario pero la realidad demuestra que no es cierto. Si a esto le sumamos la frase “hay que ser agradecido” que mi madre siempre repetía, lo que corresponde es dar las gracias a todas las personas que a su manera ha tenido que ver en este trabajo.

Muchas gracias a mis supervisores, Carlos E. Cuesta Quintero y Sascha Ossowski, por el constante apoyo, los ánimos, la confianza y la decisiva orientación durante el desarrollo del trabajo.

A los compañeros de GIA: Alberto, Carlos, Cliff, Holger, José Javier, Levan, Luis, Marin, Matteo, Moser, Radu, Ramón, Roberto y Rubén. También a los grupos Kybele y Vortice, y por supuesto a las secretarias Tuni y Carmen.

Al Profesor Andrea Omicini, del Grupo de Investigación APICe, en *Alma Mater Studiorum-Università di Bologna* (Italia), por sus interesantes aportaciones y conversaciones que hemos compartido durante mi estancia en su hermosa ciudad de Bolonia.

A mis amigos Verónica, Mariano, Valeria, Elisa, Josemari, César y los llegados hace muy poquito, Lucía e Ignacio. También a Migue, Luismi, Ángel y Mingo por la buena compañía en Madrid.

A mi querida familia argentina, Goyita, Marta, Hugo y Víctor, que ya no están pero continúan siendo mi compañía y guía. A Roberto, Marcela y Ernesto, y Juan (ya como de la familia) por todo estos años de soporte y cariño a la distancia. Y por supuesto a muchos amigos y familiares más que siempre han estado presente.

Por último, pero no por eso menos importante, a Juanqui, mi compañero, amigo, confidente, soporte. Si no lo hubiera conocido posiblemente los últimos tiempos habrían sido muy difíciles de sobrellevar.

A todos ¡muchas gracias!.

A Goyita, Marta, Hugo y Víctor

Resumen

Los sistemas de software son cada vez más complejos. Esto es una certeza desde hace mucho tiempo, y lo serán mucho más en los años venideros. Esa complejidad va incorporando características que van más allá de lo tecnológico, y podríamos decir que atraviesan muchas disciplinas.

Tanto es así que es muy usual el trabajo en equipo de personas con muy dispares profesiones, del cual suelen obtenerse resultados asombrosos y muy satisfactorios.

Una de los enfoques más interesantes de los últimos años han sido las *Tecnologías del Acuerdo*. Estas representan un nuevo paradigma en el cual las interacciones entre entidades computacionales están regidas por un concepto mucho más humano que técnico: el *acuerdo*.

Sus características son especialmente adecuadas para hacer frente a la creciente complejidad en los sistemas, la heterogeneidad y elevado número de sus elementos participantes, la apertura y la dinámica de los dominios de aplicación, etc.

Ante este panorama la elección del tema de esta tesis propone un trabajo que intenta aportar una solución arquitectónica a los recurrentes problemas de coordinación, organización y adaptación de sistemas software. Estos sistemas han sido tradicionalmente estáticos y poco flexibles en su estructura, pero las necesidades actuales exigen la capacidad de evolucionar y adaptarse dinámicamente a entornos cambiantes. De ahí el énfasis de los últimos años en los sistemas adaptativos y auto-organizados.

La utilización de sistemas multiagente, junto con propuestas avanzadas como la utilización de estructuras organizacionales y servicios como elementos importantes, nos permite plantear soluciones que se basan en la posibilidad de lograr adaptación dinámica en los sistemas estudiados.

Se propone una solución arquitectónica, donde el dinamismo necesario y el comportamiento serán proporcionados por los *acuerdos emergentes*, con el objetivo de definir *estructuras organizacionales adaptativas basadas en acuerdos*, con énfasis en los mecanismos adaptativos de coordinación y mediante la aplicación de *patrones de adaptación* que guiarán el desarrollo de estas arquitecturas.

Abstract

Software systems are becoming more complex. This is a fact for a long time. And for sure they will be more complex the coming years. This complexity adds some features that go beyond technology and we can see that spans many disciplines.

So much so that we can find teams with people working in very different professions, and they are able to achieve amazing and excellent results.

One of the most interesting approaches in those years have been the *Agreement Technologies*. They represent a new paradigm in which interactions between computational entities are ruled by a more human concept than technical: *the agreement*.

They have features that are particularly suited to address the increasing complexity in systems, heterogeneity and high number of elements involved, openness and dynamics of application domains, etc.

Facing this overview, the subject of this thesis has been chosen to propose an architectural approach to recurrent problem of coordination, organization and adaptation of software systems. These systems have been traditionally static and they lack of structural flexibility. Actual necessities demand evolution capacity and they must dynamically adapt to changing environments. The emphasis now is to have adaptive and self-organizing systems.

Multi Agent systems, together with advanced proposals like the use of organizational structures and the service orientation as important features allow us to propose solutions based on the possibility to achieve dynamic adaptation in software systems.

We propose an architectural solution, in which the necessary dynamism and behaviour will be supported by emergent agreements. The objective is to define adaptive organizational structures based on agreements, with emphasis on adaptive mechanisms of coordination and the use of adaptation patterns to guide our architectural development.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Al final lo que cuenta es lo que se hace,

y no lo que se tenía intención de hacer.

Pablo Picasso (1881-1973)

1.1. Motivación

El objetivo principal de esta tesis es la definición de esquemas de funcionamiento que permitan determinar comportamientos adaptativos en un entorno de las Tecnologías del Acuerdo (*Agreement Technologies - AT*).

Este trabajo se enmarca en el contexto de sistemas auto-adaptativos o *self-adaptive systems*. Existe un amplio campo de auto-adaptación que se extiende en el área de la Inteligencia Artificial (IA), y también fuera de esta. Lo que se intenta es acometerlo en el contexto de nuestro ámbito de investigación que comprende a las Tecnologías del Acuerdo (AT a partir de ahora).

AT es un campo emergente en ciencias de la computación que se enfoca en sistemas computacionales en los cuales *agentes de software* autónomos, a través de negociaciones, pueden llegar a acuerdos aceptables para ambas partes. Este nuevo paradigma involucra características teóricas y prácticas tan importantes como la movilidad y las interacciones,

además de la ya nombrada autonomía, en entornos cada vez más abiertos, distribuidos y complejos.

La utilización del agente de software en IA para la resolución de problemas es una práctica que viene desarrollándose con éxito desde finales de los años setenta del siglo pasado. Posteriormente su uso formando conjuntos de agentes, los conocidos como *sistemas multiagente - SMA* y que son una versión distribuida de ellos, ha tenido notable crecimiento como campo de investigación desde mediados de los noventa. Estos sistemas software son cada vez más populares en el área de IA para la resolución de problemas complejos.

Ante el dinamismo creciente en los desarrollos de SMA, en los últimos tiempos se han incorporado modelos organizativos de agentes para hacer frente, por ejemplo, a los problemas de coordinación. Estos modelos se enfocan en perspectivas organizacionales desde los inicios del ciclo de desarrollo. Las organizaciones de agentes serán más o menos complejas en base a la apertura y heterogeneidad de los SMA, al igual que sería deseable que presentaran capacidades de adaptación ante cambios de su entorno.

Tomando como punto de partida que gran parte del esquema original de AT no permitía una real adaptación, definiendo a esta como no solo un cambio de estructura de las organizaciones de agentes sino también de su composición, la idea principal es conseguir que nuestras propias organizaciones pudieran definir ecosistemas variables. A pesar que en AT el modelo propuesto es muy flexible, de hecho mucho más que los SMA usuales, y aunque los agentes tengan capacidad de aprendizaje, de tener interacciones de alto nivel, consiguieran negociar, argumentar, lograr confianza, en pocas palabras, pudieran evolucionar, la realidad es que no tenían una estructura capaz de adaptarse. Podían sí adaptarse a determinadas circunstancias por su propia inteligencia pero en sí mismos no eran adaptativos.

Es así que surge la primera pregunta *¿podemos hacerlos adaptativos?*, a la que sigue *¿y cómo lo hacemos?*. Si la primera respuesta es afirmativa, la segunda es *introduciendo mecanismos de adaptación*.

La motivación de esta tesis, lo que en definitiva constituye el principal objetivo, es *cómo lograr introducir comportamiento adaptativo en un entorno AT*. Para conseguir esto

la inspiración e idea global que se desarrollará en los próximos capítulos, y que solo se enuncia en esta introducción, es que procederemos a definir una serie de esquemas de comportamiento que se disparan en lugares concretos y en base a reacciones concretas. Hecho esto de tal manera de otorgar flexibilidad y dinamismo a estructuras que podría considerarse rígidas y altamente estáticas.

1.2. Objetivos

Con la motivación presentada en la Sección anterior y con la meta de responder a la principal pregunta que ha surgido: *¿cómo lograr introducir comportamiento adaptativo en un entorno de Tecnologías del Acuerdo?* en esta Sección presentaremos los objetivos de este trabajo de tesis.

1. Se realizará un detallado estudio del estado del arte de los conceptos relacionados con la tesis. Desde los agentes inteligentes y sistemas multiagentes hasta la utilización de patrones de arquitectura de software para lograr adaptación en los sistemas. Además de estos temas también se analizarán la coordinación y la capacidad de adaptación en numerosos trabajos de investigación para encontrar los aspectos relevantes a ser superados.

2. Se proporcionará un enfoque conceptual y tecnológico que permita integrar las carencias detectadas en los modelos estudiados, con el fin de establecer la arquitectura adaptativa capaz de responder a los requerimientos propuestos.

3. Se definirán elementos y construcciones para el desarrollo de un formato estandar y que sirva para representar el comportamiento emergente del sistema. Primero con la definición de un ciclo de vida y a continuación los esquemas que participarán en las soluciones, que para facilitar su comprensión adoptarán la forma de patrones.

4. Se listarán y describirán todos y cada uno de los patrones necesarios para definir un esquema básico de adaptación como un lenguaje de patrones capaz de representar el ciclo de vida propuesto.

5. Se realizarán las pruebas de concepto correspondientes y en dominios concretos con el fin de verificar que este sistema es viable y que el enfoque es especialmente adecuado para proporcionar comportamiento emergente a las arquitecturas adaptativas.

1.3. Estructura de la Memoria

Para lograr los objetivos mencionados, la memoria de este trabajo de tesis está estructurada de la siguiente manera:

- ***Capítulo 2***. En este capítulo se presenta un estudio detallado del estado del arte de los conceptos y trabajos de investigación relacionados con la tesis.

- ***Capítulo 3***. En este apartado se describen las condiciones de partida para la obtención de arquitecturas adaptativas de software. Se presentan las principales cuestiones conceptuales y tecnológicas consideradas relevantes para establecer esa arquitectura adaptativa, que en nuestro enfoque estará orientada a organizaciones de agentes de software.

- ***Capítulo 4***. En este capítulo se discute con más detalle el enfoque propuesto para obtener el comportamiento emergente del sistema, por medio de los acuerdos necesarios entre los integrantes de las organizaciones. También se presentan los elementos que facilitarán la creación de dichas organizaciones y su ciclo de vida.

- ***Capítulo 5***. En este capítulo se describen en detalle los patrones de adaptación que se utilizan para diseñar el comportamiento adaptativo de las arquitecturas, así como también se explica la dinámica de su funcionamiento y el ciclo de vida visto con el lenguaje de patrones.

- ***Capítulo 6***. En este capítulo se presenta el caso de estudio para realizar las pruebas de concepto en dominios concretos, a fin de comprobar que el enfoque propuesto es viable.

- ***Capítulo 7***. En este último capítulo se exponen las conclusiones que se obtienen una vez finalizado este trabajo de tesis, las principales contribuciones de esta investigación y las futuras líneas de desarrollo que se podrían seguir.

- Asimismo se han agregado dos anexos: las conclusiones en idioma inglés y el listado de publicaciones científicas relacionadas con este trabajo.

Capítulo 2

Estado del Arte

En este Capítulo se presentan conceptos relacionados con esta tesis. En primer lugar, los referidos a agentes inteligentes y sistemas multi-agente, a continuación se analizan los problemas que se originan debido a la necesidad de *coordinación* en esos sistemas, y seguidamente se estudia la capacidad de adaptación de los mismos así como también la posibilidad de utilizar *patrones de arquitectura de software* para facilitar dicha adaptación. Diversos trabajos e investigaciones relacionados con estos conceptos acompañan cada tema tratado.

2.1. Sistemas Basados en Agentes

En los sistemas software, las tecnologías inteligentes de la información están relacionadas con las capacidades de aprender, razonar y administrar el conocimiento. El concepto de *agente* representa uno de sus enfoques más modernos. La evolución de este concepto ha llevado a considerar conjuntos de múltiples agentes con complejidad distribuida, conocidos como *Sistemas Multi-Agente - SMA*, o en inglés *Multi-Agent Systems - MAS*.

2.1.1. Agentes y Sistemas Multi-Agente

Agentes

La utilización de *agentes* como elemento computacional para resolver problemas no es nueva. Sus inicios se remontan, aproximadamente, a finales de la década de los años setenta y principios de los ochenta del siglo pasado [156]. Sin embargo su concepto, y posteriormente el de SMA, han tenido un notable crecimiento como campo de investigación desde mediados de los noventa. Estos sistemas software son cada vez más populares en el área de la inteligencia artificial (IA) para la resolución de problemas complejos.

Hasta la actualidad los expertos no han podido llegar a un acuerdo con la definición de *agente*, y por lo tanto el debate y la controversia al respecto continúan. Esta falta de consenso está señalada porque según el dominio que se trate podríamos considerar como *agente* (aunque no inteligente) desde el clásico ejemplo del termostato, un simple sistema de control que regula la temperatura de una habitación, hasta un complejo software de conducción de vehículos, que controla el automóvil en lugar del usuario. La utilización de *agentes* se extiende por numerosas áreas, por ejemplo, en el control de tráfico aéreo, en aplicaciones de medicina, para recuperación de información, en robótica, comercio electrónico, juegos, entre muchas otras.

Como se expresa en [73], la riqueza de la metáfora del *agente* y la posibilidad de tener diferentes usos es tanto una fortaleza como una debilidad. *Fortaleza* porque indica que puede ser aplicada de muchas maneras, en diversas situaciones y con distintos objetivos; y *debilidad* porque al ser usada frecuentemente no habría un concepto común aceptado por todos.

Existen numerosas definiciones del término *agente*, muchas de las cuales dependen del entorno donde son utilizados. A continuación presentamos algunas que pueden encontrarse en la literatura especializada:

Let us define an agent as a persistent software entity dedicated to a specific

purpose. "Persistent" distinguishes agents from subroutines; agents have their own ideas about how to accomplish tasks, their own agendas. "Specific purpose" distinguishes them from entire multifunction applications; agents are typically much smaller.
David C. Smith et al., 1994. [214]

Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed.
Pattie Maes, 1995. [150]

An agent can be a physical or virtual entity that can act, perceive its environment (in a partial way) and communicate with others, is autonomous and has skills to achieve its goals and tendencies.
Jacques Ferber, 1999. [87]

El concepto de "agente" caracteriza a una entidad software con una arquitectura robusta y adaptable que puede funcionar en distintos entornos o plataformas computacionales y es capaz de realizar de forma "inteligente" y autónoma distintos objetivos intercambiando información con otros agentes humanos o computacionales.
Francisco J. Garijo, 2002. [104]

An agent is a computer system capable of flexible autonomous action in a dynamic, unpredictable and open environment.
Mark d'Inverno y Michael Luck, 2004. [73]

An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objetives.
Michael Wooldridge, 2009. [236]

Si bien los párrafos anteriores brindan varios conceptos sobre agentes es importante determinar el ámbito relevante para esta tesis, aún a costa de no lograr una única definición. Es así que destacamos la última de ellas [236], la que nos permite extraer tres conceptos significativos: *contextualización (entorno)*, *autonomía* y *flexibilidad.*

Por *contextualización* podemos entender que el agente recibe información de su entorno y luego lleva a cabo ciertas acciones que podrían cambiar dicho entorno.

La *autonomía* se refiere a que los agentes pueden operar de manera directa, sin intervención de otros agentes (pudiendo incluirse entre estos a los humanos), y también con control sobre sus acciones y estado interno.

Por último, la *flexibilidad* viene dada por:

- *Reactividad*, al responder a los cambios de su entorno en un tiempo deseable;
- *Pro-actividad*, cuando los agentes tienen la capacidad de tomar la iniciativa para lograr sus objetivos; y
- Su *habilidad social*, para que puedan interactuar con otros agentes (y/o humanos) utilizando algun tipo de lenguaje de comunicación.

Otros atributos importantes que pueden presentar los agentes son el funcionamiento continuo; su robustez frente a fallos o excepciones; adaptabilidad ante distintos entornos; movilidad para desplazarse entre nodos de una red y su posterior ejecución; y razonamiento y aprendizaje [156]. La presencia de estas dos últimas características permiten que se los trate como entidades *inteligentes*, aunque queda claro que son los diseñadores y desarrolladores quienes les proveen de inteligencia y capacidad de trabajo coordinado.

De acuerdo con [239] podemos considerar a los *agentes* en un nivel superior de abstracción en el diseño y desarrollo de sistemas complejos, por sobre los objetos y los componentes.

Para una primera clasificación de tipos de agentes software podemos tener en cuenta la flexibilidad comentada anteriormente y considerarlos:

- *Reactivos:* cuando son modelos simples, careciendo de mecanismos de representación de conocimiento, razonamiento o aprendizaje. Funcionan recibiendo eventos del entorno y luego actúan según el evento de que se trate y su estado interno. [134].

- *Proactivos:* cuando tienen la capacidad de ver las oportunidades que les son favorables, tomar la iniciativa para llevar a cabo determinadas tareas y tratar de conseguir sus objetivos [163].

- *Sociales:* un escalón más arriba encontramos a los agentes con habilidades sociales. Por ejemplo estos pueden comunicarse con otros agentes por medio de envío y recepción de mensajes en un lenguaje común; también pueden cooperar trabajando en conjunto para el logro de los objetivos; o también por medio de negociaciones lo que implicaría compromisos, resolución de conflictos y alcanzar acuerdos con uno o más agentes [9].

Teniendo en cuenta otros criterios también se los puede clasificar de la siguiente manera [84, 156]:

- *Según sus características individuales:* podemos tener agentes *reactivos* que funcionan bajo un ciclo de percepción/acción (ya fueron descriptos en el párrafo anterior) [134]; agentes *cognitivos*, que son más complejos ya que para realizar sus tareas utilizan algún tipo de representación simbólica del conocimiento, su funcionamiento se basa en un ciclo de percepción, asimilación, razonamiento y actuación, e incluyen algún tipo de procesador de conocimiento [103].

- *Según su entorno:* por un lado tenemos agentes que requieren de un entorno especial o algún tipo de plataforma específica (por ejemplo los llamados "agentes FIPA" [93]); por otro lado se pueden utilizar agentes que no precisen de un entorno específico y puedan implementarse usando lenguajes independientes de plataforma [156].

- *Según sus interacciones:* o lo que es lo mismo, la manera de comunicarse e intercambiar información entre agentes. Podrían ser de tipo *agente-agente*, utilizando len-

guajes de comunicación tales como el KQML (Knowledge Query and Manipulation Language) [2]; tipo *agente-humano*, para lo que tendrían que existir las interfaces adecuadas; y por último, *agente-entorno*, permitiéndole intercambiar información con elementos del entorno que le rodea, por ejemplo con una base de conocimientos.

- *Según su modo de organizarse:* es decir teniendo en cuenta sus capacidades y tipo de estructura de trabajo. Podemos clasificarlos en *individuales* cuando no precisan de la colaboración de otros agentes, incluso en su arquitectura no cuentan con la capacidad de colaborar. *Cooperativos*, los que aparte de trabajar de manera individual también lo pueden hacer en colaboración con otros formando grupos de trabajo, es así como podemos destacar una cooperación implícita (por medio de reglas o mecanismos de control) y explícita (por medio de roles, planes comunes, etc.); también podemos encontrar agentes que tienen roles fijos o variables, con capacidad para resolver conflictos o de negociación. Y por último los *competitivos*, aquellos agentes individualistas que compiten con los otros, por ejemplo por los recursos limitados.

- *Según la utilidad para la que fueron creados:* podemos considerar dos criterios, por un lado el dominio de aplicación, y así se tienen agentes en el área de comercio electrónico, economía, defensa, ocio, entre muchas otras; por otro lado, se tienen agentes según el tipo de tareas que realizan dentro de esos dominios, especializados en ayuda inteligente, control de sistemas, mediación, búsqueda y recuperación de información, monitorización, etc.

Para no considerar a los agentes solo como entes abstractos y tener en cuenta por ejemplo su estructura interna o sus operaciones, es igualmente interesante hacer una descripción sintética de las arquitecturas concretas de agentes. Un estudio en mayor profundidad puede encontrarse en [225]. Podemos determinar cuatro tipos de agentes [235]:

- *Basados en lógica:*

  En estos se utilizan representaciones simbólicas, las *fórmulas lógicas*, y su manipulación sintáctica, la *deducción lógica*, en las tomas de decisiones. Tradicionalmente la IA simbólica utiliza este método para obtener un comportamiento inteligente en los agentes. Su estado viene dado por una base de datos de fórmulas de predicados

lógicos de primer orden, lo que representaría la información del entorno que tiene el agente. Por lo tanto su comportamiento queda determinado por las reglas de deducción y la base de datos.

Sin embargo existen algunas *desventajas* en este enfoque, por ejemplo para entornos dinámicos, complejos y posiblemente físicos la representación simbólica no es tan obvia, el razonamiento sobre información temporal resulta muy difícil, la toma de decisiones resultará efectiva solo si el entorno no cambia significativamente mientras el agente está decidiendo qué acción realizar, la que debería ser racional tanto al inicio del proceso de decisión como al final.

- *Reactivos:*

  En este enfoque los agentes reaccionan al entorno en el que están situados sin razonamiento acerca del mismo, desarrollan y combinan comportamientos individuales, sin representaciones simbólicas complejas ni razonamiento simbólico. Se considera que la toma de decisiones se realiza a través de un set de comportamientos que cumplen tareas, y que cada comportamiento es una función que toma las percepciones (entradas) y las correlaciona con acciones a ejecutar (salidas). La arquitectura más conocida es la de *subsumption*, desarrollada por R. Brooks [35]. Este autor propone una *categorización/jerarquía* de comportamientos para los casos en que varios de ellos puedan dispararse al mismo tiempo: los comportamientos se disponen en capas, cuanto más bajo nivel tenga la capa es más prioritaria, es decir que capas inferiores pueden inhibir a capas superiores.

  Como *ventajas* encontramos la simplicidad, flexibilidad computacional, robustez ante fallas, entre otras. Como *desventajas* o *problemas no resueltos:* los agentes deberían tener suficiente información de su entorno local para determinar una acción aceptable si no utilizan modelos de su ambiente; no es fácil ver cómo tomar decisiones con información no local si tienen un punto de vista a corto plazo; tampoco es fácil entender cómo los agentes reactivos puros pueden diseñarse para aprender por medio de la experiencia y a mejorar con el transcurso del tiempo; si se habla de un comportamiento que “emerge” de sus interacciones, este concepto implica que no es

comprensible la relación entre los comportamientos individuales, entorno y comportamiento global; al no haber una metodología para su construcción se deben utilizar procesos de experimentación muy laboriosos para diseñarlos; etc.

- *BDI (belief-desire-intention):*

  Estos agentes tienen un proceso de toma de decisiones bastante más complejo. Sus componentes básicos son: las creencias *(belief)*, deseos *(desire)* e intenciones *(intention)*; las funciones para deliberar; y el razonamiento de fines y medios.

  En el razonamiento se ven involucrados dos procesos: decidir qué objetivos se desean conseguir (la deliberación), y cómo conseguirlos (el razonamiento de fines y medios). El proceso de decisión comienza analizando las opciones disponibles y se genera un set de alternativas, se elige una que se convierte en su "intención" y que guiará sus acciones posteriores. El agente debe perseverar en esa intención escogida, pero puede rectificarla si ha cambiado el argumento por el que fue elegida o si no podrá cumplir con ella, sin embargo es importante que cuando la elija debe creer al menos que tiene chance para llevarla a cabo.

  Conseguir balancear sus diferentes intereses constituye una clave en el diseño de este tipo de agentes, ya que él debe reconsiderar sus intenciones de tiempo en tiempo, y esto tiene su costo en recursos computacionales y de tiempo. Por lo tanto habrá un dilema entre su comportamiento pro-activo y reactivo, el que estará muy condicionado por la dinámica del ambiente en el que trabaja [137].

  En este tipo de agentes el razonamiento práctico tiene mucha similitud con el que utilizamos en el día a día, proporciona una descomposición clara de sus funciones, lo que permite definir los subsistemas necesarios para construirlo. La principal *desventaja* de este modelo es justamente cómo desarrollar eficientemente estas funciones.

- *Con arquitectura de capas:*

  Las arquitecturas vistas anteriormente presentan limitaciones, es por ello que se han propuesto modelos "híbridos". El objetivo es que el agente tenga comportamientos tanto pro-activos como reactivos. Intuitivamente podríamos crear un par de capas (subsistemas) que traten ambos comportamientos respectivamente, aunque cuanta

más capas más útil es esta topología teniendo en cuenta el flujo de información y control que fluirían entre ellas.

Pueden ser capas *verticales* cuando solo una de ellas trata con las percepciones y las acciones (un ejemplo es el sistema InteRRaP [162]); o también capas *horizontales*, donde todas tienen acceso a sensores y actuadores, y asimismo cada una puede actuar como agente individual sugiriendo acciones a realizar (en [88] se describe el sistema Touring Machines). Para la mayoría de las arquitecturas bastan con tres niveles, de abajo hacia arriba: *reactivo*, para tomar decisiones en base a los eventos actuales del entorno; *conocimiento*, básicamente el conocimiento que tiene del entorno y generalmente utilizándose una representación simbólica; y por último, *social*, donde se tratan los aspectos sociales del agente, sus deseos, intenciones, etc. El comportamiento global, como es de prever, proviene de la interacción de estos niveles [156].

Un problema clave en esta arquitectura capas es la clase de control que se debe incluir en los agentes para gestionar las interacciones entre las capas. Con las verticales la cuestión viene por el lado de su flexibilidad, ya que para tomar una decisión el control debe pasar por cada capa, si hubiera una falla en alguna de ellas las consecuencias serían graves respecto del comportamiento del agente. Con las capas horizontales puede ocurrir que el comportamiento global del agente no sea coherente debido a que cada una compite con las otras en las sugerencias de acciones. Esto puede solucionarse utilizando una función de mediación para determinar qué capa toma el "control" del agente, lo que a su vez genera otras cuestiones, ya que al incorporar este control centralizado se debe tener en cuenta todas las posibles interacciones entre capas, además de producirse el problema de "cuello de botella" en la toma de decisiones [235].

Existen situaciones que pueden ser solucionadas apropiadamente con *un solo agente*. Tomemos como ejemplo una aplicación que filtre los correos electrónicos en lugar del usuario, el agente puede aprender a priorizar, eliminar, reenviar, ordenar y archivar los *emails* [149]. Sin embargo, cuando se enfrentan problemas muy complejos lo más habitual es que la solución deba tener en cuenta la interacción de *muchos agentes*, lo que puede ser más realista e interesante desde el punto de vista del diseño y desarrollo de sistemas. El

apartado siguiente trata de estos grupos de agentes trabajando en conjunto.

Sistemas Multi-Agente

En base a lo expuesto anteriormente es posible ver una evolución de conceptos, desde un *agente inteligente* complejo a conjuntos de múltiples agentes con complejidad distribuida: los ya citados *Sistemas Multi-Agente - SMA*. Estos son cada vez más utilizados en IA para resolver problemas con múltiples características y requerimientos muchas veces enfrentados. Por ejemplo, pueden tener múltiples métodos de resolución, distintos puntos de vista, incertidumbre, múltiples sujetos participantes, o también situaciones donde utilizar un método centralizado para resolver los problemas es muy difícil o prácticamente imposible.

Como cada trabajo que involucra SMA tiene su propia definición, para nuestro enfoque elegimos como base el artículo de Nicholas R. Jennings et al. [130], en el que se amplía la definición originalmente brindada por Durfee y Lesser en [75], y que dice lo siguiente:

> *A MAS can be defined as a loosely coupled network of problem solvers that interact to solve problems that are beyond the individual capabilities or knowledge of each problem solver. These problem solvers, often called "agents", are autonomous and can be heterogeneous in nature.*

Los SMA, por definición, implican numerosas interacciones entre los agentes que los componen. Estas interacciones pueden ser tanto cooperativas como competitivas (o individualistas), según si los agentes comparten un objetivo común o si solo persiguen sus propios intereses. Sin embargo, y más allá de que cada SMA dependerá del dominio en el que se va a aplicar, estos poseen características que son compartidas, por ejemplo, que los agentes que los integran tienen información incompleta para resolver por sí solos el problema (y por lo tanto un punto de vista limitado); la información se encuentra distribuida; no existe un control global o central; y la computación es de tipo asíncrona [218]. Asimismo los agentes pueden decidir de manera dinámica las tareas a realizar y también quien las va a llevar a cabo, aunque, usualmente, estas características se ven comprometidas por la

necesidad de lograr una coordinación interna.

En teoría, un SMA puede encontrar el balance entre las reglas estrictas que definen un sistema de razonamiento (un agente) y la naturaleza adaptativa de la evolución de una "sociedad" de individuos autónomos. Mientras el primero asume un mundo cerrado y una configuración estática, el segundo provee los medios para abrir el sistema de una manera flexible, definiendo una estructura más dinámica, capaz de hacer frente a la complejidad social.

Sin embargo, en la práctica, muchos SMA son menos flexible de lo que inicialmente se espera, ya que típicamente se los diseña para funcionar con determinadas configuraciones. A menudo se estabilizan en una estructura concreta durante su fase de entrenamiento, la que ha de permanecer inalterada para mantener sus capacidades de razonamiento y, por tanto, sin posibilidades de cambiar y evolucionar. La adaptabilidad de este tipo de sistemas frecuentemente está muy restringida, y su grado de apertura, limitado.

Desde este punto de vista, el enfoque SMA cumple parcialmente una promesa original, ya que no garantiza que el sistema sea en efecto adaptativo; de hecho, en ocasiones su propia estructura se opone a ello. De igual manera el entorno impredecible de la aplicación puede hacer que los agentes tengan fallas individuales, lo que reduce significativamente la habilidad de todo el sistema para llegar a cumplir su objetivo. Es decir que si hablamos de sistemas cerrados podríamos contar con un control centralizado y esperar que los agentes se comporten correctamente. Por el contrario, en entornos más dinámicos estaríamos en presencia de sistemas abiertos, con agentes desconocidos en un principio y/o que pueden ingresar o abandonar el sistema en cualquier momento.

Como los diferentes agentes en un SMA tienen sus propias agendas, solo una acción coordinada puede asegurar un comportamiento consistente. La necesidad de la coordinación interna es alta para conseguir los objetivos. La hipótesis más importante de este trabajo es que esta coordinación no debe ser provista por los propios agentes, sino por la estructura que los contiene (la "arquitectura" en sí). Los primeros pasos en esta dirección ya han sido

dados por distintas propuestas de SMA que proveen un camino intermedio en la forma de *organizaciones*. O lo que es lo mismo, grupos de agentes que comparten características o restricciones comunes. Los SMA pueden ser divididos en fracciones más pequeñas y con agendas diferentes. La idea principal es que los agentes en una organización son coordinados automáticamente, y por tanto, se eleva el nivel de abstracción. Las organizaciones aparecen como una estructura subyacente (a menudo jerárquica), lo que proporciona un medio flexible y de gran alcance para proveer la capacidad de adaptación. Este enfoque se ve con más detalle en 2.1.3.

La especificación *FIPA*

Entre los esfuerzos llevados a cabo para lograr una arquitectura estándar para el desarrollo de SMA es importante destacar a ***FIPA*** *(Foundation for Intelligent Physical Agents)* [93]. Esta fundación es una organización internacional de la IEEE[1] Computer Society y ha realizado extensos trabajos produciendo estándares para facilitar la interconexión e interacción de los sistemas de agentes, así como también su flexibilidad.

En sus especificaciones se definen las carácterísticas a ser cumplidas por las plataformas de gestión de SMA. Entre los principios más importantes en su definición figuran, por ejemplo, conseguir un SMA abierto de forma tal que distintos sistemas puedan interactuar a nivel de sociedades de agentes; definir solo el comportamiento externo o interfaz (las decisiones finales de diseño corresponderán al equipo de desarrollo); y también establecer un modelo lógico respecto de la creación, destrucción, registro, localización y comunicación entre agentes.

La Arquitectura Abstracta FIPA define a un nivel abstracto cómo dos agentes pueden encontrarse y comunicarse entre ellos, registrándose previamente e intercambiando mensajes. No puede ser implementada de manera directa, se trata de una base para la especificación concreta, la que describirá cómo construir SMA con sus respectivos lenguajes, interfaces, protocolos, etc.

[1] Institute of Electrical and Electronics Engineers

El elemento primordial del estándar es la plataforma de agentes, que proporciona infraestructura y servicios para desarrollar un SMA. Asimismo define los servicios que deben ser proporcionados por los SMA: un sistema de administración de agentes, el *Agent Management System*; un sistema para gestionar los mensajes entre los agentes, el *Message Transport System*; un servicio de directorio, el *Directory Facilitator*; y el canal para que los agentes se comuniquen, el *Agent Communication Channel*. Otro elemento importante definido es el lenguaje para que tenga lugar la comunicación entre agentes, el *ACL, Agent Communication Language* [94].

Cabe destacar que en este trabajo, y debido a la propuesta que se tratará en Capítulos posteriores, los agentes y los SMA son compatibles con el estándar FIPA.

2.1.2. Metodologías para el desarrollo de SMA

Para poder utilizar el paradigma de agentes y SMA en el desarrollo de sistemas complejos se hacen necesarios métodos, técnicas y herramientas que permitan realizar un trabajo sistemático y de manera ingenieril. De esto también depende, en gran parte, su aceptación por parte de la industria y la empresa, los que aún no han acogido a los SMA con gran expectativa [65, 230].

Queda claro que desde sus comienzos los SMA fueron considerados como grupos o equipos de agentes, sin embargo es importante destacar que en los últimos años la tendencia de los investigadores es tener en cuenta los aspectos organizativos y estructuras de esos grupos. Este enfoque será tratado con más detalle en 2.1.3.

La construcción de un SMA es una actividad compleja porque es necesario tener en cuenta las ventajas y desventajas de utilizar agentes en la resolución de problemas. El paradigma de agentes sirve como una herramienta para descomponer la complejidad del problema y atacar de manera más “fácil” los sistemas de gran tamaño, en un nivel de abstracción superior a los objetos y componentes de software [239].

Una vez realizada una descripción general de los SMA en párrafos precedentes, presentamos a continuación una síntesis de algunas de las metodologías más conocidas para

su desarrollo, con una visión más orientada hacia el agente, sus interacciones, sus estados y la relación entre estos y su comportamiento.

Gaia

Gaia [237] es una de las metodologías más conocidas y citadas en el desarrollo de SMA. Se la puede considerar como una metodología de "propósito general", ya que no especifica que deba utilizarse una tecnología de agente en concreto ni tampoco detalles de implementación. Permite el análisis y diseño de SMA considerándolos como organizaciones o sociedades artificiales de agentes. El elemento clave en Gaia es el *rol*, el que tiene asociado ciertas responsabilidades, permisos, actividades y protocolos.
La metodología permite desarrollar sistemas complejos con una estructura organizacional estática, donde el número de agentes que los componen no cambia en tiempo de ejecución. También las interacciones, habilidades y servicios ofrecidos por los agentes son estáticos. Se diferencian claramente las fases de *análisis* y de *diseño*, sin embargo la fase de captura de requerimientos es independiente de la metodología.
Los conceptos más importantes pueden ser de dos categorías: *abstracta*, cuyas entidades son utilizadas durante el análisis para conceptualizar el sistema (pero que no necesariamente tienen realización directa dentro de él); y *concreta*, cuyos elementos son usados en el proceso de diseño y usualmente tienen relación directa con elementos del sistema.
En la *fase de análisis* el objetivo es entender cómo funciona el sistema y su estructura (sin detalles de implementación). Las entidades abstractas, siguiendo una jerarquía, son el *sistema* y luego el *rol*. Este último se define por sus *responsabilidades*, que determinan su funcionalidad; sus *permisos*, los recursos de que se disponen; sus *actividades*, las tareas que un rol puede realizar sin tener que interactuar con otros agentes; y por último sus *protocolos*, los que definen la manera que puede interactuar con otros roles. En consecuencia tendremos dos modelos, el de *roles*, que los identifica; y el de *interacciones*, que describe protocolos entre roles y captura dependencias y relaciones.
En la *fase de diseño* el objetivo es transformar los modelos del análisis en abstracciones de un nivel en el que puedan ser implementados los agentes utilizando técnicas de diseño tradicionales. En esta fase se generan tres modelos: el *modelo de agente* que identifica sus

tipos y sus instancias, cada tipo de agente tiene asignado uno o más roles; el *modelo de servicios* que identifica los servicios asociados a los tipos de roles para determinar su funcionalidad, se basan en los protocolos, actividades y responsabilidades del rol; y el *modelo de conocidos*, que simplemente define los vínculos de comunicación que existen entre los tipos de agentes, no los mensajes sino el camino.
Una crítica a la metodología es que los modelos que produce tienen un nivel de detalle muy alto por lo que para implementar los sistemas se debe seguir profundizando en el diseño. Como ya se dijo, está considerada de "propósito general" y es una de las más conocidas y utilizadas. Existe una versión "ampliada" llamada *Gaia 2* [239]que hace énfasis en abstracciones organizacionales y en el desarrollo de SMA abiertos. Esta versión se verá con más detalle en 2.1.3.

Tropos

La metodología *Tropos* [111] se basa en dos ideas importantes. En primer lugar, el concepto del agente y sus ideas mentales relacionadas, como las metas y planes que son usados en todas las fases de desarrollo del sistema, desde el análisis inicial hasta la implementación. Y segundo, se cubren las etapas tempranas del análisis de requisitos, lo que permite comprender en profundidad el entorno en el que va a operar el sistema. Esta es una de las características que también la diferencia de otras metodologías [26].
Podemos distinguir cuatro fases: *análisis temprano de requisitos*, *análisis tardío de requisitos*, *diseño arquitectónico*, y *diseño detallado*.
En el *análisis temprano de requisitos*, por medio de diagramas se realiza el modelado de actores y sus intenciones. Las intenciones son modeladas como metas, las que eventualmente determinarán los requerimientos funcionales y no-funcionales del sistema. El modelado de actores nos permite identificar a los principales así como sus dependencias sociales en el dominio. En el diagrama de objetivos podemos analizar los objetivos del actor y sus planes para conseguirlos, también nos permite identificar los requisitos funcionales, no-funcionales y recursos necesarios para llevar adelante sus tareas.
En la fase de *análisis tardío de requisitos* se representa al sistema como uno o más actores participantes en el modelo de dependencia estratégica, junto con los otros actores que se

encuentran en el entorno del sistema. Asimismo se amplían los modelos y dependencias obtenidos en la primera fase, lo que facilita la captura de nuevos requerimientos funcionales y no-funcionales debido a la incorporación de los nuevos actores.
En la tercera fase, *diseño arquitectónico*, según el estilo de arquitectura que se haya decidido utilizar, se amplía el diagrama de actores, se identifican sus capacidades y se agrupan formando tipos de agentes. En *Tropos* se han definido estilos para aplicaciones distribuidas, dinámicas y cooperativas para guiar el diseño de la arquitectura del sistema. En el *diseño detallado* se especifica la estructura interna de los agentes, utilizando diagramas que describen sus capacidades, planes e interacciones con otros agentes.
Para la implementación con la metodología *Tropos* se utiliza la plataforma *JACK* [121], basada en la arquitectura BDI de agentes. También existe una versión "ampliada" donde se propone agrupar a los agentes teniendo en cuenta estructuras organizativas para favorecer la coordinación y la obtención de objetivos. Se verán más detalles en 2.1.3

MaSE

La metodología *MaSE - Multi-agent systems Software Engineering* [64] trata el ciclo completo de desarrollo del sistema, desde la descripción del problema hasta la implementación de la solución. Utiliza el paradigma orientado a objetos considerando a los agentes como una especialización de aquellos. Básicamente consiste en dos fases, la de *análisis* y la de *diseño*. Estas a su vez tienen tres y cuatros pasos respectivamente.
Fase de análisis. Primer paso: es la *captura de objetivos*, en la que se transforman los requerimientos del usuario en un conjunto de objetivos estructurados y de alto nivel del sistema; segundo paso: luego de definir estos objetivos se *aplican casos de uso* creándose los diagramas de casos de uso necesarios y los correspondientes diagramas de secuencia para ayudar en la identificación de roles y su forma de comunicación; tercer paso: la *refinación de roles*, a partir de los anteriores diagramas se derivan y se amplían los conjuntos de roles responsables de los objetivos y se definen las tareas asociadas para conseguir esos objetivos.
Fase de diseño. Primer paso: la *creación de clases de agentes* define las clases específicas de agentes para jugar los roles que se determinaron en la fase de análisis y las conversaciones

en las que participarán; segundo paso: en la *construcción de conversaciones* se describen a estas con mayor nivel de detalle, ellas son las que definen los protocolos de coordinación entre dos agentes; tercer paso: en el *ensamble de agentes* se define la arquitectura interna de los agentes; cuarto paso: el *diseño del sistema* involucra la configuración del sistema a implementar, se determinan la cantidad, tipos y ubicaciones de los agentes dentro del sistema, asimismo se toman decisiones sobre la implementación, como por ejemplo el lenguaje de programación.
MaSE no determina ninguna plataforma específica para la implementación, aunque sí ofrece una herramienta de soporte para todos los pasos del análisis y del diseño –*agentTool*– que además automatiza la transformación de los modelos de análisis en los de diseño. Esta versión de la metodología no provee un mecanismo para modelar las interacciones del SMA con el entorno, los sistemas que se obtienen poseen una organización estática por lo que el dinamismo se encuentra comprometido. Para suplir estas y otras carencias los autores continuaron investigando y proponen una "extensión" de la metodología llamada *O-MaSE* (ver 2.1.3), en la que, por ejemplo, se capturan conceptos organizacionales en metamodelos, permitiendo obtener niveles de abstracción superiores.

Prometheus

La metodología *Prometheus* [174] fue desarrollada con la meta de que pudiera ser utilizada tanto por profesionales de la industria como estudiantes no graduados sin mayores conocimientos en la tecnología de agentes para el desarrollo de sistemas de agentes inteligentes. En este sentido *Prometheus* es *detallado* y *completo* ya que cubre todas las actividades requeridas: provee soporte "de-inicio-a-fin", desde la especificación hasta el diseño detallado y la implementación. Consiste en tres fases: la *especificación del sistema*, el *diseño arquitectónico* y el *diseño detallado*.
La *especificación del sistema* se enfoca en la identificación de las funcionalidades básicas del sistema, así como las entradas desde el entorno (percepciones), las salidas (mecanismos para afectar al entorno, o acciones) y las fuentes importantes de datos compartidos.
En el *diseño arquitectónico* se utiliza la información de la fase anterior para determinar los agentes que contendrá el sistema, la estructura global de este, y la manera en que van

a interactuar los agentes. Para obtener los tipos de agentes se agrupan las funcionalidades tratando siempre de obtener agentes con alta cohesión y bajo acoplamiento. La información de alto nivel de cada agente estará contenido en un descriptor de agente. Este describe sus objetivos, su tiempo de vida, funcionalidades, cardinalidad, datos que lee y escribe, eventos a los que debería responder, sus acciones y también los otros tipos de agentes con los que puede interactuar. En esta fase también se obtiene un diagrama general del sistema que proporciona una visión global del mismo, indica relaciones y comunicación entre agentes, datos, y entradas y salidas. A partir de distintos escenarios se obtienen protocolos de interacción para representar la dinámica del sistema.
La fase de *diseño detallado* tiene en cuenta las características internas de cada agente y cómo llevará adelante sus tareas dentro del sistema. El foco está en la definición de capacidades (módulos dentro del agente), eventos internos, planes y estructuras de datos detallados. Para ello se utiliza un proceso de refinamiento progresivo de capacidades hasta que todas ellas se hayan definido.
La fase de implementación está soportada por el entorno *JACK Development Environment - JDE* [11] y por *Prometheus Design Tool - PDT* [1], el primero permite obtener los diagramas de la fase de diseño y el segundo facilita el diseño de entidades.
Esta metodología soporta principalmente el desarrollo de sistemas con agentes de tipo BDI, lo que, en palabras de sus autores, hace que la metodología no sea tan útil para sistemas con agentes con otra arquitectura o a la hora de desarrollar sistemas abiertos, pero los pasos iniciales son apropiados y facilitan el diseño de los SMA.

MASSIVE

En la metodología *MASSIVE - Multi-Agent SystemS Iterative View Engineering* [146] se realiza un iteración de vistas que se van refinando en cada ciclo. Cada vista representa un conjunto de características conceptualmente vinculadas, por lo que cada una de ellas es una proyección del modelo sobre una cuestión en particular. Se proponen siete vistas: *entorno, tareas, roles, interacciones, sociedad, arquitectura* y *sistema*.
En la *vista de tareas* se analizan los aspectos funcionales del sistema y se genera una jerarquía de las mismas, que luego es usada para determinar las capacidades básicas para

resolver problemas de las entidades del sistema final. También se definen y cuantifican los requerimientos no funcionales del sistema.
En la *vista del entorno* se analiza el entorno del sistema desde la perspectiva del agente y de los desarrolladores. Se tienen en cuenta ambas perspectivas porque usualmente difieren, los desarrolladores tienen un visión global del entorno mientras que el agente solo tiene un conocimiento local.
La *vista de roles* determina la unión de las funcionalidades de las entidades de acuerdo a las restricciones físicas del sistema. Un rol es una abstracción que vincula parte del dominio de la aplicación con el agente que resolverá el problema en cuestión. Es posible determinar también que un agente pueda tener uno o más roles.
En la *vista de interacciones* se describen las interacciones de los agentes, las que pueden ser no solo de comunicación sino también una forma generalizada de resolución de conflictos.
La *vista de sociedad* clasifica la sociedad pre-existente dentro del contexto organizacional del sistema, teniendo en cuenta que una sociedad es una colección estructurada de entidades que persiguen un fin común. Esta sociedad se desarrolla de manera consistente con respecto a los roles definidos.
La *vista arquitectónica* es una proyección del sistema final teniendo en cuenta los atributos estructurales con respecto al diseño. Se trata de ver la arquitectura del sistema como un todo y también de indicar la arquitectura de los agentes que se utilizarán.
En la *vista de sistema* se tratan los aspectos que afectan a varias de las otras vistas al mismo tiempo o al sistema como un todo, como es el caso de las interfaces que controlan las interacciones entre el sistema y los usuarios, la estrategia para manejar errores, la implementación del sistema una vez finalizado el desarrollo, etc.
En esta metodología se asumen agentes cooperativos y benevolentes, es decir que todos cooperan para llegar al objetivo final (no se presentan comportamientos individualistas), lo que tampoco facilita la creación de SMA abiertos.

Las metodologías presentadas resultan muy efectivas para la construcción de SMA según el tipo de problema a resolver. Como es natural existen muchas otras que además enfrentan los problemas complejos desde otros puntos de vista. Una de ellas es *MAS-CommonKADS* [128], metodología basada en *CommonKADS* [208] y en el paradigma de

orientación a objetos, que es utilizada para sistemas expertos y permite el desarrollo de la gestión del conocimiento; otra es *AAII* [138], que considera dos puntos de vista que se van refinando, uno externo para identificar agentes y sus interacciones, y uno interno para describir cada uno de estos agentes, los que se basan en el modelo *BDI* (ver 2.1.1).

También pueden encontrarse interesantes resúmenes, comparativas y análisis de plataformas de agentes en diversos artículos y documentos ya publicados [127, 220, 61, 26, 156, 116].

En los últimos tiempos se han desarrollado otras metodologías que ponen mayor énfasis en los aspectos organizacionales y estructurales de los SMA, por ejemplo para controlar la entrada y salida de los agentes del sistema cuando se tiene un SMA abierto. Algunas de ellas se tratan en la subsección siguiente.

2.1.3. Metodologías de desarrollo de SMA orientados a las organizaciones

Como se ha visto en la subsección anterior, en un entorno cerrado el comportamiento de los agentes puede controlarse en tiempo de diseño y resolver la eficiencia de los mismos representa uno de los problemas más importantes. Sin embargo hay determinadas características que sería deseable poder estudiar, por ejemplo tratar de prever el comportamiento global del sistema ante el dinamismo de los agentes al entrar y salir del sistema, tener en cuenta la heterogeneidad de los mismos, la formación de grupos con distintas metas, etc. Es así que en los últimos años se están utilizando *modelos organizativos* para el desarrollo de SMA de mayor tamaño y complejidad. Estos modelos se enfocan en perspectivas organizacionales y estructurales del sistema desde sus inicios y que van a acompañarlo durante el ciclo de desarrollo. Los conceptos de roles, grupos, interacciones, etc. van a estar ahora mucho más relacionados con la estructura de la organización, sus objetivos, normas, coordinación, etc. desde las etapas iniciales del desarrollo del sistema [12, 17].

Las *organizaciones de agentes* van a depender de cuán abiertos y heterogéneos sean los SMA, lo que a su vez genera nuevos requerimientos, como la integración de los puntos de

vista individual y organizacional y la adaptación dinámica de los modelos a los cambios organizativos y de entorno [71].

Algunas de estas metodologías son "ampliaciones" de otras existentes, como es el caso de *Gaia 2*, *Tropos*, etc. Sin embargo hay otras que fueron desarrolladas con la idea de las *estructuras organizacionales* desde sus inicios. A continuación se presentan aspectos relevantes de algunas de estas metodologías de desarrollo de SMA orientados a organizaciones.

Gaia v.2

La primera versión de Gaia (ver 2.1.2) se "extiende" en Gaia v.2 [239], en la que el concepto clave de *organización* es visto como algo más importante que una simple colección de roles. Se identifican abstracciones organizacionales que se usarán en las fases de análisis y diseño que en la primera versión no existían. Con este enfoque se pueden desarrollar SMA más avanzados, considerándolos sistemas abiertos.
Como primera abstracción agregada tenemos al *entorno* (mediante un *modelo de entorno*), que especificará todas las entidades y recursos con los que el sistema va a interactuar para llegar a su objetivo. Además de este modelo se agregan otros dos: el *modelo preliminar de roles*, que define capacidades iniciales y no impone una estructura organizacional específica (que se definirá en la siguiente fase); y el *modelo preliminar de interacción* entre los roles y que puede estar incompleto. Asimismo se utiliza un conjunto de *reglas organizacionales* que gobiernan el comportamiento global de la organización, su dinámica respecto a entradas y salidas de agentes, la seguridad relacionada, etc. Estos cuatro elementos forman parte de la *fase de análisis*, en la que se identifican los objetivos de la organización y su comportamiento general, esto también permite definir *suborganizaciones* a partir de la mayor.
La *fase de diseño* puede dividirse en dos: una de *diseño arquitectónico*, en la que se define la arquitectura general del sistema (su *estructura organizacional*) teniendo en cuenta y refinando los modelos de roles e interacciones preliminares, y también haciendo uso de catálogos de patrones organizacionales existentes. La segunda sub-fase es la de *diseño detallado* que, como en la primera versión, incluye la definición del modelo de agente, sus

tipos e instancias, y el modelo de servicios, o grupos de actividades que los agentes deberán proveer.

Tropos v.2

La versión ampliada de esta metodología [139] respeta las fases originales de desarrollo (ver 2.1.2), por ende aquí también tenemos la fase de análisis de requisitos iniciales, de análisis de requisitos tardíos, de diseño arquitectónico y de diseño detallado.
Para extender la versión original de la metodología los autores se basan en ideas propuestas por dos teorías que estudian las estructuras sociales, la *Teoría de la Organización* y la de *Alianzas Estratégicas*. La primera describe las estructura y diseño de una organización, mientras que la segunda modela las colaboraciones estratégicas de organizaciones independientes que llegan a un acuerdo para conseguir el objetivo final.
Ya desde la primera fase se tiene en cuenta el punto de vista organizacional para el desarrollo del SMA. Al definir el modelo de actores, sus objetivos y sus dependencias se decidirá qué modelos organizacionales se utilizarán dependiendo del problema.
De las teorías citadas se derivan varios estilos organizacionales, los que se definen como metaclases de estructuras organizacionales que ofrecen un conjunto de parámetros de diseño para coordinar la asignación de objetivos y procesos. Como ejemplos propuestos se tienen *structure-in-5, pyramid style, matrix, bidding style, joint-venture style, co-optation style*, etc. También se provee de patrones sociales de agentes, como por ejemplo *call-for-proposal, subscription, broker, wrapper, etc.*, los que serán integrados en las últimas fases del desarrollo a los estilos organizacionales seleccionados para el sistema.

O-MaSE

La metodología MaSE (ver 2.1.2) utiliza los conceptos relacionados con organizaciones, como las metas, roles y relaciones entre entidades, pero tiene ciertas limitaciones ya que, por ejemplo, los SMA tienen una estructura fija, no permite el trabajo de agentes heterogéneos en un entorno abierto, etc. Para desarrollar SMA dinámicos los autores proponen una extensión llamada *O-MaSE (Organisation-based MaSE)* [66].
Para que los grupos de agentes se adapten a su entorno determinando su propia organiza-

ción en tiempo de ejecución se utiliza un metamodelo que describe el conocimiento requerido para definir una organización y así hacer frente a esa dinámica. Para este metamodelo son definidos las *metas*, los *roles* y los *agentes*: a los agentes (incluso heterogéneos) se les asignan roles específicos para lograr metas organizacionales. Asimismo se agregan cuatro entidades adicionales: las *capacidades*, las *asignaciones*, las *políticas* y un *modelo de dominio*. Las capacidades son básicas para el proceso de determinar qué agentes pueden jugar ciertos roles y qué tan bien lo hacen; mientras que las políticas (de asignación, de comportamiento o de re-organización) restringen la asignación de agentes a los roles, lo que facilita controlar los estados de la organización. El modelo de dominio es un componente crítico ya que define la ontología que da soporte a la comunicación entre agentes y define sus políticas de comportamiento, especifica los elementos claves del entorno, sus relaciones y las operaciones que se pueden realizar con ellos.
También se incluyen otros conceptos como el refinamiento de las metas a nivel de sistema, la integración de capacidades y la habilidad de modelar sub-organizaciones, lo que permite lograr niveles de abstracciones más altos y complementar la noción de "agentes organizacionales" en el metamodelo. Esto último facilita la representación de jerarquías de organizaciones proveyendo flexibilidad y escalabilidad al metamodelo.
Así es posible desarrollar aplicaciones en las que es deseable un cierto nivel de control organizacional global, pero los agentes tienen una autonomía limitada. Específicamente, deben aceptar la asignación de las metas a cumplir y los roles a jugar para llegar al objetivo. Como herramienta se utiliza *agentTool III (aT^3)*, la que soporta las distintas etapas del desarrollo y también facilita la generación de código.

Electronic Institutions

Electronic Institutions [82, 212] es una metodología que propone representar instituciones reales con una contrapartida virtual o "electrónica", las *instituciones electrónicas*. Permite gestionar características relativas a SMA abiertos, como por ejemplo la heterogeneidad de los agentes, su control y reputación, los cambios sociales y estructurales, entre otras. Las instituciones electrónicas son desarrolladas como entidades que forman parte del entorno en el que interactúan los agentes. Uno de los dominios típicos para su uso es el

comercio electrónico.

Para representar a las instituciones electrónicas se utilizan cuatro conceptos básicos: un *marco de diálogo*, en el que se definen la ontología a ser utilizada en el sistema, los "actos de habla" (o ilocuciones) que intercambiarán los agentes, y los roles que van a participar en la institución electrónica; las *escenas*, formadas por grupos de agentes con diferentes roles y realizando una actividad están siempre reguladas por protocolos, los que determinan las formas de interacción y comunicación dentro de la sociedad de agentes; una *estructura performativa*, que define cuáles son las escenas de la institución electrónica y de qué manera los agentes se mueven entre ellas; y por último las *normas*, que definen los derechos, obligaciones y compromisos de los agentes, de tal manera que se obtenga un comportamiento regulado de las instituciones electrónicas.

Para realizar el desarrollo de los SMA se utilizan las herramientas *ISLANDER* [81], que provee los elementos centrales de la metodología, y *AMELI* [83], que es un *midleware* que entre otras cosas facilita la interacción de los agentes que forman la institución electrónica con otros externos a la misma que son representados por los *governor*, por lo tanto cada agente del exterior tendrá su agente *governor* correspondiente.

MOISE+

MOISE+ (Model of Organisation for multI-agent SystEms) [124] propone una metodología basada en la especificación de un modelo organizacional en el que se distinguen tres dimensiones importantes: *estructural, funcional* y *deóntica o normativa*.

En la *dimensión estructural* se especifican los roles, grupos y vínculos de la organización. La definición de roles establece que cuando un agente decide jugar algún rol en un grupo lo hace aceptando restricciones de comportamiento y relaciones referidas a dicho rol. Un grupo se especifica como un conjunto de relaciones y roles, asimismo es posible definir sub-grupos, restricciones de cardinalidad, herencia y compatibilidad.

La *dimensión funcional* describe cómo deberían ser conseguidas las metas globales colectivas de acuerdo a esquemas sociales, es decir, cómo estas metas se descomponen (en "planes"), se agrupan en conjuntos coherentes (en "misiones") y se distribuyen a los agentes, los que se comprometen a llevarlos adelante en su totalidad.

La *dimensión deóntica o normativa* es necesaria para que quede clara la semántica de las relaciones entre las dos primeras. Establece de manera explícita las normas que regulan las obligaciones, prohibiciones y permisos de los agentes que juegan ciertos roles en determinadas misiones y durante cierto tiempo, existiendo también sanciones [107] que pueden ser aplicadas según los casos.
Esta metodología propone utilizar la herramienta de código abierto *S-Moise+* [123] como soporte. Se trata de un *middleware* que facilita el modelado organizacional gestionando las comunicaciones, las organizaciones y los agentes que forman el SMA abierto.

ROMAS

La metodología *ROMAS (Regulated Open Multi-Agent Systems)* [99] propone guiar el análisis y diseño de SMA con características sociales, que sean de tipo abierto y que a la vez sean regulados. Sus fases han sido definidas para desarrollar los sistemas desde su objetivo global hasta la especificación del comportamiento de las entidades individuales que lo integran. La metodología hace énfasis en el análisis y descripción de los contextos normativos del sistema y sus entidades, en el conjunto de normas (permisos, obligaciones y prohibiciones) que regulan sus comportamientos, y en el set de contratos que formalizan la relación entre dichas entidades. Estos últimos pueden ser *acuerdos contractuales*, cuando involucran un intercambio de servicios y/o productos entre las entidades; o *contratos sociales*, si representan compromisos entre las que forman la estructura de la organización. Los agentes, roles y organizaciones son definidos por medio de una estructura social formal, basada en una arquitectura de SMA abiertos orientados a servicios. La metodología está soportada por un metamodelo formal y utiliza una herramienta dirigida por modelos, para los que se usan notaciones gráficas de otros trabajos, como es el caso de Gormas [15]. Las fases de desarrollo son cinco, no forman un proceso lineal rígido ya que pueden ser utilizadas de manera iterativa y se usan guías que permiten obtener documentos y diagramas. La primera fase es la de *descripción del sistema*: se analizan los requerimientos y objetivos globales, se realizan los correspondientes casos de usos, se refinan las metas, se estudia la viabilidad de la metodología, etc.
Otra es la de *descripción de la organización*, en la que se analiza la estructura de la or-

ganización, asimismo se utilizan las acciones y restricciones relacionados con objetivos obtenidos en la etapa anterior para la identificación y descripción de los roles del sistema.
La fase tres es la *descripción del contexto normativo*. Se analiza el contexto referido a las normas que regularán el sistema y los contratos sociales ajustarán el comportamiento de las entidades dentro del sistema.
La cuarta fase es la *descripción de actividad*, durante la cual se describen mediante instancias de modelos las tareas, servicios y protocolos identificados en las fases previas. Por ejemplo, se especifican los protocolos de negociación, ejecución y de resolución de conflictos para los contratos identificados anteriormente.
La última fase, *descripción de agentes*, comprende el análisis de los requerimientos de los agentes, de sus objetivos, la asociación de estos con los roles correspondientes del sistema, y también la validación de coherencias (por ejemplo del contexto normativo del agente y de este con el del sistema).
Una ventaja de la metodología es que basa su descripción en un método estándar de FIPA [232], lo que le permite ser utilizada en otros procesos de desarrollo de SMA.

Las metodologías presentadas resumen enfoques que utilizan modelos organizacionales para el desarrollo de SMA para, por un lado, tratar a los sistemas como "abiertos" regulando las actividades de los agentes, y por otro, utilizar las estructuras organizacionales para conseguir objetivos (globales) mayores que los individuales de cada integrante. Asimismo, los sistemas vistos como una organización, e incluso las organizaciones modeladas como unidades "dentro" de los sistemas, tienen un impacto importante en el rendimiento a corto y largo plazo, según las características de los agentes, sus metas y el entorno [120]. El enfoque organizacional facilita el uso de mecanismos "sociales" para regular la actividad autónoma de los agentes a efectos de lograr los objetivos globales [70].

Aparte de los anteriores, desde hace varios años se vienen desarrollando diversos modelos organizacionales, en su mayoría acompañando metodologías para la construcción de SMA. Por ejemplo tenemos los siguientes: *AALAADIN* [85], que fue uno de los primeros y sirvió de base para el modelo *AGR (Agent-Group-Role)* [86]; *ANEMONA* [110], que extiende la metodología *INGENIAS* [129] que a su vez se basó en los resultados obtenidos por

MESSAGE (Methodology of Engineering Systems of Software Agents) [96]; *SADDE (Social Agents Design Driven by Equations)* [213]; *ROADMAP* [219], que extiende y mejora Gaia v.2 (2.1.3); *OperA (Organizations per Agents)* [69], que sirvió de fundamento para *OMNI (Organizational Model for Normative Institutions)* [72]; *SODA (Societies in Open and Distributed Agent Spaces)* [159]; entre muchos otros.

Sus diferencias radican en que algunos modelos son más específicos que otros a la hora de resolver problemas cada vez más complejos y cercanos a la realidad; unos admiten la participación de agentes no cooperativos y otros no lo permiten, al no centrarse, por ejemplo, en la definición de mecanismos de control del comportamiento de los agentes; algunas metodologías detallan en mejor medida la estructura de las organizaciones por medio de las definiciones de los roles y los grupos, aunque, por ejemplo, los agentes no estan habilitados para modificar las organizaciones definidas; entre otras características.

Además de los resúmenes y comparativas propuestos en 2.1.2, existen otras publicaciones que presentan, analizan y comparan otras metodologías y enfoques orientados a las organizaciones; por citar algunos [16, 54, 120, 153, 33, 55, 70]. En [3] puede encontrarse un análisis de la ingenería de software orientada a agentes (incluidas las centradas en organizaciones) con interesantes propuestas para un mayor acercamiento del paradigma a la empresa e industria.
En la sección 2.2 se tratarán SMA que presentan capacidades de adaptación, utilizando por ejemplo algunos mecanismos de re-organización y auto-organización.

2.2. Adaptación en SMA

Recordando lo dicho en 2.1.1, un SMA es un sistema descentralizado de agentes, con poco acoplamiento y que interactúan entre sí para resolver problemas cuya complejidad va más allá de su capacidad o conocimiento individuales. Estos agentes son individuos autónomos y pueden ser de naturaleza heterogénea. En las subsecciones 2.1.2 y 2.1.3 se han dado características de varios desarrollos de SMA, sin embargo varios de ellos proponen

enfoques cerrados, es decir que los sistemas son creados con estructuras y objetivos fijos.

Queda claro que los problemas tienden a ser cada vez más complejos, por lo que una manera de resolverlos es utilizar SMA abiertos en los cuales los agentes pueden incorporarse y abandonar dinámicamente el sistema. Como no necesariamente comparten las mismas metas, la funcionalidad global del sistema resulta más de las interacciones de los agentes que de sus capacidades individuales.

La adaptación en los sistemas, vista como la capacidad de ajustar su comportamiento en respuesta a la percepción de la dinámica del entorno y del sistema en sí mismo, ha adquirido mucha relevancia como tópico de investigación en los últimos años. Los SMA, al no presentar características centralizadas y estar compuestos por individuos autónomos (los agentes), se caracterizan por tener cualidades interesantes, entre ellas la escalabilidad y la posibilidad de lograr adaptación, lo que motiva la utilización de esta tecnología ante la complejidad y el dinamismo del entorno.

De acuerdo con [27] podemos diferenciar adaptación *débil* y adaptación *fuerte* en los SMA. La primera viene dada por las características intrínsecas de los agentes, como su autonomía, proactividad, habilidad social, planificación, etc.; mientras que la segunda tiene en cuenta eventos impredecibles que hacen que los SMA reaccionen ante entornos dinámicos para lograr su objetivo, por lo que representa un desafío más importante. Para este caso (*strong adaptation*), en [42] y [113] se propone la teoría ***AMAS*** (Adaptive Multi-Agent System), que se enfoca en el uso de sistemas auto-organizados para problemas complejos. Básicamente provee una solución para construir sistemas abiertos y complejos, en los que todas sus interacciones con el entorno no pueden ser totalmente previstas, por lo que el sistema debe adaptarse a eventos impredecibles. La función global del sistema emerge desde el comportamiento cooperativo de los agentes que lo componen. Cada agente interno solo persigue su meta individual e interactúa con agentes que conoce respetando técnicas cooperativas que le llevan a evitar situaciones impredecibles (aquí llamadas “Non Cooperative Situations” - NCS). Para enfrentar un NCS un agente cooperativo actúa de tal manera que llega a un nuevo estado cooperativo y permanentemente se adapta a esas situaciones mientras aprende de los demás. Las interacciones dependen de su visión local y de su habilidad para cooperar entre sí. El cambio en estas interacciones locales reorganiza el sistema y por lo tanto cambia su comportamiento global [26].

ADELFE

La teoría AMAS ha sido aplicada con éxito a numerosos proyectos, como por ejemplo a una herramienta para pronósticos adaptativos de inundaciones, o a otra de comercio electrónico para la mediación de servicios. También es tomada como base para la metodología *ADELFE (Atelier de DEveloppement de Logiciels à Fonctionnalité Emergente)* [27]. Esta metodología sigue el proceso RUP (Rational Unified Process) [142] de desarrollo de sistemas y también utiliza UML y AUML [39]. El proceso ha sido expresado utilizando SPEM (Software Process Engineering Metamodel) de OMG [112], que consiste en seis definiciones de trabajos (WD): requerimientos preliminares, requerimientos finales, análisis, diseño, implementación y pruebas. Cada definición de trabajo (WD) tiene actividades asociadas (A), y estas a su vez constan de determinados pasos (S). Al proceso clásico se han agregado algunos pasos para hacerlo específico para SMA adaptativos, especialmente aquellos concernientes a la cooperación. Por ejemplo, caracterización del entorno, con definición de contexto y entidades activas y pasivas con respecto al sistema, también de las entidades potencialmente cooperativas, identificación de "fallas cooperativas" (las NCS nombradas anteriormente), entre otros. No es una metodología de propósito general pero sí que se enfoca en aplicaciones que requieren de un diseño de SMA adaptativos, asimismo ofrece guías para identificar las áreas de aplicaciones que mejor se ajustan a ella por medio de una herramienta de soporte a decisiones. También permite el uso de otras dos herramientas, una basada en *OpenTool* [62] y otra interactiva que soporta el proceso y ayuda a los diseñadores a seguirlo y ejecutar las tareas asociadas.

EANN

Otro enfoque interesante encontramos en [233] y que se inspira en el comportamiento humano, la sicología y la ciencia neuronal. Se presenta una arquitectura de agentes en cuatro niveles que combinan dos formas fundamentales de adaptación: aprendizaje y evolución. Cada capa representa una funcionalidad que hace que el agente esté mejor preparado para enfrentar el dinamismo del entorno. El agente evoluciona de tal manera que la acción requerida es identificada por aprendizaje en lugar de estar pre-definida o planificada. Las capas son: *herencia, entrenamiento, experiencia* y (lo) *inesperado*. El *framework* concep-

tual utilizado se implementa como una red neuronal artificial evolutiva - *EANN (Evolutionary Artificial Neural Network)*. Los sensores de entrada, la capa de lógica difusa y la capa GSOM (Growing Self-Organising Map) [4], una red neuronal no supervisada, autoorganizada y evolutiva, pueden ser considerados como el módulo cognitivo que captura y clasifica repetidamente en el entorno real. Asimismo, GSOM junto con otras redes neuronales de retropropagación - BPNN (Back Propagation Neural Networks) [119] forman el módulo de aprendizaje que acumula experiencia de manera progresiva y permite que el agente realice la acción más efectiva. Ambos módulos trabajando en conjunto hacen que el agente sea capaz de reaccionar ante el dinamismo del entorno [233].

RADAR

El *Reflective Agents with Distributive Adaptive Reasoning* [105] es un proyecto de la Agencia DARPA[2] de EEUU que desarrolló una aplicación adaptativa de un asistente personal cognitivo. Sus características claves incluyen (i) extensibilidad a través del uso de una arquitectura *plug-in* de agentes, (ii) una integración transparente con aplicaciones *legacy* y datos de entornos de escritorio actuales, y (iii) el uso extensivo de la capacidad de aprendizaje, por lo que el entorno se adapta al usuario en el tiempo. Básicamente está construida por sobre arquitecturas de sistemas distribuidos y orientados a agentes, y el *framework* provee la integración de "especialistas", los que pueden agregarse o removerse en cualquier momento y que aumentan la habilidad del usuario para realizar sus tareas (a veces complejas), como por ejemplo la gestión de sistemas de archivo, automatización de tareas rutinarias, etc. La implementación está basada en capas, la de más bajo nivel consiste en un *middleware* estándar proveedor de servicios para sistemas distribuidos, la siguiente es una arquitectura de agentes compatibles con *FIPA* [93], la siguiente es una capa específica para la comunicación entre los "especialistas" y los servicios de gestión de tareas, y en la de nivel más alto se tienen los componentes de RADAR, por ejemplo extractores y categorizadores que entienden términos de lenguaje general e información específica de las tareas, una base de conocimiento, o integradores para la comunicación con otras aplicaciones, entre otros.

[2]DARPA: Defense Advanced Research Projects Agency (Agencia de Proyectos de Investigación Avanzados de Defensa), Estados Unidos de Norteamérica.

Swarming Systems

El enfoque propuesto por los llamados ***Swarming systems*** (o "sistemas de enjambre") [221] se basa típicamente en una población de elementos individuales –en nuestro caso los *agentes*– que interactúan localmente entres sí y con su medio ambiente. En la publicación citada también se define a *swarming* como "auto-organización útil de múltiples entidades a través de interacciones locales". Como ejemplo, en [60] se tiene un SMA para selección de tareas con algoritmos dinámicos y adaptativos de subasta. En este trabajo se plantea un dominio de problema de tipo "buscar-y-ejecutar", en el que las unidades *swarm* tienen que buscar y descubrir individualmente objetos de interés pero se requiere que colaboren entre sí para llevar adelante las acciones sobre esos objetos. Es así que una tarea solo puede ser completada si múltiples agentes comparten sus recursos computacionales. Cada unidad es modelada como un robot móvil con un agente de software embebido en él. Para que se produzca el comportamiento tipo "enjambre" se simula la *estigmercia* (en inglés *stigmercy*) de la actividad social de insectos –como las hormigas– que colaboran a través del medio físico. De esta manera se consigue que entre las unidades emerja una auto-adaptación para realizar las tareas incompletas. Algo similar se propone en Brutschy et al. [37], una asignación auto-organizada de tareas en un "enjambre" de robots. Su método está basado en el "retraso" de los robots que trabajan en una sub-tarea mientras esperan los resultados de otra sub-tarea.

IBM - Infection-Based Mechanism

El trabajo de Salazar et al. [204] propone en su enfoque adaptativo los ***mecanismos basados en infecciones*** *(IBM, infection-based mechanisms)*. Estos mecanismos evolutivos computacionales de auto-adaptación les facilitan a los agentes una evolución de manera distribuida de su comportamiento social para alcanzar las mejores convenciones sociales. Se basan en el fenómeno social de contagio para explotar la noción de *infección positiva*: los agentes con *buenos comportamientos* se vuelven *contagiosos* y propagan estos comportamientos a la sociedad de agentes como si se tratara de una enfermedad infecciosa. Para prever el estancamiento también se incorporan mecanismos innovadores que permiten que los agentes exploren comportamientos de manera constante con la esperanza de encontrar

los mejores. Combinando la *infección* con la *innovación* el modelo computacional ayuda a los SMA abiertos a establecer mejores convenciones aún cuando unas menos eficientes están muy establecidas en la sociedad. En [205] se utilizan los *IBM* para evaluar espacios de convenciones mucho mayores, analizar la comunicación emergente y lograr consenso en un léxico simple y globalmente compartido, a pesar de los cambios dinámicos y de la topología de interacciones.

Además de los casos presentados, numerosos trabajos también plantean adaptación en SMA. Por ejemplo, en [18] se utilizan protocolos para adaptar las normas que deben seguir los agentes a fin de lograr coordinación. Un modelo de planificación para facilitar la auto-adaptación en una sociedad de agentes (vista como una organización virtual) es presentado en [201], en el que se emplean agentes *CBP-BDI* (Case-Based Planning - BDI agents) [199]. Si tenemos en cuenta mecanismos basados en interacciones, encontramos que en [238] se utilizan los de tipo directo, como localización, interacciones y computaciones locales entre agentes para cambiar la estructura de la organizaciones. En [192] los autores proponen mecanismos indirectos de interacción entre agentes debidos a cambios en el entorno para lograr comportamientos complejos del sistema.

Otros trabajos también se enfocan en mecanismos basados en el orden social obtenidos a través del uso de normas sociales en comunidades virtuales [185], mientras que otros se centran en la reorganización de los agentes como un proceso asociado al dominio, donde las organizaciones son entidades de primera clase, modeladas separadamente de los agentes y su dinámica está gobernada por leyes [229, 228]. En [41] se plantea utilizar composición, cuando dos agentes se unen y forman uno solo, o descomposición, cuando uno de ellos se divide en dos, como método para lograr adaptación ante las demandas del entorno. Si tenemos en cuenta que los principios de los sistemas autonómicos pueden ser asociados con la noción de agentes, y ya que estos son autónomos por definición, se pueden conseguir SMA con la capacidad de adaptación inherente [63]. También es posible lograr que los agentes modifiquen y/o adapten su comportamiento con mecanismos de refuerzos, por ejemplo premios y castigos para aumentar o disminuir su comportamiento, respectivamente [179]; o incluso utilizando *políticas* que pueden ir cambiando en tiempo de ejecución, como es el caso de la aplicación de políticas a agentes en [74]. Otras propuestas destacables

son la de Marrow et al. [154], *DIET - Decentralised Information Ecosystem Technologies* que consiste en un ecosistema de agentes robusto, adaptativo y escalable que permite la exploración y manipulación de fuentes de información distribuidas; o el proyecto *ALIVE* [6], que combina mecanismos de coordinación y organización con diseño dirigido por modelos en un *framework* para el desarrollo de sistemas abiertos de servicios activos.

Como se ha visto en los párrafos anteriores, los SMA representan un importante campo de investigación en el área de sistemas adaptativos. En [115, 43, 67, 224, 200, 5, 122, 184, 227] podemos encontrar interesantes propuestas acompañadas de comparaciones, revisiones y otros enfoques referidos a mecanismos de interacción, auto-organización, coordinación, etc.

2.3. Sistemas Adaptativos y Auto-adaptativos

Conforme crece la complejidad del contexto computacional en el que los sistemas deben interactuar, tanto con otros sistemas como con humanos, se hace necesario el desarrollo de sistemas que tengan la posibilidad de ajustar su comportamiento en respuesta a la percepción de su entorno y del sistema en sí mismo, o que autónomamente puedan adaptar cambios en sus condiciones de operación. Podemos partir considerando que un sistema será adaptativo si se logra su reconfiguración para hacer frente a esos cambios; y si además ajustan su comportamiento con poca o nula intervención humana, estaremos hablando de la "auto-adaptación", que es un concepto mucho más significativo [157]. Como es deseable, esta adaptación (y auto-adaptación) de sistemas debe producirse sin perder de vista la corrección, robustez y eficiencia de los mismos.

Numerosos enfoques se han propuesto en los últimos años para el desarrollo de este tipo de sistemas. Las principales motivaciones van, por ejemplo, desde reducir el trabajo de mantenimiento de los mismos hasta los cambios de entorno y de necesidades de los usuarios, mientras se espera que continúen con su normal funcionamiento. Han sido estudiados desde diferentes puntos de vista, incluyendo ingeniería de requisitos, arquitectura

de software, *middleware*, desarrollo de componentes, teoría de control, robótica, sistemas distribuidos, computación autónoma, entre muchos otros.

De acuerdo con [36], los sistemas auto-adaptativos pueden ser caracterizados según *cómo funcionan* o según *cómo sean analizados*, y también por ciertas propiedades, por ejemplo si son *centralizados* o *descentralizados*, *top-down* o *bottom-up*, si la *incertidumbre* respecto del entorno es alta o baja, etc. Un sistema *top-dowm* es generalmente centralizado y su funcionamiento se ve guiado por un control central o por ciertas políticas, evalúa su propio comportamiento y se adapta si el monitoreo y el análisis se lo garantizan. Por el contrario un sistema *botton-up* es generalmente descentralizado, por lo que no tienen un control central, sus componentes interactúan localmente según reglas simples y el comportamiento global emerge de estas interacciones. La mayoría de los sistemas adaptativos pueden encuadrarse entre estos extremos, e incluso pueden presentarse combinaciones de ambos [36].

Las propuestas para tratar la adaptación y auto-adaptación parten desde distintos puntos de vista. Los primeros enfoques estaban muy acopladas con el código fuente y eran específicas para las respectivas aplicaciones, por lo que resultaban complicadas de desarrollar y posteriormente de mantener [44]; también tenemos las técnicas basadas en la arquitectura de los sistemas [168], donde un ejemplo es separar la lógica de la adaptación de la funcionalidad del sistema [23]; otro punto de vista es la utilización de la computación evolutiva [76]; mientras que otros desarrollos se enfocan en la ampliación del *middleware* con el fin de proveer servicios de adaptación para sistemas distribuidos [140]. Algunas propuestas son resumidas a continuación.

Reflection

La adaptación a través del uso de la *reflexión* tiene numerosos enfoques, tanto a nivel conceptual como tecnológico. Podemos definir *Reflexión Computacional* [56, 58] como la capacidad de un sistema (computacional) para razonar y actuar sobre sí mismo. Para ello un sistema reflexivo debe contar con un modelo interno que lo represente, y este modelo

esté causalmente conectado con el sistema. Esto significa que cada cambio en el modelo se reflejará en el sistema, y viceversa. Esta característica hace posible que se ajuste a las condiciones cambiantes, lo que provee al sistema de alta flexibilidad. Cuando un sistema actúa sobre el modelo de otro sistema (al que se llama sistema *base*) se lo trata como *meta-sistema* del anterior.

Un sistema es llamado *reflexivo* cuando actúa como su propio meta-sistema. Cuenta con dos puntos de vista de sí mismo, llamados nivel *base* y *meta-nivel*. El primero describe la operación normal y la estructura del sistema, mientras que el segundo describe cómo el sistema se percibe y/o se modifica a sí mismo. La operación que lo "lleva" del nivel *base* al *meta-nivel* es conocida como *reificación*[3], que puede ser vista como una relación que exprese una conexión causal entre ambos niveles. Cuando un elemento "base" es *reificado* y alterado en el *meta-nivel*, sus cambios deben ser reflejados en el nivel *base*, lo que representa la *reflexión*.

Aspect Oriented Systems

Los sistemas software desarrollados utilizando el paradigma de programación orientada a aspectos *(AOP - Aspect Oriented Programming)* también presentan capacidades de adaptación [25, 59]. Usualmente existen cuestiones comunes que suelen estar diseminadas en todo el código del sistema, son los llamados *crosscutting-concerns*. Lo que AOP propone es su separación (y encapsulación) en otras entidades, los *aspectos*. Esta separación evita el "enmarañamiento" del código y permite la reutilización del mismo *aspecto* en diferentes unidades de software (componentes, módulos, etc.). Es decir que no solo desacopla las cuestiones transversales de la lógica funcional sino que también las localiza. Esto permite obtener sistemas software con estructuras que se pueden desarrollar, ampliar, adaptar, etc. de manera más sencilla.

Como dijimos, los *aspectos* son unidades que implementan cuestiones como la persistencia, sincronización, seguridad, etc. Estas unidades deben interactuar con el resto del código del sistema por lo que los "tejedores" *(weavers)* realizan la mezcla en los llamados "puntos

[3] De la palabra en inglés *reification* que deriva del verbo *reify*: construir algo concreto partiendo de algo más abstracto.

de corte" *(point-cuts)*. Los *weavers* pueden, según el caso, coordinar los códigos de manera *estática*, en tiempo de compilación, o *dinámica*, en tiempo de ejecución. Esta última podría llevarse a cabo con un aspecto que controle la configuración del sistema y que proporcione los servicios para modificarla, y de esta forma facilitar la reconfiguración dinámica buscada [53].

Rainbow

En el enfoque propuesto en *Rainbow* [47] se utiliza el modelo arquitectónico del sistema para monitorearlo y razonar acerca de las acciones apropiadas que deben realizarse. Los mecanismos de monitoreo, las "sondas" (*probes*) y los "medidores" (*gauges*), miran al sistema en funcionamiento, y estas observaciones se relacionan con las propiedades del modelo de la arquitectura. Este es evaluado periódicamente para asegurar que el sistema funciona dentro de rangos aceptables, y si esta evaluación señala lo contrario se dispara el proceso de adaptación determinándose la acción que debe llevarse a cabo. La adaptación se ejecuta vía *efectores* mientras el sistema está funcionando. El *framework* consiste en dos partes, una infraestructura genérica reutilizable de mecanismos de adaptación comunes que los desarrolladores pueden aplicar a diferentes tipos de sistemas; y un conjunto de puntos explícitamente definidos de personalización que permite especificar la infraestructura "a medida" de una determinada clase de sistema. El principio subyacente es separar los mecanismos de las políticas, es decir, la parte que realiza el trabajo de adaptación de las instrucciones para hacerla [48].

Autonomic Computing

La *computación autonómica* [135] da lugar a sistemas software que pueden gestionarse a sí mismos en base a los objetivos de alto nivel impuestos por los administradores (humanos) con poca o casi nula intervención de estos últimos [161]. La compañía IBM (International Business Machines Corporation) comenzó esta iniciativa en el año 2002 y es una de las más interesantes para enfrentar la creciente complejidad de los sistemas debido a su tamaño, heterogeneidad, conectividad, interacciones, etc.
La esencia de la computación autonómica está en la auto-administración *(self-management)*,

con la intención de liberar a los administradores de sistemas de los detalles en la operación y mantenimiento de los mismos, y de esa manera proveer a los usuarios de un sistema que funcione de manera constante y correcta. En base a esto, cuatro aspectos son muy relevantes: la auto-configuración *(self-configuration)*, de manera que la configuración automatizada de los componentes y los sistemas responda a políticas de alto nivel para responder dinámicamente a los cambios; la auto-optimización *(self-optimization)*, para que se busque continuamente las oportunidades para mejorar su eficiencia y rendimiento; la auto-curación *(self-healing)*, para detectar, diagnosticar y reparar automáticamente los posibles problemas internos; y la auto-protección *(self-protection)*, para así proteger al sistema de cualquier ataque.
Estos sistemas, desde el punto de vista arquitectónico, están compuestos por colecciones de elementos autonómicos que interactúan y que a su vez poseen sus propios gestores, también autonómicos. Estos últimos, por medio de un ciclo de *monitorio, análisis, planificación* y *ejecución (MAPE)*, son los encargados de controlarlos y representarlos, lo que a su vez también les permite ser proactivos en los procesos correspondientes a los aspectos antes citados. Una infraestructura distribuida y orientada a servicios sería apropiada para dar soporte a estos elementos autonómicos y sus interacciones [125].

Aura

Aura [216, 106] *(Distraction-free Ubiquitous Computing)* es un *framework* arquitectónico para usuarios móviles en un entorno de computación ubicua. Idealmente una infraestructura de computación ubicua permite a los usuarios cambiar sus tareas computacionales fácilmente de un entorno a otro y tener la posibilidad de aprovechar las capacidades locales y los recursos dentro del entorno, aún con la constante entrada y salida de otros usuarios y dispositivos. El enfoque de Aura es novedoso porque crea modelos independientes de aplicación y de alto nivel de las tareas del usuario, y además utiliza estos modelos para la configuración y adaptación de los sistemas ubicuos.
Ataca estos problemas por dos frentes: por un lado le permite al usuario preservar la continuidad de su trabajo ajustando sus tareas a los recursos disponibles, y por otro, adaptando la continua computación de un entorno particular frente a la dinámica variabilidad de los

recursos del entorno. Los elementos claves del *framework* arquitectónico: la representación explícita de las tareas del usuario como una colección de servicios; la observación del contexto que permite que las tareas sean configuradas de manera que sean las apropiadas para el entorno dado; y la gestión del entorno, que ayuda con el monitoreo y la adaptación buscada. Cada una de estas capacidades es encapsulada en un componente de la arquitectura: el gestor de tareas, el observador del contexto y el gestor del entorno, respectivamente. Los servicios necesarios para soportar las tareas de un usuario son implementados por un conjunto de componentes llamados "proveedores de servicios", los que usualmente son *wrappers* de aplicaciones y servicios más tradicionales. Las interacciones entre las partes están implementadas por conectores explícitos que ocultan los detalles de la distribución y heterogeneidad de los proveedores de servicios. Aura se enfoca en minimizar las distracciones en la atención del usuario a sus tareas, por lo que "crea" un entorno que se adapta a su contexto y necesidades.

Como es natural existen numerosas propuestas más y la atención que despierta esta área de investigación ha aumentado en los últimos años, tanto en el campo académico como en el de la industria. Prueba de ello son las conferencias y workshops internacionales que se realizan periódicamente, por ejemplo las actuales *SEAMS* (International Workshop on Software Engineering for Adaptive and Self-Managing Systems), *SASO* (International Conference on Self-Adaptive and Self-Organizing Systems) y *CAC* (The ACM Cloud and Autonomic Computing Conference). También fue tópico de otras anteriores, como *SOAR* (Self-Organizing Achitectures), *WOSS* (Workshop on Self-managed Systems), *CHIACS* (Conference on the Human Impact and Application of Autonomic Computing Systems), *ICAC* (International Conference on Autonomic Computing), *FOSE* (The Future of Software Engineering Symposium), *IWSAS* (International Workshop on Self-Adaptive Software), entre otras.

Otros trabajos que se pueden citar: *MADAM* [158], donde sus autores representan los modelos arquitectónicos, en tiempo de ejecución, de aplicaciones móviles para que los componentes genéricos del *middleware* puedan adaptarse; en [30] se propone un modo mixto de adaptación, por lo que las interacciones entre el sistema *viejo* y el sistema *nuevo*

se permiten durante el proceso de adaptación; los autores de [240] proponen un proceso de desarrollo de sistemas dinámicos y adaptativos dirigidos por modelos, enfocado en los modelos de comportamiento; otro enfoque de diseño y adaptación de sistemas software basados en modelos es el propuesto en [114], en particular de arquitecturas de software dinámicamente reconfigurables y evolutivas consideradas como líneas de productos.

SELFMAN [223] hace uso de dos importantes áreas: computación autónoma y redes superpuestas estructuradas *(structured overlay networks)*, para la construcción de sistemas auto-administrados; en [136] se propone un método para planificar adaptaciones dinámicas basadas en el aprendizaje reforzado de sistemas auto-administrados; en el area de trabajo con sistemas auto-organizados, [109] propone el enfoque donde unidades auto-administradas se coordinan hacia un modelo común, una estructura arquitectónica definida utilizando formalismos arquitectónicos de Darwin [151], conocido lenguaje de descripción de arquitecturas.

Otro trabajo es *CASA* [160], que propone el uso de un motor externo de adaptación para proveer de acciones adaptativas tanto a bajo nivel como a nivel de aplicación; los autores de [68] proponen que aplicaciones basadas en servicios reaccionen autónomamente ante los cambios en la implementación de los servicios utilizados, detecten posibles discrepancias en la integración y ejecuten dinámicamente estrategias de adaptación; en [108] la propuesta radica en el desarrollo de sistemas robóticos auto-adaptativos aplicando un enfoque basado en la arquitectura y en políticas que encapsulan el comportamiento adaptativo del sistema.

NEXUS [203] permite diseñar un sistema de información basado en nodos sociales y técnicos organizados, con la capacidad de soportar la dinámica de situaciones complejas y de las estructuras formales e informales dentro de organizaciones por medio de la sinergia holónica emergente.

Estas son algunas de las numerosas propuestas referidas a adaptación y auto-adaptación de sistemas software de los últimos años.

Por el lado de la industria, además de *Autonomic Computing* [126] de IBM que ya lleva más de diez años de investigación y desarrollo, existen otras iniciativas para desarrollar sistemas auto-administrados o autónomos, a fin de que la sobrecarga de gestión de dichos sistemas no recaiga en su totalidad en los administradores humanos. Resolver el problema

de la gran complejidad es central en todas ellas. Algunas de estas iniciativas son *Plant-Care* [144] de Intel Research; *Dynamic Systems Initiative* [52] de Microsoft; *Harmonius Computing Project (HARC)* [193] de Hitachi América Ltd. y *Adaptive Enterprise* [51] de Hewlett-Packard.

Finalmente, en las publicaciones [157, 77, 206, 178, 222] pueden encontrarse interesantes resúmenes y comparativas de desarrollos para auto-organización, adaptación y auto-adaptación de sistemas. También análisis y ampliaciones en los tópicos de investigación y los mecanismos relacionados en [40, 118, 141, 46].

2.4. Patrones de Software

El uso de *patrones de software* en el campo de la ingeniería de sistemas es una práctica que viene realizándose desde hace mucho tiempo por parte de arquitectos y desarrolladores de software. La importancia de los patrones en la construcción de sistemas cada vez más complejos favoreció la generación de propuestas en esta área. Ejemplos de estas son las de Gamma et al. [98] y Buschmann et al. [38]. Ambas proponen su propio catálogo de patrones de software, el primero más orientado al diseño de sistemas y el segundo hacia la arquitectura de los sistemas software.

Algunos de los actuales lenguajes de programación resuelven muchos de los problemas que presentan [98] y [38] por medio de sus bibliotecas, y como además ambas propuestas han sido presentadas hace ya más de quince años, parte de la comunidad desarrolladora de sistemas los podría considerar enfoques superados. Sin embargo sus soluciones continúan siendo un punto de partida más que apropiado para la resolución de problemas con los actuales sistemas software.

Originalmente la noción de *patrón* fue aportada para el campo de la arquitectura de edificios por el arquitecto Christopher Alexander [8, 7]. Una manera simple de describir la función de un patrón es decir que brinda una solución comprobada para un problema similar, se halla convenientemente documentado y de forma consistente, y usualmente integra

una colección mayor.

En palabras de Alexander *"cada patrón describe un problema que ocurre una y otra vez en nuestro entorno y luego describe el núcleo de la solución a ese problema de tal manera que se puede utilizar esta solución un millón de veces sin tener que hacerlo de la misma forma dos veces"* [8].
Más adelante en esta tesis se ampliará la visión de Alexander, el que también propone un *lenguaje de patrones* que de manera similar a una *red* (de patrones) que puede recorrerse estos pueden organizarse para obtener la solución. O en otras palabras, *un conjunto de patrones que trabajan juntos para definir una actividad compleja o una estrategia de diseño*.

En [241] se presenta un interesante análisis de las complejas relaciones entre los patrones de diseño, organización y categorización de los mismos como base para el desarrollo de un lenguaje de patrones (para sistemas orientados a objetos).

A su vez, en [98, 38], el concepto de *patrón de diseño* puede ser resumido como *la descripción de una plantilla* (o, en inglés, *template*) *de una solución a un problema en un contexto adecuado*. Estos *templates* son de gran utilidad porque representan, como ya se ha dicho, soluciones probadas en problemas recurrentes de diseño. Al ser productos de la experticia de los desarrolladores hacen que las técnicas sean más accesibles para los desarrolladores de nuevos sistemas y permiten organizar el diseño en un formato estándar y de fácil referencia. Asimismo pueden ser usados para asegurar consistencia en la construcción de los sistemas, y con ello también favorecer su calidad, además de ayudar a elegir alternativas que hagan que esos sistemas sean reutilizables y no "inventar" una nueva solución completamente distinta de las existentes[176].

En la propuesta de esta tesis se utilizan patrones de diseño para la resolución de los problemas que se generan con la adaptación de sistemas. No son los clásicos patrones orientados a objetos ya que son utilizados en un sentido más amplio: desde el *punto de vista arquitectónico*, por lo que podemos partir del concepto general para luego adecuarlos al tema de este trabajo, los *patrones arquitectónicos de adaptación*. Se proponen patrones eficientes para la adaptación de sistemas basados en agentes, centrados en organizaciones y

orientados a servicios. En línea con [177] se tienen en cuenta las buenas prácticas existentes, documentando las estructuras, supuestos, dinámicas y consecuencias de una decisión de diseño.

Numerosos trabajos, entre los que se pueden citar [165, 190, 226, 102, 90, 91], demuestran que la utilización de patrones de diseño en el desarrollo de sistemas adaptativos, y en numerosos casos también con SMA, se ha convertido en un área de investigación en crecimiento. A continuación se hace una descripción sucinta de trabajos propuestos que se relacionan con el tema de esta tesis.

Propuesta de Oluyomi et al.

En [165, 166], A. Oluyomi et al. abordan temas relacionados con la ingeniería de software orientada a agentes y sistemas multiagentes (SMA) buscando que con su aportación estas tecnologías sean adoptadas más ampliamente por la comunidad de la ingeniería del software. Con este objetivo se realiza un análisis en detalle de las interacciones y se plantea la experiencia compartida entre desarrolladores por medio de patrones de software.

Respecto del análisis de las interacciones, al ser una de las características comunes de los SMA, estas proveen una base para analizar y comprender el comportamiento del sistema, de manera independiente de lo complejo que sean los agentes que lo forman. Esto también permite razonar sobre conceptos y aplicabilidad de esta tecnología. Para conceptualizar, modelar e implementar las interacciones en los SMA los *protocolos de interacción de agentes* son muy importantes ya que dirigen el comportamiento global del sistema y también influyen en los procesos de decisión de los agentes. Es así que se presenta un enfoque específico para el desarrollo de *protocolos de interacción de agentes*, una estructura de los protocolos (*protocol structure*) y un conjunto de propiedades (*protocol property suite*). Estos dos últimos son los productos finales del enfoque que tiene dos fases involucradas: el *análisis dirigido por el dominio* y el *diseño/verificación*, fases que facilitan la reusabilidad productiva de los protocolos de interacción de agentes.

Con la utilización de patrones de software y la tecnología de agentes se logra presentar los conceptos y principios de esta tecnología a la amplia comunidad de profesionales del

software. Los patrones orientados a agentes estructuran una conexión entre los problemas de desarrollo y los conceptos aplicados en su resolución.

La contribución de Oluyomi et al. en esta área propone un "esquema de clasificación de dos vías" (en inglés *Two Way classification scheme*) que provee una plataforma para clasificar, analizar y describir los patrones. Este esquema comprende dos dimensiones, una *vertical*, basada en los niveles de abstracción en la ingeniería de software orientada a agentes, y una *horizontal*, definida por las tareas de desarrollo de software en cada uno de los niveles de abstracción. De esta manera cada categoría en el esquema de clasificación está definida por el nivel de abstracción y una tarea de desarrollo a ese nivel. Con los atributos de las categorías se define el criterio para el análisis de los patrones y la base para el diseño de los *templates* de dichos patrones. Este análisis basado en atributos permite evaluar la versatilidad de la solución de un patrón, su calidad y así clasificarlos según el esquema. A su vez, el *template* es el medio para transmitir las experiencias que se deben compartir en el desarrollo de software. Se analizaron, clasificaron y documentaron veintiocho patrones, lo que permitió relacionar calidad y complejidad. Para ello el *template* estructura la descripción de los patrones, derivándose de él otros *templates* adaptables a las diferentes categorías de los patrones, los que también resultan consistentes en sus derivaciones. Los *templates* se derivaron de las ocho categorías del esquema de clasificación. Su diseño apunta a caracterizar los conceptos de la tecnología de agentes en un formato conocido para la comunidad de los profesionales del software y así promover la comunicación y comprensión de la tecnología de agentes.

Sin embargo, en este trabajo no se han definido nuevos patrones ni se han utilizado la abstracción de organizaciones o el concepto de acuerdos entre agentes para la resolución de problemas, como en la propuesta en esta tesis.

Propuesta de Ramírez y Cheng

En el trabajo de investigación de A. Ramírez y B. Cheng [189, 190] se han estudiado numerosos enfoques y proyectos relacionados con la adaptación de sistemas. De esta manera, generalizando desde varias soluciones de diseño existentes, han obtenido y desarrollado diversos patrones orientados a la adaptación, los que están a nivel de diseño de software y

con la finalidad de facilitar la construcción de sistemas de software adaptativos.

El aporte de este trabajo ha consistido en:

- un *template* de patrón de diseño de adaptación, basado en el de Gamma et al. [98], que asiste a los desarrolladores en la comprensión y diseño de sistemas adaptativos;

- el conjunto de los doce patrones de diseño para promover la reutilización de las decisiones de diseño y también para comprender las alternativas existentes para la adaptación de los sistemas en tiempo de ejecución; y

- extensiones del proceso de desarrollo basado en modelos introducido por Zhang y Cheng [240], que incorpora el uso de patrones de diseño de adaptación y técnicas de modelado basado en estados, separando, en el proceso, las especificaciones de los comportamientos adaptativo y no adaptativo de los sistemas.

Consideran tres elementos claves que se encuentran presente en la mayoría de los sistemas adaptativos, incluyéndose también a los sistemas autonómicos. Ellos son: el *monitoreo (monitoring)*, que permite que una aplicación pueda ser consciente de su entorno y detectar condiciones que favorezcan su reconfiguración; la *toma de decisiones (decision-making)*, que determina qué conjunto de condiciones controladas disparará una respuesta específica de reconfiguración; y la *reconfiguración (reconfiguration)*, que permite que una aplicación se cambie a sí misma para conseguir su objetivo. Asimismo, estas tres claves son utilizadas para clasificar sus patrones basándose en su objetivo mayor. Ampliando la clasificación inicial (*M - monitoring, DM - decision-making*, y *R - reconfiguration*), M y DM pueden también ser *creacionales (C - creational)*, o *estructurales (S - structural)*. De igual manera los patrones R pueden clasificarse a su vez como *de comportamiento (B - behavioral)*, o *estructurales (S - structural)* [191, 190].

Los patrones facilitan, según los autores, el desarrollo por separado de la lógica funcional y de la lógica adaptativa. En base a la clasificación propuesta, se tienen los patrones M: *Sensor Factory, Reflective Monitoring, y Content-based Routing*; los patrones DM son *Adaptation Detector, Case-based Reasoning, Divide and Conquer, Architecture-based, y TradeOff-based*; y por último los patrones R: *Component Insertion, Component Removal, Server Reconfiguration, y Decentralized Reconfiguration*. Para evaluar su maduración se validó cada patrón contra instancias en sistemas adaptativos que no hayan sido usados en el

proceso de recolección y también probaron cómo varios patrones pueden ser combinados para construir sistemas autónomos y auto-adaptativos.

Este enfoque busca no estar acoplado con tecnologías o dominios específicos que limiten su aplicabilidad ya que los patrones de diseño se utilizan en altos niveles de abstracción. Sin embargo la investigación llevada a cabo por estos autores no tienen en cuenta a los agentes de software y por ende tampoco a los sistemas multiagente, elementos fundamentales en nuestro enfoque. Tampoco se utilizan los conceptos de organizaciones y acuerdos, que son utilizados para obtener el sistema de patrones arquitectónicos propuesto por esta tesis.

Propuesta de Weyns

En [231, 226] se presenta el desarrollo de un conjunto integrado de patrones arquitectónicos para *sistemas multi-agente situados*. Los agentes estan situados en un entorno que mantiene una virtualización de las partes relevantes del mundo y que les sirve como medio de coordinación, mediando en las interacciones entre los agentes y también para que estos accedan a los recursos.

El *SMA situado* se enfoca en la modularización del comportamiento del agente, toma de decisiones eficiente y coordinación indirecta. Se caracteriza porque su control es descentralizado, lo que es esencial para tratar con la propia distribución de los recursos; asimismo su funcionalidad resulta de la cooperación de los agentes a través del entorno intermediario que emplean para compartir información y coordinar su comportamiento. El dinamismo y cambio se refiere a las modificaciones a que el sistema puede estar sujeto durante su operación [226].

El conjunto integrado de patrones propuesto recibe el nombre de *Estrategia Arquitectónica* y origina, según los autores, "un lenguaje de patrones destinado a proveer soporte en el diseño de una familia de sistemas auto-adaptativos descentralizados" [231].

Este lenguaje está formado por cinco patrones: *Situated Agent, Virtual Environment, Selective Perception, Protocol-based Communication* y *Roles & Situated Commitments*. Los dos primeros son los patrones básicos y más importantes de dicho lenguaje, con respecto a *Selective Perception*, permite que un agente situado pueda sensar el entorno virtual y actualizar su conocimiento acerca del mundo; *Protocol-based Communication* habilita

el intercambio de mensajes entre agentes de acuerdo a una secuencia bien definida; y por último, *Roles & Situated Commitments* consiste en actitudes sociales de los agentes situados; un rol representa una parte coherente de la funcionalidad de un agente situado en un contexto de colaboración y un compromiso situado define un compromiso entre agentes en una colaboración, afecta el comportamiento de los agentes involucrados en el compromiso en favor de los roles que los agentes juegan en la colaboración [226].

Este trabajo también utiliza un *template* para la documentación y un lenguaje de descripción arquitectónica. Se documenta cada patrón con una representación principal, los elementos constitutivos con la explicación de sus responsabilidades y un razonamiento para aclarar las decisiones de diseño subyacentes y la calidad de los atributos asociados con los patrones. Asimismo se incluyen especificadiones formales y la manera en que se utilizan conjuntamente.

Aunque se propone un enfoque basado en patrones no se utilizan los conceptos de organización y acuerdos entre entidades para lograr metas y objetivos, conceptos que sí son utilizados en nuestra propuesta.

Propuesta de Gardelli, Viroli y Omicini

En la propuesta de Gardelli et al. [102, 101, 100] se hace hincapié en que al tiempo de realizar la investigación, se notaba la ausencia de un catálogo sistemático de patrones de software a ser utilizado en el desarrollo de sistemas auto-organizados. A estos efectos los autores proponen un enfoque metodológico para enfrentar las primeras fases del diseño de SMA auto-organizados que se basa en el metamodelo de Agentes y Artefactos (A&A) [196]: los mecanismos de auto-organización están contenidos en un tipo de agentes llamados *agentes de entorno* y que se encuentran en el ambiente. De esta manera para modelar un SMA se utilizan las abstracciones *agentes*, definidos como entidades autónomas (ver 2.1.1), y *artefactos*, vistos como elementos pasivos o reactivos que proveen servicios y funcionalidades a los agentes.

Los patrones que se presentan han sido obtenidos a partir de trabajos existentes relativos a auto-organización de sistemas, incluso de modelos de actividades naturales como el comportamiento social de insectos. Para la descripción de los patrones han utilizado el

esquema (o *template*) para SMA propuesto en [145].

Sus patrones son *Replication, Collective Sort, Evaporation, Aggregation* y *Diffusion*:

- Con *Replication* se puede lograr incrementar la seguridad y robustez del sistema, así como también es posible reducir el tiempo de acceso requerido a ciertos recursos.

- *Collective Sort* tiene como objetivo agrupar información similar en el mismo nodo, también permite separar tipos de datos aunque en ocasiones no es posible saber a priori dónde surgirá un *cluster* específico.

- *Evaporation* puede ser considerado como un mecanismo para reducir la cantidad de información en base a un criterio de relevancia en el tiempo.

- El patrón *Aggregation* es visto como un mecanismo de refuerzo impulsado por los agentes usuarios del sistema y usado en combinación con el anterior patrón puede obtenerse un ciclo positivo/negativo de retroalimentación.

- Con *Diffusion* se distribuye la información de manera equitativa entre todos los nodos que forman el sistema, siempre en base a metas y objetivos.

Se utiliza un enfoque metodológico basado en el modelado, simulación y *tuning* de varios modelos de patrones con la meta de encontrar el que puede proveer los atributos de calidad requeridos por la aplicación. De esta manera se pueden evaluar prototipos candidatos de *agentes de entorno* y su interacción con los artefactos antes de aplicarlos al diseño final. En este metamodelo no hay una representación explícita de topologías, aunque podrían ser modeladas implícitamente, por ejemplo, permitiendo que los *agentes de entorno* tengan acceso solo a un subconjunto de artefactos, o también aprovechando el concepto de espacio de trabajo (*workspace*) de A&A, como se utiliza en [197].

Es importante notar que si bien los patrones presentan mecanismos básicos son considerados muy generales para aprovechar su utilidad. Sin embargo, la construcción de patrones más complejos por encima de otros más simples puede permitir una mayor comprensión de la dinámica de los sistemas y favorecer su control y coordinación. Nuevamente los conceptos de organización y acuerdos entre agentes no se encuentran presentes en esta propuesta.

Propuesta de Fernández-Márquez et al.

A partir del estudio de sistemas naturales bio-inspirados y con características de auto-organización, Fernández-Márquez et al. [90, 91] proponen un catálogo de patrones de diseño de SMA con el fin de facilitar la ingeniería de sistemas artificiales. Estos involucran individuos autónomos y proactivos, alta interacción y descentralización en la coordinación. Por otro lado, esto permite que también puedan presentar robustez, escalabilidad y capacidad de adaptación a cambios en el entorno.

Básicamente sus patrones de diseño son modulares y reusables. Consisten en mecanismos de auto-organización expresados de manera uniforme y organizados en diferentes niveles. El modelo computacional utilizado para describir su dinámica y las relaciones entre las entidades está claramente inspirado por la biología, aunque está especializado para los sistemas artificiales que usarán dichos patrones. Se parte considerando que el modelo biológico está constituido por dos niveles: los *organismos*, que colaboran en el proceso biológico; y el *entorno*, que es el espacio físico donde ellos están y que ofrece recursos y eventos para el desarrollo de los procesos. Para simular este modelo de manera artificial se agrega una capa de infraestructura entre las entidades –agentes– y el entorno para permitir las interacciones.

El catálogo obtenido está formado por patrones clasificados en tres niveles: *Básicos, Compuestos* y de *Alto Nivel*. Para su descripción utilizan un *template* ad hoc aunque similar a otros ya vistos [98, 145].

- Los *Básicos* describen mecanismos elementales frecuentemente utilizados, y que con su composición se pueden obtener los otros dos tipos de patrones. Ellos son: *Spreading, Aggregation, Evaporation,* y *Repulsion*. Con el primero los agentes pueden propagar información sobre la red para reducir la falta de conocimiento de las otras entidades; con el segundo se aplican operadores de combinación y sintetización cuando la información es muy abundante. El tercero es un mecanismo que periódicamente reduce la relevancia de la información para que prevalezca la más reciente; con el último patrón se produce la "repulsión" de agentes y se los envía a zonas con menor concentración de entidades.

- Los patrones *Compuestos* son *Gradients, Digital Pheromone,* y *Gossip*. Con el primer patrón se utilizan filtros durante el proceso de agregación de la información y de la ubica-

ción de sus fuentes, además con esto y con "evaporación" se pueden adaptar las topologías de las redes. El segundo provee una manera de coordinar el comportamiento de los agentes utilizando comunicación indirecta. Con el último patrón los agentes pueden llegar a un entendimiento cuando los sistemas son de gran escala.

- Los de *Alto Nivel* son *Ant Foraging, Chemotaxis, Morphogenesis, Quorum Sensing,* y *Flocking*. El primer patrón provee reglas para explorar descentralizadamente el entorno y aprovechar sus recursos. Con el segundo se consigue coordinación de movimientos descentralizados con el objetivo de detectar las fuentes o límites de eventos. El tercer patrón facilita a los agentes las tomas de decisiones respecto de sus roles o planes de actividades basados en su posición espacial. Con el cuarto es posible que los agentes tengan la capacidad para tomar decisiones colectivas. Con el último patrón los grupos de agentes pueden moverse en el entorno mientras mantienen su formación e interconexiones.

Si bien con el patrón Flocking es posible enfrentar el problema de la coordinación de movimientos de un "enjambre" de agentes, el resultado solo facilita el traslado de los mismos. A diferencia de esto, en nuestro trabajo se utiliza una abstracción mayor y con alcance más importante: las organizaciones de agentes, con características similares a las organizaciones reales y con diferentes topologías estructurales.

2.5. Conclusiones

El estado del arte presentado en este capítulo nos permite ver que, como ya hemos dicho en la Sección 2.1, la forma más avanzada para trabajar con agentes de software y SMA es la utilización del concepto de *organizaciones*.

De manera similar hemos notado que existe una serie de esfuerzos importantes en la adaptación de sistemas y también en adaptación de SMA. Esta área de investigación brinda actualmente amplias posibilidades de estudio.

Si ponemos el énfasis en los SMA para entornos abiertos, heterogéneos y muy dinámicos hemos visto que la utilización de organizaciones en su desarrollo se ha incrementado. Esto indica que este enfoque representa una manera adecuada para enfrentar problemas

y dominios cada vez más complejos. Sin embargo las organizaciones propuestas en estos desarrollos presentan ciertas limitaciones ya que en general son organizaciones estáticas y no adaptativas. Es más factible encontrar desarrollos de SMA auto-organizados que de SMA adaptativos y con organizaciones. Por otro lado, con respecto a los sistemas adaptativos no es usual que estos tengan en cuenta una estructura de organizaciones, en el sentido que se utiliza en el enfoque con agentes de software.

Como no es usual encontrar desarrollos de SMA auto-organizados que puedan "generar" organizaciones, esto va a representar una de las metas de este trabajo. Se utilizarán agentes que integran SMA, se estructuran en organizaciones y además en una propuesta en la cual ya no se hace énfasis solo en la agencia sino también en la potencia de los *acuerdos* (dinámicos) entre agentes. Para lograr versatilidad en los agentes estructurados en organizaciones y que a su vez sean capaces de lograr organizaciones adaptativas los esquemas de comportamiento deben seguir ciertos patrones. Estos van a facilitar el comportamiento emergente de nuestras organizaciones de agentes.

Resumiendo, hemos examinado los sistemas de agentes y notamos que los más evolucionados utilizan conceptos de organizaciones en su definición y desarrollo. También hemos visto la importancia de la adaptación, inclusive en trabajos importantes con sistemas de agentes generando un interesante campo de investigación. La adaptación en sistemas continúa evolucionando y es posible definir esquemas de adaptación complejos y con patrones.

Creemos que los sistemas que utilizan agentes y que implementen adaptaciones teniendo en cuenta estructuras de organizaciones no son los más frecuentes, por lo que consideramos que la forma más evolucionada que puede alcanzar un SMA auto-organizado en un esquema de patrones es precisamente "definiendo" organizaciones.

Nuestro objetivo es proporcionar los patrones de adaptación que sean capaces de definir organizaciones en sistemas que utilizan agentes de software, y a eso dedicaremos el próximo capítulo.

Capítulo 3

Hacia una Arquitectura Adaptativa

3.1. Introducción

Si nos hacemos la pregunta *porqué se necesitan sistemas adaptativos, o incluso auto - adaptativos*, las razones para responder pueden ser muy variadas. Entre ellas podemos encontrar, por ejemplo, el aumento del costo para hacer frente a la complejidad de los sistemas actuales; o realizar cambios de prioridades y políticas que gobiernan el sistema; gestionar condiciones inesperadas o los cambios en el entorno del funcionamiento. Si a estas razones le agregamos que en la actualidad varias de ellas deben ser realizadas en tiempo de ejecución (*runtime*), visto esto como una evolución dinámica, entonces para conseguirlo es necesario que haya un control del sistema y de su entorno para detectar los cambios, tomar las mejores decisiones y actuar en base a ellas [206].

En este Capítulo se presentan las principales cuestiones conceptuales y tecnológicas consideradas relevantes para establecer una arquitectura adaptativa, que en nuestro enfoque estará orientada a organizaciones de agentes software. Como se verá más adelante, la adaptación a los cambios en el entorno pueden alterar no solo la efectividad de las organizaciones sino también ser causantes de la modificación de su estructura y/o componentes.

Los elementos fundamentales para ello se examinarán en orden y permitirán definir el contexto preliminar en el que se establece este trabajo.

3.2. Arquitectura Basada en Agentes, Orientada a Servicios y Centrada en Organizaciones

En el Capítulo anterior hemos visto que numerosos problemas han sido resueltos utilizando agentes y SMA. Esto nos permite considerar a ambos como conceptos cada vez más populares en IA para la resolución de problemas complejos.

Se han propuesto diversas estrategias de desarrollo para que los SMA sean flexibles y puedan coordinarse a efectos de adaptarse a los cambios del entorno. Por otro lado, también se acepta que los SMA no han tenido el éxito esperado en la industria del software, como bien se discute en [65, 230]. Probablemente esto se deba a una cultura de desarrollo de sistemas diferente, o también en una reticencia a cambiar de paradigma en la resolución de problemas por parte de muchos desarrolladores. Incluso debido a que cuando las soluciones provienen de la interacción entre agentes software en muchas ocasiones no queda muy claro cómo y quién o quienes han resuelto el problema, en vista de determinar responsabilidades al respecto.

La propuesta de esta tesis es intentar salvar esta brecha utilizando conceptos orientados a servicios, los que sí cuentan en la actualidad con una mayor popularidad [34, 22, 80, 148]. Si además se demuestra que los SMA pueden adquirir la capacidad de auto-adaptación se podrá lograr, en definitiva, que el sistema sea capaz de adaptarse por sí mismo a los cambios del entorno.

La arquitectura propuesta para dar soporte al modelo se define como un SMA abierto, que además está orientada a servicios y centrada en organizaciones. En las secciones siguientes se describirá el diseño básico de esta arquitectura, y también la evolución de la misma para que sea capaz de obtener estructuras dinámicas y emergentes. El objetivo final es que este diseño describa una arquitectura verdaderamente auto-adaptativa.

Debido a que se propone un entorno flexible y dinámico, se requiere el uso de capacidades semánticas y de alta tecnología. Por lo tanto, se considera el uso de agentes en un contexto más amplio, con el agregado de una capa superior de servicios para proveer una función de interoperabilidad. Es fácil concebir un servicio como una manera de presentar

las capacidades operacionales de un agente, y mejor aún, de una colección de agentes como una organización, la que será proveedora del servicio.

Utilizar agentes permite aprovechar varias de sus características frente al enfoque puramente en servicios, como por ejemplo que los agentes pueden hacer un tratamiento explícito de la semántica, son capaces de lograr una coordinación estructurada, si poseen capacidad de aprendizaje esto les permite ser conscientes de la existencia y capacidades de otros agentes con los que interactuar, también es factible usar una metodología para el desarrollo de servicios y estructurarlos en organizaciones, entre otras.

Como se ha dicho anteriormente, entre las metas principales de este trabajo de tesis se persigue primero la definición de una plataforma general para identificar la arquitectura subyacente *basada en agentes, orientada a servicios y centrada en organizaciones*; en segundo lugar, la introducción de *estructuras adicionales* para hacerla adaptativa; y por último, la identificación de las *estructuras genéricas de adaptación para las organizaciones* basadas en acuerdos.

La noción central es el servicio, el componente básico de la arquitectura es el agente, y la estructura que unifica los conceptos es la organización, concebida como una composición jerárquica y recursiva de agentes. Implícita en la definición de SMA se encuentra la necesidad de registrar a los agentes en el sistema para separar aquellos que pertenecen a la arquitectura de los que no. Un enfoque similar será utilizado con los servicios: varias operaciones de agentes serán promovidas por organizaciones y exportadas específicamente. Para permitir su acceso externo, primero deberán ser explícitamente registrados y agrupados como parte de un servicio. Estos servicios podrán posteriormente ser localizados (*descubiertos*) por otras entidades dentro del registro distribuido del sistema, lográndose de esta manera una de las características dinámicas más importantes de una arquitectura orientada a servicios.

La computación orientada a servicios se refiere a aplicaciones construidas con servicios constituidos como componentes modulares de software autocontenidos y muy poco acoplados. Estos servicios se registran y se localizan para hacer uso de ellos, también se

han desarrollado APIs [1] y herramientas que hacen posible manejar cierto estándar y producir aplicaciones muy eficientes.

Con los sistemas multiagente se tienen niveles altos de abstracción, comportamiento complejo e inteligente, existe interoperabilidad semántica, publicación y descubrimiento de servicios descritos semánticamente, pueden utilizarse agentes intermediarios y también modelos organizacionales.

Con la integración de estas dos soluciones es posible modelar entidades computacionales autónomas y heterogéneas en entornos abiertos, dinámicos y distribuidos. El objetivo perseguido es la conjunción de estas tecnologías con el concepto de *acuerdo* como elemento aglutinante, como se verá más adelante. En esta línea los agentes pueden proveer y solicitar servicios de un modo transparente a otros agentes o entidades (organizaciones), e incluso esta interacción puede darse entre entidades externas y los agentes utilizando los servicios ofrecidos.

3.2.1. El Rol de las Organizaciones en la Coordinación

Los agentes software fueron originalmente concebidos como entidades individuales que poseen ciertas características como la autonomía, proactividad, flexibilidad, habilidad social, etc., como se ha discutido en el Capítulo anterior. Con la necesidad de dar solución a problemas más complejos y que obligan a distribuir inteligencia entre sus elementos se desarrollan los SMA, con los que se puede lograr un cambio de enfoque y utilizar nuevas metodologías. De agentes individualistas es posible pasar a considerar, por ejemplo, *agentes cooperantes* dentro de un grupo con determinados objetivos. Uno de sus mayores desafíos es cómo inducir la cooperación entre los agentes, los que tienen cierto nivel de autonomía, metas individuales que pueden ser conflictivas, y que además tienen una visión local del entorno en que se encuentran. Es ahora el sistema quien tiene la responsabilidad de conciliar los distintos puntos de vista y resolver el problema para el que fue creado. En suma, un SMA estaría en condiciones de resolver cuestiones muy complejas, aunque subyace el problema de *coordinar* las entidades que lo forman.

[1] API (Aplication Programming Interface, o en castellano, Interfaz de Programación de Aplicaciones) es un conjunto de elementos y herramientas que permite construir aplicaciones de software.

La RAE, Real Academia Española [187], en su segunda acepción define la palabra *coordinar* como *"concertar medios, esfuerzos, etc., para una acción común"*. Si nos enfocamos ya en el campo de los SMA posiblemente una de las definiciones más aceptadas de *coordinación* provenga de la Teoría de la Organización que la define como la *"gestión de las dependencias" entre actividades organizativas* [152]. De estas definiciones se pueden deducir ciertos elementos que intervienen en la coordinación, como por ejemplo las metas, actividades, actores e interdependencias. En un entorno como el empleado en este trabajo, las entidades a coordinar son los *agentes*, mientras que los elementos de coordinación usualmente son las metas, acciones o planes [156].

Si se hace un análisis desde un punto de vista centrado en el agente [155] la coordinación es entendida como una *adaptación al entorno*. Si nuevos agentes entran en el contexto de otro, este último tiene que coordinarse con aquellos reconsiderando sus metas, planes y acciones, y deberá tener en cuenta las nuevas y potenciales interdependencias. El resultado de la coordinación para el agente en este caso será cada vez mejor si demuestra un alto grado de cooperación para conseguir el objetivo del sistema.

Si por el contrario, la visión se centra en el SMA, los resultados de la coordinación pueden verse como una cuestión global (plan, decisión, acuerdo, etc.). Pudiendo ser un plan "compartido" [202] si los agentes consiguen llegar a un acuerdo explícito durante la coordinación; o ser una suma de planes individuales, en ocasiones llamado "multi-plan" [171]. En este nivel, los resultados de la coordinación pueden ser evaluados como una conjunción de las metas de los agentes o teniendo en cuenta la funcionalidad del SMA [156, 172].

De hecho, una estructura de agentes auto-organizada podría considerarse como óptima si contara con la capacidad para resolver las cuestiones de coordinación. En este contexto, la hipótesis es que los agentes están adecuadamente organizados, es decir, que "se coordinan solos", como individuos en una sociedad. En un entorno cerrado, el comportamiento de los agentes puede controlarse en tiempo de diseño y sólo habría que resolver la eficiencia de los mismos. Sin embargo, en entornos abiertos y de gran escala esta auto-organización podría ser, al menos en parte, *emergente*. La idea es que los agentes deberían organizarse

de tal manera que "espontáneamente" reaccionen como se desea, en la dirección correcta. Pero para que esto sea satisfecho, el enfoque orientado a agentes debería ser complementado con una *estructura de coordinación auto-organizada*, esto es, a nivel de *arquitectura*.

Los trabajos pioneros donde se relacionan agentes y coordinación fueron abordados en el campo de la IA hace ya más de tres décadas. En estos trabajos la cooperación, por ejemplo, es vista como un concepto requerido para la resolución inteligente de problemas complejos [156]. Algunas de las cuestiones más importantes fueron cómo encontrar los modelos cooperativos para cada clase de problema; cómo representar estos modelos de manera que se puedan ejecutar; y cuáles eran los métodos necesarios para que los sistemas cooperativos fueran realmente efectivos. A estas se agregaron el cómo dividir los problemas en tareas más pequeñas para distribuirlas entre los agentes; o cómo la solución de un problema puede ser efectivamente combinada a partir de las soluciones de los subproblemas [236].

Para dar respuesta a estas cuestiones, podemos considerar [156] dos puntos de vista:

- Desde el punto de vista del *agente colaborador*: la cooperación es introducida como un mecanismo más para aumentar su eficiencia en la resolución de problemas, el sistema decide por sí mismo cuándo y cómo colaborar, observa los resultados de este proceso y aprende para el futuro;

- Desde el punto de vista del *problema a resolver*: encontrar la mejor manera de estructurar y descomponer un problema complejo para su resolución, dado un conjunto de entidades colaborativas.

Entre las soluciones que se desarrollaron encontramos la arquitectura de *pizarra* [78], que provee cooperación entre fuentes de conocimiento usando un mecanismo sencillo de comunicación. Cada subsistema toma datos de la pizarra, produce sus resultados y luego los escribe en ella. Otra es la *red de contratos* [215], que propone la negociación como mecanismo para coordinar y asignar tareas a las diferentes entidades que participan en la resolución del problema. Cada entidad puede ser contratante o contratada y las condiciones de cooperación se establecen al firmar el contrato. Otra solución fue la propuesta de una arquitectura *reactiva* [35], que como se explicó en 2.1.1 se trata de obtener un comportamiento inteligente desde modelos simples, sin mecanismos de representación de conocimiento, razonamiento o aprendizaje. Como un avance en el desarrollo de estas soluciones (ver 2.1.3),

tenemos las arquitecturas de agentes con *capacidad organizacional*. En estas, aparte del conocimiento del dominio, los agentes necesitan conocer sus propias características sociales. Este conocimiento les permitirá saber cuál es su rol en la organización, las reglas y normas, las capacidades de los otros agentes, etc. [124].

Los SMA pueden ser diseñados con una estructura organizacional fija, los que estarán compuestos por agentes estáticos, como se ha visto en 2.1.3. Si por el contrario se utilizan agentes flexibles o evolutivos se pueden obtener modelos organizacionales más complejos, los que permitirían reducir la brecha entre el comportamiento de organizaciones humanas y los sistemas que simulan e implementan ese comportamiento. Para lograr esto es necesario que se puedan modificar propiedades importantes, como los roles, los niveles de autonomía, los mecanismos de control, los protocolos de interacción, entre otras.

Las metodologías de SMA puramente orientadas al agente se concentran usualmente en la visión del agente; se asume que el comportamiento final del sistema emerge de las interrelaciones entre los agentes diseñados. El comportamiento global no se analiza en detalle. Si agentes externos ingresan a la organización, el comportamiento global del sistema puede verse afectado a menos que se establezcan controles, normas, sanciones, etc. Por otro lado, las metodologías orientadas a las organizaciones tienen en cuenta la organización del sistema desde los inicios y el análisis del sistema sí que se hace desde una perspectiva global. Incluso los objetivos describen los propósitos organizacionales a un nivel alto. Esto permite la determinación de tareas, tipos de agentes, asignación de recursos entre miembros, etc. El sistema se estructura por los roles, interacción entre agentes y los lenguajes de comunicación que utilizan. Las normas sociales describen el comportamiento deseado de los miembros. Estas normas luego derivan en el control, prohibiciones, sanciones, etc. para lograr el comportamiento global esperado. Se tienen en cuenta mecanismos para permitir que agentes externos entren en la organización y controlar su comportamiento. Todas estas características son particularmente útiles para el diseño de SMA abiertos [12, 15].

Estamos considerando el concepto de *coordinación* como una cuestión previa a la adaptabilidad debido a que cuando utilizamos un SMA como solución, el problema de la coordinación siempre está presente, y por ende la adaptabilidad también se verá comprometida.

Un modelo de coordinación adecuado debería abarcar los temas de creación y destrucción de los agentes, su comunicación y su distribución espacial, así como la sincronización y la distribución de sus acciones en el tiempo [50]. En un sistema de coordinación pueden considerarse los siguientes componentes: las *entidades* (llamados también *coordinables*), cuyas interacciones se rigen por el modelo; los *medios*, las abstracciones que rigen las interacciones; y las *leyes*, que definen el comportamiento de los medios de coordinación en respuesta a las interacciones [50].

Asimismo, y de acuerdo con [175], podemos considerar dos tipos de modelos de coordinación: los modelos *control-driven*, o dirigidos por el control, y los modelos *data-driven*, o dirigidos por los datos. Los primeros se centran en el acto de la comunicación, mientras que los segundos en la información intercambiada en dicha comunicación. De hecho, en lugar de considerarlos como modelos opuestos y diferentes, se los podrían ver como dos maneras diferentes de observar la coordinación.

Cuando se tiene el caso de un *meta-modelo de espacio de tuplas* [50, 43], posiblemente el más conocido de la coordinación *data-driven* (o generativa), las entidades basan sus interacciones (cooperación, competición, etc.) en las propias tuplas de datos. La coordinación tiene lugar mediante la producción, el consumo, modificación, etc. de las tuplas. En resumen, se crea un entorno de *comunicación generativa*. Y avanzando un poco más en la evolución de la coordinación, los sistemas *auto-organizados* [10, 50] tienen un creciente nivel de organización interna entre sus componentes en términos de interacciones, estructuras y roles.

Una definición de *auto-organización de la coordinación* [43] plantea una gestión de interacciones del sistema con propiedades de auto-organización, donde las interacciones son locales y los efectos de coordinación global deseados aparecen por *emergencia*. Constructivamente, la auto-organización de la coordinación se consigue mediante un medio de coordinación distribuido sobre el entorno topológico, estableciendo reglas de coordinación probabilísticas y dependientes del tiempo.

Teniendo en cuenta el enfoque de la *Teoría de la Actividad* [194], que conceptualiza la actividad humana como unidad fundamental de análisis para determinar las interacciones entre el ser humano y su entorno material, en [195] se usan los llamados *artefactos de*

coordinación. Estos artefactos son abstracciones utilizadas para instrumentar el entorno, que encapsulan y proveen servicios de coordinación a agentes que se encuentren en cierto entorno social.

Como ya se ha dicho, un enfoque significativo y cada vez más popular es considerar a los agentes dentro de organizaciones, esto puede verse como un enfoque de coordinación en un nivel más alto: en lugar de tener una coordinación normal de todo el sistema, las organizaciones permiten tener un *ámbito* de coordinación. Con esta segunda estructura se simplificaría la adaptación: se reducirá mayormente a la *reorganización* de agentes, esto es, si un agente es segregado de una organización y es reagrupado en otra con diferentes restricciones. Como es de notar, esto puede traer consecuencias no solo para el propio agente sino también para las organizaciones, tanto la de origen como la de destino, las que podrían a su vez necesitar adaptarse a la nueva configuración.

Al mismo tiempo esta situación implica otras cuestiones: por un lado, acerca del rol interno de las organizaciones en el SMA y su función en el sistema; y por otro, la necesidad de proveer coordinación –de orden superior– para las propias organizaciones, de manera que puedan conseguir su adaptación.

En este sentido, hay una relación intrínseca entre la coordinación auto-organizada mencionada anteriormente y el trabajo de esta tesis, que tiene en cuenta la definición de una arquitectura adaptativa en base a organizaciones emergentes y por lo que se obtendrá una coordinación *auto-organizada, con ámbito y basada en acuerdos*, como se verá más adelante.

3.2.2. El Rol de los Servicios

Las capacidades y funcionalidades de la arquitectura propuesta para dar soporte al modelo se brindan como *servicios*, por lo que el concepto de servicio constituye la piedra angular de esta arquitectura, la que se describirá más adelante. En nuestro contexto un servicio reúne un conjunto de operaciones, está descrito en su interfaz estándar y comprende una secuencia (semi-ordenada) de actividades. Asimismo está detallado semánticamente por un perfil y un modelo explícito de procesos, y estos últimos a su vez pueden ser divi-

didos en varios procesos menores. Podrían existir varias implementaciones para un mismo servicio y un idéntico perfil, y ser ofrecidos por diferentes, y posiblemente muchos, proveedores de servicios.

En pocas palabras, un servicio tiene una naturaleza relativa al comportamiento y presenta una interfaz funcional, está provisto por un proveedor y por un rol concreto dentro de este último. En línea con [13], según sea el proveedor, existen básicamente dos conjuntos distintos de servicios en la arquitectura que se propone como base:

- *Servicios base*, o simplemente *servicios*. Son a nivel de usuario, definidos para una aplicación concreta. El propósito de la arquitectura es servir de soporte para definir y usar servicios de esta clase.

- *Servicios de sistema*. No son estrictamente "servicios" ya que no son ofrecidos por un proveedor concreto a nivel de usuario y están usualmente a un nivel más bajo que un servicio base. El término se refiere a los servicios que no son provistos por ninguna otra entidad (organización, rol o agente) excepto por el propio sistema, es decir, por el *middleware*. En resumen, son los servicios de soporte de la plataforma.

Sin embargo, de acuerdo a la función y la extensión de sus capacidades pueden identificarse tres conjuntos separados de servicios [13], a saber:

- *Servicios estructurales*, que son los que permiten definir una cierta estructura organizacional/arquitectónica, creando y registrando organizaciones (unidades), sus elementos (roles) y propiedades (normas), y también sus relaciones. Hacen posible establecer y modificar las especificaciones del sistema, tanto estructural como normativo. Estos servicios también son estáticos por naturaleza, por ejemplo, una vez que las unidades han sido creadas y sus normas y propiedades fueron establecidas, pueden ser eliminadas pero no modificadas.

- *Servicios de información*, son los servicios que proveen información específica (nombre, cantidad, roles, etc.) de los componentes en una organización. Algunos son servicios publicados y registrados, mientras otros solo están concebidos para el uso interno de la propia infraestructura y permanecer invisibles. Son ofrecidos por el sistema para dar información acerca del propio sistema, o bien, implementados por una entidad para proveer información acerca de su propia composición y propiedades.

- *Servicios dinámicos*, son los que permiten a los miembros ingresar o abandonar dinámicamente una organización, así como adoptar roles existentes. Los agentes y los roles deben existir previamente, pero pueden ser asociados dinámicamente con cierta unidad organizacional. Estos servicios están habilitados para modificar servicios, unidades y roles, proveen una reconfiguración dinámica pero no intentan cubrir todo el dinamismo de la arquitectura, aunque sí hay un alto grado de dinamismo al permitir el agregado o eliminación de roles en una organización.

Pueden haber varias implementaciones para el mismo servicio, el que es descrito usando la misma interfaz y el mismo perfil semántico. Cada implementación está especificada semánticamente por un diferente modelo de proceso de servicio y, a su vez, esa implementación conforme al modelo de proceso puede ser provista por varios y distintos proveedores de servicios.

Un concepto importante a tener en cuenta es el de *proceso de servicio*. Este intenta dar una perspectiva semántica clara de la funcionalidad de un servicio. Este proceso interactúa por medio de una serie de entradas, salidas y efectos, y puede estar restringido por precondiciones. Sus salidas y efectos también pueden tener condiciones, las postcondiciones.

Según [156], en el modelo de proceso de servicio pueden identificarse tres tipos de procesos:

- Procesos atómicos. Pueden ser invocados directamente (enviando un mensaje a un agente específico), ejecutados en un único paso y no pueden ser descompuestos.

- Procesos simples. También se perciben como capaces de ser ejecutados en un único paso, aunque no pueden ser invocados directamente. De hecho, pueden verse como procesos abstractos, por ejemplo de uno atómico (una tarea, un servicio en un agente) o como una representación simplificada de proceso compuesto.

- Procesos compuestos. Son descompuestos en sub-procesos, los que pueden ser definidos a su vez como atómicos, simples o compuestos. De esta manera la funcionalidad del servicio se desarrolla recursivamente como una estructura jerárquica compuesta.

De esta manera, el proceso de servicio también es la base para la definición de servicios compuestos. Aunque aquí el enfoque tiene una naturaleza semántica, esencialmente es el mismo que se usa para este propósito desde una perspectiva de comportamiento, en el

contexto de la composición de servicios basada en orquestación [132].

Los procesos de servicios son inherentemente compuestos, concebidos como una agregación de tareas, la que puede ser definida como un proceso simple, según la definición anterior. Aún más, como los servicios los proveen las organizaciones, están descritas naturalmente como la combinación de una serie de procesos más simples, la que debe estar implementada por diferentes roles en la organización. Estos roles son las unidades que actúan como proveedores de servicios para esos sub-procesos, los que por tanto también pueden ser descritos como procesos de servicios.

Hay una relación implícita entre estas estructuras recursivas: procesos (compuestos) pueden ser provistos como servicios por unidades organizacionales (compuestas); cuando estos procesos son descompuestos, los sub-procesos resultantes pueden ser provistos a su vez por otras unidades. Esto es, sub-procesos de un proceso compuesto podrían ser provistos por los miembros (unidades) de la organización compuesta que provee en el nivel superior. Cuando esto sucede, la estructura recursiva de los procesos simula la estructura recursiva de las organizaciones. Sin embargo esto no es estrictamente necesario, y no es una restricción de la plataforma.

Lo opuesto también es factible, comenzando por una tarea simple, un método vertical de composición podría ayudar en la definición de la jerarquía organizacional, mientras al mismo tiempo define los procesos complejos resultantes. Como en el caso de las organizaciones, la recursión finaliza al nivel del agente. Eventualmente, los procesos abstractos (o simples) no se describen más como compuestos sino como procesos atómicos, y por lo tanto estos son tareas, por definición. Y, de nuevo por definición, las tareas son provistas por los agentes, que también son las unidades organizacionales atómicas.

3.2.3. Las Tecnologías del Acuerdo

Las millones de interacciones y transacciones actuales, tanto en ámbitos privados como públicos, a nivel de negocios, entretenimiento, burocracia, etc., están siendo realizadas con medios computacionales a gran escala. Las redes formadas por estos elementos son cada vez mayores, más abiertas, heterogéneas y con entidades que ingresan o salen de su entorno según ciertas condiciones. Son cada vez más necesarios nuevos mecanismos y

técnicas [29, 28] para que sus estructuras puedan adaptarse de tal forma que todos los requerimientos sean cumplidos, tanto funcionales como no funcionales.

La visión de las *Tecnologías del Acuerdo* (Agreements Technologies - AT) se refiere a los sistemas distribuidos abiertos en los cuales las interacciones entre los elementos computacionales (agentes) están basados en el concepto del *acuerdo* [173] y que trasciende los límites de las primeras etapas de desarrollo de los sistemas software. Son los acuerdos que deben conseguirse en "tiempo de ejecución" de los sistemas, es decir, que no se tratan de los previstos en las fases de análisis y diseño, ya que en sistemas abiertos, distribuidos, heterogéneos y a gran escala no sería posible determinar una estrategia unificada y prediseñada para enfrentar el dinamismo del entorno y del propio sistema.

Como este trabajo de tesis se desarrolla dentro del contexto de AT, la noción central de la propuesta es, precisamente, el *acuerdo* entre entidades computacionales autónomas, los agentes software. Y más específicamente, en una *coordinación basada en acuerdos*. Esta idea está concebida como una construcción arquitectónica entre *organizaciones* en niveles superiores pero también entre *agentes* en los niveles inferiores. El proceso debe ser consistente con el contexto normativo en el que se encuentran los agentes, y que una vez aceptado el acuerdo se les permita solicitar los servicios, así como también les sean requeridos los propios, en su debido momento, por los otros agentes [172].

Para lograr la coordinación basada en acuerdos deben considerarse temas claves que aporten las capacidades necesarias a los agentes. En [172, 210] se proponen métodos y mecanismos provenientes de distintos campos, los que resultan de importancia para dar soporte a esos acuerdos.

Estos temas claves pueden ser conceptualizados como capas en una estructura piramidal o en "torre" (ver figura 3.1), donde cada nivel provee funcionalidad y entradas al que se encuentra por encima. Por consiguiente, el acuerdo debe ser visto como una estructura de capas por definición, y si a cierto nivel se ha llegado a un acuerdo, los individuos ubicados en niveles inferiores deben respetarlo a su propio nivel. Básicamente, si una organización llega a un acuerdo, los agentes agrupados en ella deben respetar los términos de ese acuerdo.

Figura 3.1: Estructura original de las Tecnologías del Acuerdo [173]

La estructura piramidal determina el conjunto de capas que define la esencia conceptual de un *acuerdo*. Haciendo el análisis de las capas, de abajo hacia arriba, tenemos [173]:

- *Semántica.* Es la capa que da soporte a las superiores, ya que las cuestiones semánticas influyen sobre todas ellas. Al tratarse de acuerdos donde intervienen sistemas abiertos, distribuidos, heterogéneos y a gran escala, los sistemas contarán, en el mejor de los casos, con su propia ontología, que no necesariamente será compartida por los demás. Es así como surge la necesidad de lograr la alineación semántica de las ontologías [21], para evitar las no correspondencias y tener un entendimiento común, por ejemplo, de las normas que regulan las interacciones entre agentes. El impacto de la semántica se nota en al menos dos niveles diferentes: en el *dominio semántico*, que reúne elementos que comparten la misma ontología, o al menos una compatible que les permita comprenderse mutuamente; y la *interacción semántica*, que define la "arquitectura semántica", una estructura implícita que tiene en cuenta los conectores y conexiones semánticos (¿con quién deseo hablar?, ¿con quién necesito hablar?, ¿con quién me han ordenado hablar?, etc.).

- *Normas.* Este nivel está relacionado con la definición de convenciones y reglas que determinan las restricciones que deben satisfacer los acuerdos y los procesos para lle-

gar a ellos por parte de los agentes [31]. Las normas pueden involucrar roles estructurales que afectan o controlan el comportamiento de dichos agentes, entrelazados con el dominio semántico. Asimismo ciertos principios vistos como abstracciones de normas pueden definir determinados contextos normativos [32]. Las normas también están ligadas con *patrones de interacción*, al tener cierto grado de consenso y además asociárseles permisos, obligaciones, sanciones, etc. Como es natural, las normas pueden evolucionar y modificarse con el tiempo, tanto por fuerzas externas como por una acción concertada de los agentes, lo que inducirá a los mismos, por ejemplo, a negociar los nuevos términos, razonar sobre la adopción de las nuevas normas, o en caso negativo una posible salida del SMA, etc.

- *Organizaciones.* Las organizaciones implican una súper-estructura que restringe la manera en que se consiguen los acuerdos, los que una vez logrados permitirán que un grupo de agentes con distintas capacidades esté en condiciones de resolver problemas mucho más complejos. Estas estructuras definen la arquitectura "social" que a su vez afecta a los niveles inferiores, al fijar la propia de los agentes, las capacidades de sus roles y las relaciones entre ellos [17]. Necesidades organizacionales definen estructuras, pero también hay cuestiones organizacionales que definen roles arquitectónicos, como por ejemplo cambiar de topología, de una organización de tipo jerárquica a otra de estructura plana al disminuir la cantidad de integrantes. Deberían poder responderse a preguntas del tipo "¿quién pertenece a determinada organización?", "¿qué organización puede resolver esta clase de problema?", "¿este tipo de organización es la más adecuada?", entre otras.

- *Argumentación y Negociación.* Ambos pueden verse como protocolos que definen la estructura de un acuerdo entre agentes, respetando las limitaciones impuestas por las organizaciones y sus normas [131]. Una negociación entre organizaciones forma un acuerdo a gran escala, por ejemplo una *convención* o un *acuerdo colectivo*, de tipo formal (B2B, Business-to-Business); si la negociación se produce entre individuos estamos ante un acuerdo de bajo nivel, un *pacto* o un *acuerdo individual*, y de tipo informal (P2P, Peer-to-Peer); por último, entre una organización y un individuo se puede generar un acuerdo de "tamaño medio", como por ejemplo un *contra-*

to o un *acuerdo legal*, el que también puede ser de tipo formal (B2C, Business-to-Consumer). La negociación no solo debe ser vista como un proceso de intercambio de utilidades, sino también como de intercambio de información, complementado con los enfoques basados en argumentación para facilitar y acelerar la consecución de los acuerdos [188].

- *Confianza.* En el nivel superior encontramos la *confianza*. Talvez la "menos" arquitectónica de las capas pero que también define contextos y relaciones, además de especificar interacciones para confiar (o desconfiar) por medio de protocolos. Los agentes necesitan modelos y mecanismos de confianza que resuman la historia de los acuerdos y las ejecuciones posteriores de los mismos, de tal forma que les permita decidir si renovarlos o no [117]. También si es conveniente continuar confiando en las entidades intervinientes, o por el contrario si es necesario seleccionar otras. Estos modelos y mecanismos de reputación son necesarios ya que permiten recoger ciertas mediciones que se utilizarán a la hora de lograr los acuerdos, y con el fin de incrementar la confianza entre los participantes para construir relaciones a largo plazo entre ellos [211]. Es así que un acuerdo duradero entre las mismas entidades crea la sensación de confianza (seguridad) y da una noción de la reputación de los agentes. Un acuerdo debe construirse por encima de esta capa, a fin de conseguir relaciones confiables.

Estos cinco niveles no deben analizarse de manera aislada, ya que aunque queda claro que el flujo de información más importante iría de abajo hacia arriba, las distintas capas pueden beneficiarse unas de las otras. Consideremos por ejemplo la *negociación*, que es un tema que se desarrolla en varios planos. En el más alto define cómo dos organizaciones separadas negocian para crear un acuerdo (lo que podría generar una nueva organización), pero en otros planos también describe cómo dos (o más) agentes se asocian creando una de ellas, e incluso la negociación de un *contrato* entre una organización y un agente. Tiene cuestiones normativas y de semántica, pero obviamente también interviene la confianza. Los modelos organizacionales podrían ser modificados si hay cambios en las normas o para aprovechar algún método de negociación. Las modificaciones en los modelos de confianza también pueden influir de manera importante sobre las normas y por ende sobre la estruc-

tura de las organizaciones [173].

Es importante notar que, en base a lo expuesto, el acuerdo puede considerarse que se encuentra "embebido" en todos los niveles de la estructura presentada. Sin embargo, es posible concebir un cambio de visión de la estructura en torre descrita anteriormente a una figura con estructura *multi-facetada*, donde cada faceta representa a las capas analizadas. El acuerdo influye (y es influenciado por) todas las facetas (o niveles). En este sentido las facetas se entrelazan (figura 3.2) aunque el acuerdo continúa siendo una estructura de capas, y de hecho las capas se unen en ambos sentidos: el acuerdo es una estructura transversal, que mantiene una relación bidireccional con los elementos que contiene.

El *acuerdo define la arquitectura*, en el sentido que solo aquellos elementos que *aceptan/acuerdan* estar unidos o conectados pasarán a estar contenidos en la estructura; pero al mismo tiempo la *arquitectura define el acuerdo*, ya que canaliza las fuerzas del entorno y provee una estructura concreta, definiendo roles que debe ser cubiertos por elementos específicos. El acuerdo es *moldeado* (formado) por estas fuerzas, pero su existencia también *moldea* la reacción a ellas, y a su vez modela la futura evolución del sistema.

Figura 3.2: Estructura multi-facetada de un Acuerdo

La perspectiva *multi-facetada* (el "pentágono" de la figura) no intenta reemplazar a la estructura en torre, ya que la arquitectura descrita anteriormente es jerárquica en muchos

sentidos y podemos considerar un acuerdo formado no solo por capas sino también como una construcción con múltiples facetas. Las capas proveen de una separación lógica de conceptos, naturalmente no son niveles físicos. Incluso pueden acercarse a la idea de ser considerados como *fractales*, donde las capas son fachadas compuestas por servicios, por lo que cierto servicio podría estar formado por elementos de múltiples niveles. Cuando estos niveles están globalmente definidos no pueden existir límites rígidos entre ellos y por lo tanto toda separación debe ser necesariamente lógica. De otra manera, una estructura puramente recursiva podría fácilmente llevar a una contradicción.

Para comprender la estructura *multi-facetada*, por ejemplo a un nivel de implementación, es suficiente considerar que un agente puede estar agrupado en (o unido a) varias organizaciones, incluso al mismo tiempo. La estructura está relacionada a una jerarquía organizacional, pero aún cuando debe ser considerado en el momento de publicar servicios, esto no es requerido al tiempo de proveerlos, haciendo corresponder funciones o roles de una organización con agentes.

El enfoque propuesto provee los elementos requeridos para construir una arquitectura adaptativa, de modo que para definir un *acuerdo emergente* solo se requiere la identificación de los patrones estructurales y el conjunto de protocolos inter-niveles. Asimismo pueden plantearse mayores refinamientos, como considerar meta-elementos, o definir agentes específicos para realizar tareas de apoyo para el propio acuerdo, tales como sensores, observadores, controladores, planificadores, etc.

3.3. Adaptación y las Tecnologías del Acuerdo

Desde los inicios la necesidad de que los sistemas software puedan adaptarse a los cambios, tanto de requerimientos como del entorno en que son utilizados, es una demanda a la que los desarrolladores se han enfrentado con asiduidad. En la actualidad, con la creciente complejidad de los sistemas se ha enfatizado aún más en esta necesidad, tanto es así que la posibilidad de adaptación a los cambios debe ser tenida en cuenta desde los primeros pasos del desarrollo.

También ya hemos comentado que la utilización de SMA en la resolución de problemas complejos es una alternativa válida en la IA desde hace mucho tiempo. Sin embargo, para conseguir un alto nivel de generalidad es necesario que puedan lograr una "real" auto-adaptación, es decir, que no solo modifiquen su configuración sino también el tipo de entidades que los forman y, en relación con esto, es importante lograr la *coordinación* de sus integrantes. Un avance en este tema ha sido la agrupación de los agentes en *organizaciones*, como se ha visto en los distintos casos del Capítulo anterior.

Como el trabajo de esta tesis está orientado a lograr la *coordinación basada en acuerdos* de las organizaciones de agentes, la adaptación será tenida en cuenta tanto desde la formación de las organizaciones como en los procesos de auto-organización, reorganización y evoluciones posteriores. Esto puede interpretarse, según [183], como casos en que la organización no es la adecuada para conseguir el objetivo, o también que su estructura no es la apropiada para adaptarse al entorno. Por ejemplo el sistema podría realizar una serie de reorganizaciones hasta encontrar un punto óptimo y detenerse, por supuesto que para que esto sea posible es necesario poseer la capacidad para detectar la configuración nueva y su estabilidad. Si no es suficientemente estable se debe continuar con la evolución hasta encontrar una configuración que responda a los requerimientos. Esto, claramente, puede ser un proceso continuo ya que la situación misma está evolucionando.

La estructura *adaptativa* buscada estará fuertemente ligada a los conceptos de AT vistos en la sección anterior (3.2.3). La meta es lograr, idealmente, una reconfiguración automática del sistema, en particular en un entorno orientado a agentes y centrado en organizaciones. Por ejemplo, las normas utilizadas para definir restricciones y las organizaciones para definir los ámbitos ayudarán a establecer la estructura; incluso distintas negociaciones pueden ser utilizadas para activar nuevos acuerdos que modifiquen dicha estructura.

3.3.1. Hacia la Arquitectura Básica

La necesidad de continuas adaptaciones se hace notoria a medida que los sistemas software deben conciliar los cambios, y estos a su vez los vuelven más y más complejos. Deben enfrentar dichos cambios, tanto cuando el entorno o el objetivo del sistema se modifican

como cuando hay nuevas exigencias de funcionalidad, rendimiento, seguridad, compatibilidad, etc.

Básicamente existen requisitos que resultan difíciles de resolver con los sistemas usuales [200], como es el caso de considerar la complejidad "social" [92] de los sistemas actuales, o aplicaciones cada vez más distribuidas, heterogéneas y descentralizadas, componentes computacionales que formen coaliciones, entornos distribuidos de ejecución conjuntamente con métodos de coordinación y políticas de control, seguridad y privacidad, entre otros.

Si hablamos de SMA algunos ejemplos de cambios que generan la necesidad de una adaptación pueden ser cuando los agentes no cumplen las expectativas esperadas, o si están agrupados en organizaciones y estas no pueden cumplir con las metas asignadas.

Partiendo de la premisa que un sistema para adaptarse debe "comprender" *qué hace*, *cómo lo hace*, *cómo evaluar su performance*, e incluso *cómo responder a las condiciones de cambio* [143], la identificación de los requisitos no es tarea sencilla. Nuestra propuesta es utilizar *organizaciones adaptativas de agentes*, y en este caso, y conforme a [70], podemos considerar tanto cambios *dinámicos* como *estructurales*, los que nos permitirán definir los requisitos a tener cuenta para lograr las adaptaciones del sistema.

En el primer caso, consideramos que son *cambios dinámicos* cuando un nuevo agente ingresa al sistema o cuando lo abandona, teniendo en cuenta las condiciones de ocurrencia (por ejemplo aceptar los acuerdos inherentes y así los nuevos integrantes podrán jugar los roles correspondientes, o si se trata de un reemplazo, se les podrán asignar las tareas de otro/s agente/s), o también cuando se modifican los patrones de interacción entre los agentes, los protocolos de comunicación que se habían acordado, etc. [173]

Por otro lado, como *cambios estructurales* podemos distinguir una *auto - organización* de una *reorganización*, dependiendo de si los cambios emergen a nivel local entre agentes generando cambios globales en la organización o si se producen en la propia estructura [173].

Estos cambios tendrían distintos enfoques [200], con respecto a las unidades organizacionales los agentes podrían crear o eliminar organizaciones, o producir cambios en las

configuraciones o topologías de estas últimas; con respecto a los roles, podrían agregarse, eliminarse o actualizarse por la especialización o generalización de los ya existentes, asimismo las condiciones para registrar o de-registrar un rol en una organización pueden cambiar con el tiempo, e igualmente la cantidad de agentes que pueden desempeñar cierto rol; las normas no solo restringen la formación de organizaciones y su funcionamiento "institucional" sino también las relaciones entre los agentes (como es el caso de los contratos), por lo que sería decisivo determinar el momento en que aparecerán nuevas normas, y también que exista la posibilidad de modificarlas en tiempo de ejecución. Con respecto a las relaciones entre roles debería ser posible que cambien con el transcurso del tiempo, al igual que las condiciones para activarlas o desactivarlas y la cantidad de roles que intervienen, y como estos cambios también afectan la topología del sistema es necesario que se verifiquen si dichos cambios son consistentes.

Otras modificaciones en las organizaciones vendrían ocasionadas directamente por el entorno y sus elementos, como los recursos, artefactos representativos, otros sistemas, etc. También pueden analizarse ciertos cambios funcionales que influirán en nuestras organizaciones, por ejemplo si cada organización está asociada con un objetivo todo cambio en este (como ser la separación de un objetivo complejo en otros más simples, la aparición de otras metas relacionadas con cambios en los roles, etc.) será un disparador para la adaptación de la organización y por ende del sistema; o respecto a los servicios y los roles, como ambos están relacionados, los cambios en estos últimos se deben reflejar en los servicios correspondientes, incluyéndose la especialización o generalización de los mismos [200].

Las organizaciones adaptativas últimamente constituyen un tópico importante de investigación en el dominio de los SMA, por lo que para hacer frente a los requisitos de adaptación la utilización de las mismas en nuestra propuesta resulta más que apropiada ya que presentan propiedades particularmente útiles [173]: la organización va a necesitar adaptarse tanto por causa de los cambios en su entorno como también por los que se produzcan en sus metas, objetivos, requisitos internos, etc. Asimismo la organización es considerada un sistema abierto ya que los agentes pueden entrar y salir de ella en cualquier momento, y además algunos de los roles que juegan son responsables en las decisiones de cambios.

Para lograr una verdadera adaptación se necesitan mecanismos que produzcan los cam-

bios no solo en la de composición de las organizaciones, sino también en el tipo de elementos que las integran. De esta manera nuevas construcciones arquitectónicas, al igual que nuevos elementos, podrán surgir –emerger– de manera dinámica en las unidades organizacionales.

Es así que para llevar adelante el proceso de diseño de la solución se han propuesto, como ya se ha dicho, tres *conductores (o factores) arquitectónicos* como principales abstracciones: la arquitectura será *orientada a servicios*, *basada en agentes*, y *centrada en organizaciones*. El marco tridimensional definido por estas perspectivas permitirá dar forma a la estructura conceptual del sistema y su arquitectura subyacente. Sin perder de vista, por supuesto, los requisitos usuales relacionados con la calidad, como son el manejo de errores, la eficiencia, el consumo de recursos, el balanceo de carga, entre otros. En los siguientes apartados se presentarán la propuesta de solución, la plataforma utilizada como base y su posterior evolución.

3.3.2. Resumen de la Propuesta

La solución arquitectónica que se propone debe proveer a nuestras *organizaciones* de las capacidades de *adaptación*. Esto representa una característica relevante y poderosa, aunque usualmente es compleja y difícil de obtener. La adaptación no debe estar basada en una estrategia predefinida y cerrada, por lo que consideramos debe tener un alto grado de emergencia. En consecuencia, según nuestro enfoque, la arquitectura del sistema debería presentar comportamiento *emergente*.

Uno de los problemas subyacentes al utilizar SMA, como se ha mencionado, es la *coordinación*, por lo que al utilizar organizaciones se está elevando el nivel de abstracción y es así como la organización pasa a ser el elemento unificador de la propia arquitectura. De esta manera se produce un cambio desde ese punto de vista y la coordinación pasa a ser considerada por "ámbitos". La jerarquía recursiva organizacional es la que hace posible definir, simultáneamente, una arquitectura orientada a servicios, en el nivel más alto, y basada en agentes en el más bajo, por lo que el concepto de organización será el nexo entre am-

bas perspectivas. Son grupos estructurados de agentes que ofrecen servicios, se unen para lograr objetivos y mantener esa unión por medio de mecanismos de coordinación. Una función importante y deseable en un sistema abierto es la habilidad de cambiar de estrategia de coordinación *al vuelo (on-the-fly)* y así lograr la adaptación satisfactoria del comportamiento del sistema y de sus elementos. Esto también refuerza la utilización de la tecnología de agentes y del paradigma de SMA en los dominios de aplicación muy complejos.

En vista de las ventajas que proporciona utilizar las organizaciones como abstracción para lograr coordinación, se propone además que estas organizaciones estén basadas en *acuerdos* para lograr la adaptación buscada. De esta manera se conseguirá trabajar con las ya nombradas *organizaciones adaptativas basadas en acuerdos*, obteniéndose estos acuerdos por medio de procesos que involucran las capas lógicas explicadas anteriormente (3.2.3).

Como soporte para conseguir esta adaptación se propone el uso de patrones de software, desde el punto de vista arquitectónico. Es de esperarse que no siempre se pueda hacer frente a todos los requisitos planteados con un solo patrón, por lo que una adecuada selección y combinación de ellos podrá resolver la complejidad del problema. A esta combinación y uso conjunto de los patrones, como si de una coreografía se tratara, es lo que denominamos *sistema o lenguaje de patrones* que será presentado en el Capítulo 5.

Es importante notar que nuestro enfoque no solo propone el uso de los patrones de adaptación, sino también el tratamiento del dinamismo del entorno, y del sistema en sí, desde el mismo comienzo, cuando se están definiendo los objetivos y la composición del sistema. Esto puede verse con la propuesta de *ciclo de vida de estructuras auto-organizadas* que se define en 4.3.1. Este ciclo parte de uno o más agentes, se analizan los objetivos perseguidos (que generalmente son mayores que los objetivos individuales de los agentes), se aplican los patrones adaptativos necesarios y se produce una serie de iteraciones hasta que todos los integrantes del grupo, acuerdos mediante, generan una estructura preliminar: la *iniciativa* (4.2.2). Esta *iniciativa* puede evolucionar hasta llegar a originar una organización, la que es adaptativa por formación. Esta organización también podrá integrar organizaciones mayores, siempre basadas en acuerdos y participando en la obtención de objetivos superiores

a los propios.

3.4. La Arquitectura Seleccionada

Podemos definir a la arquitectura que da apoyo al modelo de nuestra propuesta como un SMA abierto, orientado a servicios (AOS) y centrado en organizaciones. Los componentes básicos de la arquitectura son los agentes software que brindan servicios de manera individual o grupal, y la estructura que los aglutina es la organización. Esta naturaleza multinivel y de múltiples puntos de vista debe ser específicamente habilitada por la arquitectura técnica, por lo consideramos que el lugar lógico para proveer este soporte es el *middleware*, ya que la plataforma se concibe como un sistema distribuido.

Para que ambas definiciones, SMA y AOS, puedan ser utilizadas en conjunto, una de estas estructuras y conceptos centrales (agente, servicio) debería ser la base de la otra. Aunque las arquitecturas de agentes tienen algunas características similares a las orientadas a servicios, como por ejemplo utilizar directorios donde anunciar sus funcionalidades y en los cuales otros las localizan, los agentes son autoconscientes y mediante modelos, aprendizaje e interacciones pueden saber de la existencia de otros y aprovecharse de sus capacidades a fin de adaptar sus servicios al cliente, a diferencia de los servicios propiamente dichos. De igual manera los agentes pueden mediar cuando se encuentren diferencias en el uso de las ontologías, y también actuar de manera proactiva, a diferencia de los servicios que se encuentra en fase pasiva hasta su invocación [89].

En base a lo anterior, la alternativa que hemos seleccionado es la de *agentes soportando servicios*: por un lado, los agentes son entidades computacionales bien conocidas, implican cierta granularidad y cumplen con estándares existentes [93, 232, 94]; por el otro, los servicios están definiendo su rol y relevancia y ya tienen numerosos estándares [34, 49, 80, 148], estos están definidos en diferentes niveles de granularidad y la mayoría solo imponen restricciones en las interfaces de los servicios. Por tanto, es fácil concebir un servicio como una manera de presentar las capacidades operacionales de un agente, o incluso de un grupo de agentes, reunidos en una organización. En resumen, la elección es tener una plataforma definida como una AOS, construida por sobre un SMA.

3.4.1. Modelo de la Arquitectura

La propuesta de esta tesis se construye sobre trabajos ya existentes, como lo son THOMAS [170, 133, 14] y GORMAS [12, 15, 79].

THOMAS consiste en una arquitectura abstracta para SMA abiertos y de gran escala. Se basa en un enfoque orientado a servicios y permite el diseño de organizaciones virtuales. La arquitectura está formada esencialmente por un conjunto de servicios estructurados modularmente. También hace uso de la arquitectura FIPA [232], expandiendo sus capacidades con respecto a organizaciones y servicios. Por su parte, GORMAS propone un modelo de organización que permite describir sus principales aspectos, como su estructura, funcionalidad, normalización, dinamicidad y entorno. Esto facilita su análisis y diseño, y asimismo permite especificar, principalmente cuáles son los servicios que ofrece la organización, cuál es su estructura interna, qué normas rigen su comportamiento, etc. [12]. Con el concepto de unidad organizativa se representa a la agrupación de agentes que llevan a cabo determinadas tareas específicas y diferenciadas y que siguen ciertos esquemas de comunicación y cooperación predefinidos. Tiene además un carácter recursivo, pues no sólo contendría agentes, sino también otras unidades organizativas, actuando como entidades atómicas. Se asocian a la unidad organizativa aspectos tanto de entidad global y atómica como de agrupación de sus entidades internas, relacionadas entre sí en base a sus roles, funcionalidad, recursos y entorno [79].

La Figura 3.3 presenta un esquema de la arquitectura THOMAS, incluyendo su *middleware* [13]. Se estructura en tres niveles, aunque no son estrictamente capas. Estos niveles están ortogonalmente soportados por componentes que a su vez están incluidos como parte de tres diferentes subsistemas. No existe total coincidencia entre estos y los niveles conceptuales porque el subsistema *Platform Entities Management* también está estructurado en "capas": sus componentes son el *Organization Management System (OMS)* y el *Agent Management System (AMS)*, los que son utilizados para proveer de capacidades específicas a la plataforma. Los niveles en que se estructura inicialmente la arquitectura son:

- *Platform Kernel.* Es el nivel inferior que soporta todo lo que se ubica por encima de él. Básicamente es el *kernel* del *middleware* e incluye tanto a la capa de red *(Network*

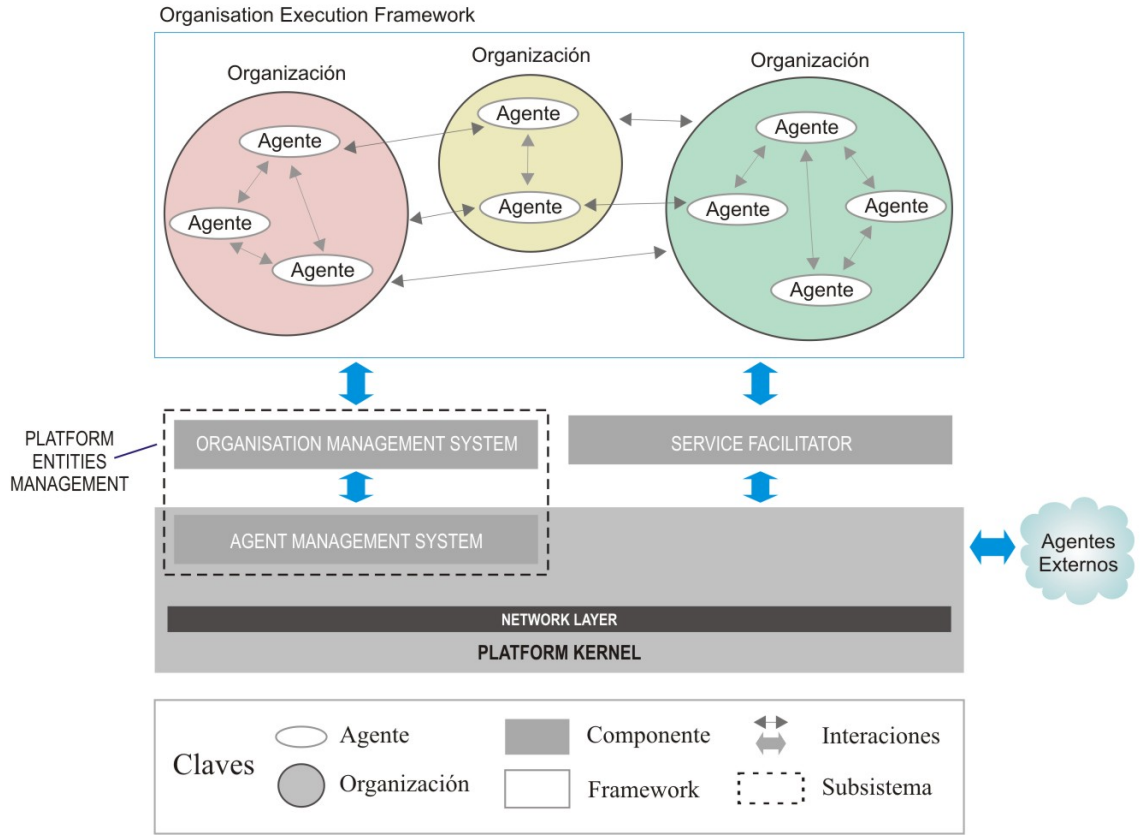

Figura 3.3: Arquitectura técnica de THOMAS (inspirada en [13])

Layer) como al componente *AMS*. Conceptualmente provee de todas las capacidades requeridas de una arquitectura compatible con los estándares de FIPA. Por tanto a este nivel de descripción la plataforma es un SMA abierto como era requerido.

- *Service & Organization Management.* Es un nivel conceptual compuesto por el *OMS* y el componente *Service Facilitator (SF)* y no se lo presenta como un subsistema ya que carece de una interfaz unificada y por lo tanto no puede ser categorizado como una "capa". Sin embargo mantiene el resto de las restricciones de una arquitectura en capas ya que ambos presentan interfaces al *Platform Kernel* y en conjunto proveen todas las caraterísticas y abstracciones relevantes al *framework* de ejecución.

- *Organization Execution Framework.* Es el "espacio" donde todas las entidades computacionales *viven* y realizan sus actividades. Los niveles citados en primer término de-

ben proveer a este de todas las capacidades requeridas. Por lo tanto, los agentes y sus organizaciones, y los servicios que estos ofrecen, están conceptualmente localizados en este *framework*. Resumiendo, cada aplicación debería ser concebida, diseñada y ejecutada en este nivel de abstracción, y en consecuencia el resto del sistema puede ser considerado como *middleware*.

Los tres principales componentes de la plataforma que se mencionaron anteriormente son los que proveen al *middleware* de la mayoría de sus capacidades. Ellos son:

- *Agent Management System (AMS).* Componente que provee todas las capacidades y funcionalidades requeridos por los agentes (crear, comunicar, evolucionar, destruir). Es el módulo que garantiza la compatibilidad con FIPA y también es el subsistema que da soporte a la perspectiva *basada en agentes* de la definición de la arquitectura base de este trabajo.

- *Organization Management System (OMS).* Este componente provee todas las capacidades y funcionalidades requeridas por una organización, concebida tanto como un grupo de agentes como una unidad atómica independiente. Es el módulo que mantiene unido a todo el sistema y además da soporte a la perspectiva *centrada en organizaciones* de la definición de la arquitectura mencionada.

- *Service Facilitator (SF).* Es el componente que provee las capacidades y funcionalidades para que una selección de operaciones en una organización se comporte como un servicio unificado. Es el módulo que provee una perspectiva alta del sistema y es el que soporta el punto de vista *orientado a servicios* de la definición de la arquitectura dada anteriormente. Tratándose de *middleware* una *"facility"* generalmente es una capacidad incorporada por sobre los servicios comunes de la plataforma. La existencia del *SF* muestra que aquí la *orientación a servicios* es un nivel más de abstracción, específicamente creado para proveer una perspectiva adicional del sistema y conceptualmente situado a un nivel más alto.

Como se ha explicado anteriormente, el subsistema *Platform Entities Management* no define un nivel pero puede verse como formado por capas (Figura 3.3). Su función es proveer todas las capacidades requeridas para gestionar cada entidad computacional activa, es

decir, los agentes y sus organizaciones. El *AMS* debe pertenecer a este subsistema porque una organización (o también unidad organizacional), al más bajo nivel, es siempre un agente, y por tanto sus funciones están necesariamente acopladas. Sin embargo el *AMS* también es una parte integral del *Platform Kernel*, ya que de otra manera no podría ser compatible con FIPA. Por lo tanto está organizado ortogonalmente con respecto a la estructura multinivel ya descrita.

La parte de la arquitectura técnica denominada *Organization Execution Framework* define un *framework* orientado a servicios ya que es soportado por el *SF*. En este espacio los servicios son provistos por unidades organizacionales, y como son gestionados por el *OMS* la orientación es hacia una estructura centrada en organizaciones. Esta estructura se construye por encima de un SMA abierto donde los agentes, como parte de una organización, proveen los servicios requeridos.

Esta arquitectura provee una explicación satisfactoria de la perspectiva multinivel y multiaspecto que permite conciliar las visiones previamente descritas. Esta plataforma se presenta como punto de partida para el desarrollo propuesto de una estructura de *coordinación basada en acuerdos*.

Hemos visto en 3.2.2 que los servicios juegan un papel fundamental en esta propuesta y los conceptos expuestos se corresponden con los servicios que ofrece esta arquitectura. De manera similar, el otro elemento activo importante es la organización (ver 3.2.1), que además funciona como concepto unificador de la arquitectura. La jerarquía organizacional es la que hace posible definir, simultáneamente, la arquitectura como orientada a servicios (a alto nivel) y basada en agentes (a bajo nivel).

La noción de organización define una estructura recursiva, ya que una organización estará compuesta por unidades o unidades organizacionales. Una unidad es una entidad activa con un comportamiento definido y externamente observable, y puede tener tanto una naturaleza colectiva (donde la unidad es una organización en sí misma) como una autónoma (cuando la unidad es también un agente). Cuando la unidad es una organización puede facilmente verse la estructura recursiva.

El modelo se propone como basado en organizaciones por lo que la unidad provee soporte para el resto de ellas. Con esta concepción, la unidad es el substrato que soporta

tanto el agrupamiento de agentes como la definición de los servicios, describiendo la idea recursiva que explica la relación entre ambas perspectivas [15, 79]. Para una descripción más detallada de la arquitectura THOMAS, sus capacidades y funcionalidades se pueden consultar [170, 13, 133, 14].

3.4.2. Evolución de la Plataforma

La plataforma continúa su evolución a fin de obtener características más avanzadas para permitirle mayores posibilidades de adaptación. Se buscan desarrollar sus capacidades y funcionalidades tendientes a conseguir el grado de automatismo deseado y de esa manera poder hablar de una real auto-adaptación.

La arquitectura propuesta, basada en agentes, orientada a servicios y centrada en organizaciones, ofrece servicios dinámicos para que los agentes puedan asumir o abandonar roles en tiempo de ejecución, o demandar u ofrecer servicios acorde a las normas establecidas. De hecho el *SF* es una mejora del *Directory Facilitator* del estándar FIPA ya que se pueden registrar servicios ofrecidos y demandados, tanto de agentes como de organizaciones (o también unidades organizacionales).

Los progresos sobre la arquitectura se están dado a nivel de los servicios con su adaptación a la especificación OSGi [169]. Este es un sistema modular y plataforma de servicios para el lenguaje de programación Java que implementa un modelo dinámico y completo. Básicamente consiste en modularizar aplicaciones o componentes en entidades más pequeñas denominadas *bundles*. Estas entidades, de manera remota, pueden ser instaladas, iniciadas, detenidas, actualizadas o eliminadas "al vuelo" (*on-the-fly*) y dinámicamente, proporcionando la posibilidad de cambiar el comportamiento del sistema sin tener que interrumpir su operación. El registro de servicios permite a los *bundles* detectar el agregado o eliminación de servicios y promover la adaptación correspondiente [169].

Entre los servicios que provee la especificación OSGi destacamos el *Service Tracker* (trazador de servicios), y especialmente para nuestra propuesta, ya que permite rastrear servicios registrados en la plataforma, además de verificar que los que ya han sido provistos continúan o no disponibles. En un sistema que utiliza *bundles*, tanto los producto-

res como consumidores de servicios pueden aparecer y desaparecer dinámicamente. Para realizar el paso de mensajes, síncronos o asíncronos, entre dos entidades que pertenecen a distintos *bundles*, la especificación proporciona una herramienta: el patrón *Whiteboard* (pizarra). Este utiliza el registro de servicios de OSGi para mantener identificados los *listeners* ("oyentes") del sistema. Uno de los beneficios de esta herramienta es que el control del ciclo de vida de productores y consumidores de eventos se delega en la plataforma, que notifica a consumidores la desaparición de productores y viceversa.

También como parte de esta evolución, en [198] se plantea proveer capacidades de reorganización incorporando a la plataforma un enfoque basado en planificación. Para ello se han desarrollado servicios de planificación que han sido modelados e integrados dentro de los SMA. Estos involucran mecanismos de auto-adaptación que facilitan la asignación automática de tareas dentro del SMA. De esta manera la arquitectura puede hacer frente a una reorganización, por ejemplo cuando se introduce un nuevo agente, llevando adelante las adecuaciones necesarias de las planificaciones.

Las investigaciones, incluidas como parte del proyecto OVAMAH [200, 201, 199], se desarrollan ampliando los objetivos de la plataforma THOMAS. Además de proveer la tecnología necesaria para el desarrollo de organizaciones virtuales en entornos abiertos, permiten dar respuestas dinámicas por medio de la adaptación y/o evolución de las propias organizaciones.

Como ejemplos podemos citar:

- los agentes que integran una unidad organizacional podrán crear (o eliminar) otra unidad, afectando a los grupos del sistema;

- con respecto a las normas, podrán definir en qué momento es necesario agregarlas o eliminarlas;

- el tipo de relación social entre roles podrá cambiar en tiempo de ejecución, tanto las condiciones para activarlas/desactivarlas como la cardinalidad entre roles;

- la topología del sistema (dada por las relaciones) podrá cambiarse (agregar/eliminar relaciones) también en tiempo de ejecución y luego verificarse que este cambio es consistente con los objetivos y el tipo organizacional;

- los servicios podrán ser asociados a nuevos roles que sean necesarios en el dominio;

- dividir un objetivo en metas concretas; etc.

En línea con [71], se busca que las organizaciones dinámicas de agentes tengan capacidad de optimización automática, puedan emerger como grupos para ofrecer servicios combinados, y también evolucionar. La plataforma para soportar la ejecución de sistemas como los propuestos debe ser robusta, eficiente y adaptativa en el tiempo.

3.5. Conclusiones

En este Capítulo se han presentado conceptos y cuestiones tecnológicas relevantes para nuestro enfoque de arquitecturas adaptativas. Básicamente la propuesta ha sido la integración de dos soluciones muy utilizadas para la resolución de problemas complejos, como son los *sistemas multiagente, SMA*, y también la *computación orientada a servicios, SOC*.

Estas dos soluciones, al ser posible aplicarlas en conjunto, son especialmente adecuadas para modelar entes autónomos, heterogéneos, dinámicos, distribuidos y en entornos flexibles y abiertos.

En las secciones correspondientes se han analizado y descrito los elementos fundamentales que otorgan a la arquitectura propuesta las características buscadas: el agente como componente básico, el servicio como noción central, y la organización como estructura que unifica los conceptos. Por lo tanto la arquitectura que se propone está *basada en agentes, orientada a servicios y centrada en organizaciones*.

Se ha visto la importancia del rol que tienen las organizaciones de agentes al ser un enfoque significativo y cada vez más popular para resolver problemas complejos y que requieran de un trabajo altamente colaborativo. Las organizaciones proveen un nivel alto de coordinación del sistema ya que también permiten tener *ámbitos* de coordinación. Algo similar sucede con el rol de los servicios, ya que estos pueden ser ofrecidos tanto por los agentes software como parte de su funcionalidad como también por las organizaciones que estos componen. En definitiva, la jerarquía organizacional es la que hace posible definir la arquitectura como orientada a servicios (a alto nivel) y basada en agentes (a bajo nivel).

Las Tecnologías del Acuerdo (AT) [173] constituyen el contexto general de este trabajo y los mecanismos y métodos que proporcionan permiten lograr que las interacciones entre los elementos computacionales estén basados en el concepto del *acuerdo*. Esta coordinación basada en acuerdos debe ser conseguida en tiempo de ejecución ya que en sistemas abiertos, distribuidos, heterogéneos y a gran escala tener una estrategia unificada y prediseñada para enfrentar el dinamismo del entorno y del propio sistema resulta muy difícil, o incluso imposible.

Junto con los conceptos propuestos por AT, el modelo de la arquitectura estudiada y que puede dar soporte a esta propuesta se construye sobre trabajos ya existentes: THOMAS [133] y GORMAS [15]. Mientras el primero basa su enfoque en los servicios y en organizaciones virtuales para construir SMA abiertos y a gran escala, el segundo propone modelos de organizaciones para describir sus estructuras, funcionalidades, dinamicidad, entorno, etc. Como es habitual en las investigaciones la evolución de los trabajos continúa, por ejemplo en el nivel de servicios y organizaciones dinámicas, planificación de tareas, etc. [201].

Para lograr una verdadera adaptación se necesitan mecanismos que produzcan cambios en la composición de las organizaciones y en el tipo de elementos que las integran. La meta es conseguir que nuevas construcciones arquitectónicas, al igual que nuevos elementos, puedan surgir o emerger de manera dinámica.

El contexto ya ha sido planteado en este Capítulo, en el próximo se describirá la propuesta de este trabajo con mayor detalle.

Capítulo 4

Propuesta: del Acuerdo Emergente a la Arquitectura Adaptativa

4.1. Introducción

Nuestra propuesta adaptativa se presenta como parte del *"sandbox"* [1] del proyecto Tecnologías del Acuerdo (Agreement Technologies - AT) para proveer a los sistemas de capacidades de *auto-adaptación*. Se ha partido del análisis de sistemas software distribuidos, cuyos elementos integrantes tienen capacidades de coordinación, autonomía, cierto grado de inteligencia y razonamiento, entre otras características distintivas: son los llamados *agentes de software*. Estos agentes proveen *servicios* a otros agentes, y además pueden optar por formar parte de una *organización*.

El enfoque de una construcción arquitectónica como solución al problema de coordinación parte de la convicción de que el nivel arquitectónico presenta la abstracción y generalidad necesarios para resolverlo. En nuestra propuesta, *basada en agentes*, *orientada a servicios* y *centrada en organizaciones*, los elementos interactúan entre sí teniendo como base el concepto del *acuerdo* entre pares. Además pueden *negociar* para conseguir

[1] "sandbox" (o caja de arena, en castellano) se refiere a un entorno en el que las operaciones de software y potencialmente peligrosas pueden ser probados en aislamiento, reduciendo así la probabilidad de daño colateral al entorno del sistema principal.

sus metas individuales, que en definitiva contribuirán con las del sistema en general. Los elementos que forman nuestras organizaciones podrán tanto auto-organizarse como reconfigurarse y reorganizarse, a fin de lograr la adaptación deseada.

En las secciones siguientes se tratarán con más detalle los temas relevantes de la propuesta, como por ejemplo el obtener comportamiento emergente del sistema por medio del acuerdo necesario entre los integrantes del grupo, los controles y protocolos que facilitarán la creación de una "semilla" organizacional que llamamos *iniciativa*. De igual manera se presentan el ciclo de vida de estructuras auto-organizadas, los elementos del cambio, patrones de adaptación y operaciones que las organizaciones pueden llevar adelante para obtener el comportamiento emergente buscado.

4.2. El Acuerdo Emergente

Cuando se utiliza un SMA abierto como solución para un problema complejo, es usual que se requiera cierta adaptación del sistema. Al mismo tiempo, la propia estructura debería ser flexible para lograr la coordinación interna entre los agentes.

El enfoque de una construcción arquitectónica como solución al problema de coordinación parte de la convicción de que el nivel arquitectónico presenta la abstracción y generalidad necesarios para resolverlo. Entre los potenciales beneficios que ofrece un enfoque arquitectónico [141] se pueden citar:

- Generalidad: los principios y conceptos subyacentes pueden ser utilizados en una vasta gama de dominios de aplicación. Nuestra propuesta, al utilizar los conceptos de agentes, servicios y organizaciones, resulta muy apropiada para el uso en dominios de aplicación cada vez más complejos.

- Nivel de abstracción: puede ser el apropiado para describir los cambios dinámicos en el sistema. Con el uso de patrones nuestro enfoque permite que el sistema pueda adecuarse favorablemente a los cambios dinámicos propios y del entorno.

- Potencial para la escalabilidad: esto se ve favorecido por el hecho de que las arquitecturas utilizan diversos mecanismos de composición en los casos de sistemas muy comple-

jos. En nuestra propuesta, con la *iniciativa*, se tiene la capacidad de incorporar y remover los patrones que permitirán escalar y modificar su estructura.

- Potencial para un enfoque integrado: debido a que existe una amplia variedad de lenguajes de descripción de arquitectura *(ADL - Architecture Description Language)* que permiten la configuración, implementación y reconfiguración de sistemas. Aunque en nuestra propuesta no se utilice específicamente un ADL, se logran representar formalmente los protocolos que definen los patrones utilizando el Cálculo-π [207] por la precisión y expresividad que proporciona.

Asimismo, consideramos al *acuerdo* entre entidades computacionales como el concepto adecuado para abordar la búsqueda de una *estructura adaptativa*. El objetivo es "descubrir" una estructura apropiada, de modo que emerja como un *acuerdo global*. De esta manera se puede hacer un paralelo entre obtener un *acuerdo global* entre los integrantes del grupo y conseguir el *comportamiento global emergente* del sistema.

En este tipo de sistemas distribuidos, abiertos y a gran escala, las adaptaciones no podrían estar basadas en una estrategia predefinida, o mejor aún, no sería posible tener una estrategia preprogramada para gestionarlas, ya que al ser numerosos los elementos computacionales que interactúan debería estar embebida en cada uno de ellos para hacer frente a las adaptaciones. Esto justifica que el comportamiento no pueda ser totalmente prediseñado sino que debería ser *emergente*, a partir de la dinámica del sistema. También podríamos decir que estamos hablando de *sistemas auto-organizados* si encontramos auto-organización tanto desde la formación de una organización a partir de un "desorden" de entidades como reorganizando una organización que ya existe, debido a nuevas metas, nuevos requerimientos, etc., lo que en definitiva llevan a la adaptación del sistema. Ambos conceptos, la *auto-organización* y la *auto-adaptación* pueden por tanto relacionarse, aunque no necesariamente uno implique al otro. Sin embargo es factible el caso de que un sistema tenga que auto-organizarse como respuesta adaptativa ante algún cambio externo y esto puede implicar la *emergencia* de comportamientos no prediseñados o previstos [58].

Los elementos que participan en un acuerdo deben ser capaces de ajustarse por ellos mismos a los cambios del entorno. El objetivo de estos ajustes es llegar a las metas del

acuerdo, lo que llevará a realizar cambios, no solo en los propios elementos sino también en su configuración, como por ejemplo en una adecuación de los recursos existentes. De hecho, el criterio utilizado para decidir si un agente pertenece a un acuerdo puede ser gestionado de la misma manera: esto define un *acuerdo emergente*, donde no solo una parte del comportamiento emerge de esta situación sino también la propia estructura del acuerdo.

Si idealmente la meta es lograr una reconfiguración automática del sistema, en particular en un entorno orientado a agentes, es necesario que se realicen una serie de reorganizaciones hasta encontrar un punto óptimo y allí detenerse. Para que esto sea posible debe ser capaz de detectar que la configuración nueva es, en cierto modo, "estable". En caso contrario, continuará con su evolución hasta encontrar una configuración óptima o, al menos, un punto de óptimo local en un proceso continuo acompañando los cambios de la situación.

Los mecanismos que pueden favorecer la coordinación emergente, y a posteriori, harían posible definir una organización también emergente y sobre la base de acuerdos, son ya conocidos en el desarrollo de sistemas software: son los llamados *controles* y *protocolos*. Con ellos pueden construirse estructuras básicas y luego, en base a cambios en el entorno, requerimientos, metas, etc., pueden conseguirse estructuras más complejas.

4.2.1. Canalizando el Acuerdo Emergente: *Controles* y *Protocolos*

De manera genérica, y cualesquiera que sean sus objetivos, todo grupo de individuos puede ser ordenado en estructuras más o menos flexibles, tanto si se trata, por ejemplo, de una sociedad (una estructura social) como de una arquitectura de software (una estructura de sistemas). Esta ordenación representa un gran potencial si se realiza de manera apropiada ya que permitirá afrontar numerosas situaciones complejas, sobre todo cuando se tratan de sistemas a gran escala. Las *relaciones* entre estos individuos pueden ser implícitas o explícitas, y es posible que también estén prediseñadas o sean emergentes, dependiendo del caso.

Que los individuos se organicen implica también que el *comportamiento* del grupo se vea reforzado. Y si además tienen la capacidad de modificar posteriormente su estructura

y comportamiento, como por ejemplo al agregar, eliminar o substituir elementos durante la ejecución del sistema, estamos ante la *adaptación dinámica* buscada.

La formación de estas estructuras se puede provocar explícitamente mediante el uso de dos tipos de mecanismos diferentes. Estos tienen su fundamento en la limitación de las actividades de los individuos y del rango de acciones válidas de los mismos: son los llamados *controles* y *protocolos*.

En cuanto a los primeros, los *controles*, pueden tanto *imponer* como *prohibir* interacciones específicas, lo que a menudo se materializa en la forma de conexiones arquitectónicas. Las estructuras auto-adaptativas, que según [10] serían típicamente centralizadas, muestran muchos ejemplos clásicos de este tipo: la mayoría de ellos manifiestan lazos explícitos de control, inspirados en los *reguladores* de la teoría de control clásica.

Por otro lado, los *protocolos*, que pueden *permitir* o canalizar comportamientos, se basan en el consenso y en los acuerdos. Pueden ser descritos, genéricamente, como una manera de controlar las estructuras descentralizadas (o incluso distribuidas) [97]. Es decir, con los protocolos cada agente sabe cómo interactuar con el resto, consiste en un conjunto de reglas que les facilita coordinarse, solo es necesario que se pongan de acuerdo para poder comunicarse. Al mismo tiempo estos protocolos *regulan* el desarrollo de la estructura de interacción, pueden "dirigir" el comportamiento del sistema, influir en los procesos de decisión de los agentes, etc.

Estos dos mecanismos definen un amplio espectro de regulaciones. Las organizaciones de agentes y sus arquitecturas son regulados por controles unitarios, atómicos, como las normas, límites, bloqueos, lazos de control o restricciones. Por ejemplo en una sociedad de agentes (formando un sistema "normativo"), una norma o convención puede considerarse una regla de comportamiento social, la que es aceptada de manera general o tácita con el objetivo de mejorar la eficiencia de esa sociedad. Por lo tanto las normas pueden implicar roles estructurales, con definiciones semánticas claras, donde ciertos principios pueden considerarse como abstracciones de normas, incluso pueden tener sanciones asociadas y además la capacidad de evolucionar y cambiar.

De igual manera, la regulación puede estar dada por los protocolos, que pueden ser múltiples, conectivos, como los concentradores, puentes, canales o espacios. Los conceptos de argumentación y negociación, muy utilizados en SMA, están firmemente ligados a los protocolos, que a su vez definen una estructura en un acuerdo. Los diferentes niveles de negociación se replican también en niveles de protocolos, como por ejemplo la negociación entre organizaciones implicaría un protocolo colectivo que llevaría a un acuerdo a gran escala, formal. Mientras que si se realiza entre individuos (agentes) tendríamos protocolos individuales, informales (por ejemplo un pacto) y el acuerdo obtenido estaría a un nivel más bajo que el anterior. Es posible llegar también a un acuerdo de nivel medio, como es el caso de un contrato entre un individuo y una organización.

El protocolo, en palabras de A. Galloway [97] puede considerarse un estilo de administración o gestión que inyecta "control en las fronteras del desorden". Asimismo lo ve como un sistema de administración distribuida que facilita las relaciones entre entidades autónomas, con virtudes como robustez, flexibilidad, inter-operabilidad, heterogeneidad, entre otras.

Los protocolos (y controles) que participan en la creación de las estructuras internas preliminares en nuestros patrones de adaptación funcionan de manera similar, guiando las interacciones de los agentes para componerlos, en un principio, en grupos informales y posteriormente en organizaciones.

En arquitectura de software se conocen muchas soluciones y patrones basados en controles implícitos y restricciones normativas. El enfoque que se propone en esta tesis establece una solución basada en el *consenso*: también se trata de un enfoque a nivel arquitectónico, ya que la descripción de las interacciones pertenece a este nivel.

El propósito de la utilización de estos mecanismos es conseguir (o *descubrir*) una estructura adecuada de los controles y protocolos que permitan que una estructura global pueda *emerger*, es decir, que se pueda obtener la definición de diversas formas de arquitectura. Estos elementos harán posible definir las principales estructuras internas a fin de lograr las *organizaciones basadas en acuerdos*.

Una vez que una estructura primaria pueda ser definida, un grupo elemental surge como

una organización preliminar. Esta será referenciada como una *iniciativa* y se describirá a continuación.

4.2.2. Definición de un Acuerdo Emergente: el concepto de *Iniciativa*

Como ya hemos señalado, existen mecanismos que pueden utilizarse para generar dinámicamente una organización preliminar dentro de un grupo de individuos. En esta tesis los individuos se corresponden con *agentes*, aunque en otro contexto también podrían ser componentes genéricos.

Cierta combinación controles y protocolos pueden generar la estructura, por lo que varios de ellos son considerados como *controles generativos* y *protocolos generativos*. Esta estructura es la que da lugar a un grupo que crece con la dinámica del entorno. Este unión preliminar emergente es lo que llamaremos una ***iniciativa***: aún no se encuentra plenamente establecida, pero sigue evolucionando.

La Real Academia Española [187], en su primera definición determina que *"iniciativo"* da principio a algo. De manera similar podemos decir que es el primer paso de un proyecto o del punto de partida de alguna acción. Estas definiciones son apropiadas para el enfoque de este trabajo, ya que en nuestro caso es una formación primaria de individuos, el inicio de un grupo que posteriormente puede fortalecer su unión.

Nuestra *iniciativa* puede seguir creciendo y mutando ya que su naturaleza es adaptativa, y lo hará hasta que consiga cierto tipo de estructura "estable", cuando ya es posible considerarla como una *organización*. Esta estructura se logra cuando todos los participantes llegan al *acuerdo* necesario para conseguir el objetivo principal que provocó su reunión. La organización resultante es conceptualmente similar a las organizaciones de varios enfoques de SMA (como se vio en 2.1.3), e incluso de la arquitectura THOMAS (3.4.1) analizada previamente y que da soporte a este trabajo.

Los párrafos anteriores implican tres conceptos de importancia para esta tesis [180]:

- El concepto de *iniciativa*. Es un grupo preliminar de individuos (agentes) que se ensamblan siguiendo una determinada estructura generada por un conjunto de controles

y protocolos, y también por ciertos patrones arquitectónicos asociativos;

- El concepto de *organización*. Se trata de un grupo ya establecido, dinámicamente originado a partir de una *iniciativa*. Una vez que ha sido creada es funcionalmente equivalente a las organizaciones estáticas que ya existan;

- El concepto del *acuerdo*. Es el acto por el cual una *iniciativa* se convierte en una organización "estable". De hecho, esto puede verse como el *consenso* que se alcanza entre los individuos dentro de la "semilla" inicial del grupo. Se establece así una relación entre partes que es dinámicamente establecida y gestionada. Dentro del acuerdo entran a jugar temas tan importantes como los roles, derechos y obligaciones de las partes, además de cuestiones no funcionales como el rendimiento, disponibilidad, etc.

El proceso que genera la *iniciativa* puede verse como si el sistema se moviera a un nuevo estado, en el que la estructura del *pasado* es suplantada por una estructura *nueva* emergente. Obviamente esta novel estructura admite nuevos elementos debido al entorno dinámico, pero ahora uno de sus objetivos es reforzar su naturaleza y tender a la persistencia. Como las estructuras pueden ser cada vez más complejas, queda claro que para ciertos tipos de problemas es necesario que los individuos se agrupen en protoorganizaciones, y después en organizaciones "estables" *basadas en acuerdos*.

Como aclaración, las siguientes frases serán tomadas como sinónimos de la *iniciativa*: protoorganización, preorganización, organización preliminar, grupo preliminar.

Tomemos como ejemplo una situación real de crisis y que es necesario que varios coches policiales acudan al lugar de los hechos. A priori, ninguno de los policías presentes es el líder de esa reunión. Siguiendo un protocolo interno (y de hecho la jerarquía policial es un protocolo), se elige a uno de ellos como líder: este acuerdo genera la organización preliminar. A esto lo podemos denominar *protocolo generativo*: cuando los individuos siguen este tipo de protocolo, se definen patrones estructurales implícitos.

Una *iniciativa* puede generarse a partir de estos patrones, a los que llamaremos *patrones de adaptación*, utilizando el término "patrones" en un sentido arquitectónico. Estos patro-

nes son prediseñados a partir de los servicios necesarios por una *iniciativa*, y concebidos para el refinamiento semántico correspondiente.

Los patrones representan una estructura "estática" (un fragmento) que tiende a una estructura dinámica, la *iniciativa*, que busca una forma "estable", la *organización*. Como se ha dicho en varias oportunidades, el sistema está concebido fundamentalmente como una arquitectura orientada a servicios, por lo que metodológicamente nuestra primera organización "estable" debe ser originada como proveedora de ciertos servicios de alto nivel. Asimismo, estos servicios pueden ser propuestos como los puntos de partida para la definición funcional de aquellas primeras organizaciones.

La descomposición funcional de estos servicios también podrán ser usados para diseñar la estructura jerárquica de las organizaciones. El concepto de proceso de servicios, en este contexto, intenta proveer una perspectiva semántica clara de la funcionalidad de los servicios, describiéndola como un flujo de tareas. Cada servicio (de alto nivel) se desdobla en un proceso (semántico), el que describe la coordinación entre los servicios de bajo nivel. Cada servicio es proporcionado por una organización de bajo nivel, proveyendo la descomposición estructural de la anterior organización de alto nivel. Este proceso dirige la definición (semántica) de cualquier organización orientada a servicios y es utilizado para definir las estructuras estáticas.

Por otra parte, incluso los acuerdos emergentes dinámicos deben ser consistentes con estas definiciones semánticas. Tanto es así que este proceso además proporciona un método para descubrir los patrones requeridos para las *iniciativas*. Algunos de estos patrones son *Façade, Mediator* y *Surveyor*, y aunque su denominación resulte familiar es importante recordar que se definen desde el punto de vista arquitectónico. Estos patrones y otros que componen la propuesta de *lenguaje de patrones* presentado en esta tesis serán descriptos con más detalle en el Capítulo 5.

4.3. Ciclo de Vida de las Estructuras Auto-organizadas

Se dijo previamente que un conjunto de individuos puede organizarse en base a metas concretas en determinadas estructuras utilizando controles y protocolos. Estos mecanismos

hacen posible definir las estructuras internas relevantes para obtener las *organizaciones basadas en acuerdos*. Una vez que la estructura primaria está definida, un grupo elemental emerge como entidad preliminar: la *iniciativa*. Esta tiene la capacidad de cambiar con la dinámica del entorno e incluso puede consolidarse como una organización plena.

Hasta llegar a formarse la estructura preliminar que llamamos *iniciativa* y luego la *organización* pertinente, los agentes participantes se verán involucrados en una serie de pasos. Dada la heterogeneidad presente en el entorno y según los servicios que brinden los agentes, sus capacidades de interacción y su propio conocimiento, deberán tomar decisiones, acatar ciertas normas, utilizar determinados protocolos, etc. A esta sucesión de pasos denominamos *ciclo de vida* de nuestras estructuras auto-organizadas y se verá con más detalle a continuación.

4.3.1. El Ciclo de Vida

La serie de pasos que se citaron anteriormente constituyen el *ciclo de vida* de nuestras estructuras auto-organizadas, y está representado en la Figura 4.1 [57].

Dicho ciclo comienza con un simple agente – (i) en la figura, con capacidades de interacción y con potencial para exportar algunos de sus servicios. Cuando inicialmente un agente ingresa en el sistema no forma parte de ninguna organización. Sin embargo se encuentra con un conjunto predefinido de *controles* y *protocolos* (ii), como ya se ha explicado previamente. De esta manera las interacciones del agente puede ser guiadas y asimismo puede mantener conversaciones estructuradas con otros agentes, componiendo grupos informales entre ellos (iii).

Ante la generación de un cambio externo, como por ejemplo modificarse el objetivo original de reunión, realizarse cambios en algunas prioridades, incrementarse el nivel de seguridad requerido, etc., el sistema debe reaccionar con un comportamiento adaptativo. Esta funcionalidad es la que debe generar el disparo o lanzamiento (*triggering*) de la formación de nuestras estructuras auto-organizadas. Para que sea posible conseguir la reacción deseada se provee de cierto número de *patrones de adaptación* (iv).

Estos patrones no son estrategias cerradas ni tampoco una descripción completa de

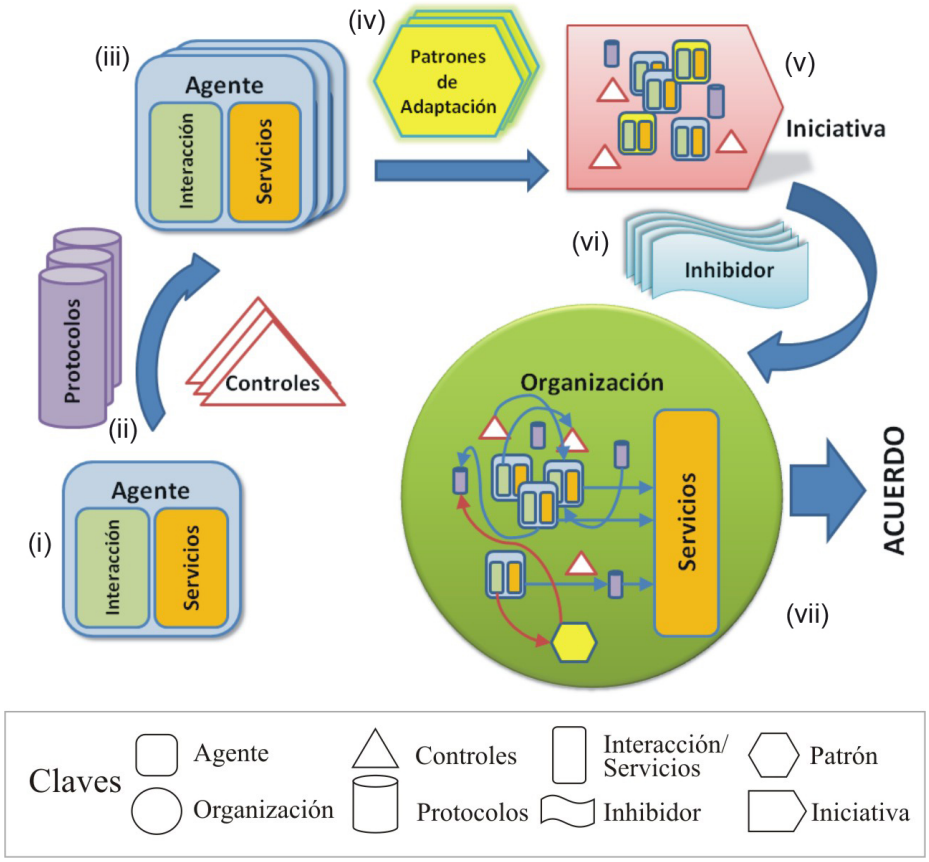

Figura 4.1: Ciclo de vida de una estructura auto-organizada (de [57])

comportamientos. Son más bien definiciones parciales de elementos y sus inter-relaciones, los que incluyen suficiente información para que un agente pueda llevar adelante un comportamiento detallado y relevante. Este comportamiento incluye la definición de algunos protocolos, como ya se ha mencionado. En estos patrones tienen activa participación los que denominamos "elementos del cambio", los agentes *shifter* y *changent*, que serán explicados posteriormente.

Bajo la influencia de un *patrón de adaptación* ciertos agentes dentro del grupo pueden adquirir funciones específicas, y comenzar a formar una estructura: esto es lo que llamamos una *iniciativa* (v), concepto clave para nuestro enfoque adaptativo.

Este proceso continúa, por lo que la *iniciativa* puede crecer y evolucionar. Viendo esto como una metáfora biológica, por ejemplo haciendo un paralelo con las *células madre* [164], estas evolucionan adquiriendo funciones concretas hasta que los inhibidores

existentes detienen este proceso para prevenir que su evolución acabe en células cancerosas [57]. De manera similar, cuando la *iniciativa* está completando las funciones requeridas –cada servicio está siendo provisto por algún agente– nuestros inhibidores (vi) detienen su crecimiento y la transforman en una *organización* completa (vii).

Estas organizaciones son adaptativas por formación y por lo tanto tienen la capacidad para evolucionar y participar en acuerdos mayores, los que podrían disparar la formación de otra organización aún más grande. En resumen, este proceso garantiza que el resultado sea en definitiva una *arquitectura adaptativa basada en acuerdos*.

4.3.2. Elementos del Cambio

El enfoque que estamos utilizando es orientado a servicios y centrado en organizaciones, además de usar agentes, como ya se ha dicho. Como características principales de nuestros agentes se destacan la colaboración y la cooperación. De hecho no son tenidos en cuenta los comportamientos interesados, egoístas o competitivos ya que para este enfoque se proponen escenarios que propician tanto la benevolencia de los agentes como el trabajo solidario, altruista y en equipos. Asimismo, en 3.4.1 se hace referencia a la arquitectura FIPA de los agentes participantes.

En el ciclo de vida de nuestras organizaciones basadas en acuerdos, explicado anteriormente, se utilizan los *patrones de adaptación*, en los cuales participan activamente dos tipos de entidades a las que llamamos *agentes shifter* y *agentes changent*.

Al comenzar el ciclo estas entidades son consideradas regulares o “neutras”, las que luego pueden evolucionar para llevar a cabo funciones específicas. Estas funciones pueden basarse tanto en los cambios del entorno como ser disparadas por los patrones de adaptación y por las interacciones entre los propios agentes. Los agentes software son entidades computacionales que cuentan con ciertas características diferenciadoras, por ejemplo la autonomía, la proactividad y la flexibilidad, como se ha visto en el Capítulo 2. Si además cuentan con el aprendizaje como otro atributo relevante son más que adecuados para erigirse en nuestros elementos del cambio.

Por ello seguidamente intentaremos clarificar las diferencias entre los agentes que intervienen en los cambios.

Agentes *Shifter*

Un agente *Shifter* [57] es un agente software que tiene la capacidad de adaptar su comportamiento a los cambios del entorno. En un principio puede ser un agente estándar pero en base a los requerimientos, tanto del sistema como del entorno, tiene la facultad de generar un cambio en su comportamiento cuando es inducido a ello. Puede verse como si "cambiara su forma", es decir, que puede modificar su capacidad para interactuar con otros, desempeñar otras funcionalidades una vez que se le haya instruido para ello.

Como ejemplo, estos agentes podrían hacer uso de ciertas funciones especiales que forman parte del *middleware* y que no están habilitadas para agentes regulares. Como alternativa puede darse el caso que uno de ellos puede trabajar en paralelo con otro agente y que este último cuente con funcionalidades diferentes y se "fusiona" con el primero. Al trabajar en conjunto se los puede considerar a ambos como una única entidad, un único agente que ha cambiado. El elemento existente se une a un nuevo agente (o proceso) que añade la nueva funcionalidad buscada.

Agentes *Changent*

Por otro lado tenemos agentes *Changent* [181]. Estos son los agentes "del cambio". Su principal característica, más allá de sus propias funciones, es que se encarga de inducir el cambio en el resto de los elementos (agentes) del patrón. Básicamente entendemos como cambio una modificación de estructura, un cambio en la interfaz de comunicación, en la capacidad de comportamiento, y no un mero cambio de estado en los agentes.

Por ejemplo tienen la capacidad de habilitar a los agentes *shifters* para hacer uso de ciertas funciones específicas del *middleware* que no están permitidas a todos los agentes. Si lo vemos a bajo nivel podría cambiar el estado de una variable, mientras que en un nivel más alto podría ser el iniciador del despliegue de un patrón completo (o varios de ellos), según la dinámica presente. También tienen la capacidad para utilizar mecanismos que pueden incentivar o sancionar el uso de ciertas funcionalidades por parte de otros agentes

[45], de tal manera que impliquen un mejor comportamiento del sistema y no comprometan el normal desarrollo del mismo.

Estas alternativas para generar cambios en los elementos permiten conseguir un interesante dinamismo en las arquitecturas. Tienen el potencial necesario para que se puedan crear sistemas que se adaptan y así lograr que emerjan los comportamientos y propiedades deseados.

Ejemplos de estos elementos del cambio, los agentes *shifter* y *changent*, se ponen de manifiesto cuando se explican las especificaciones realizadas en Cálculo-π [207] de cada *patrón de adaptación* en el Capítulo 5.

4.3.3. Patrones de Adaptación

La utilización de *patrones de diseño* por parte de arquitectos y desarrolladores de sistemas software es una práctica extendida, como se hizo referencia en el Capítulo 2. También queda claro que ayudan a resolver cuestiones claves en el diseño de software como la escalabilidad, eficiencia, flexibilidad, extensibilidad, entre otras. Los patrones que proponemos en este trabajo son utilizados desde el punto de vista de la arquitectura de sistemas, en un nivel de abstracción más alto que los usuales patrones orientados a objetos [98] utilizados por programadores y diseñadores de sistemas.

Como un dato histórico es interesante recordar que el término *patrón arquitectónico* (de software) fue utilizado por primera vez en el renombrado libro de Buschmann et al. [38] en 1996. Previamente, en 1995, Gamma et al. [98] había hecho lo propio con un catálogo de patrones para desarrollo de sistemas orientados a objetos.

También es importante recordar que el concepto de *patrón* se debe sin lugar a dudas al arquitecto (de edificios) Christopher Alexander. Este profesional escribió en los años 70 varios libros con detalles de enfoques arquitectónicos para resolver problemas comunes de diseño en los edificios. Entre sus declaraciones encontramos lo que representa un patrón: “una regla de tres partes, la que expresa una relación entre cierto contexto, un problema y una solución” [8].

Como parte de su producción ha documentado una colección de patrones y los ha or-

ganizado en una serie predefinida a la que llamó "secuencia". Esto acabó resultando en un *lenguaje de patrones arquitectónicos* que aún en estos días sirve de inspiración para numerosas comunidades, y entre ellas la de los profesionales de sistemas software.

En el trabajo de esta tesis se propone que la *iniciativa* es la construcción arquitectónica básica de nuestro enfoque y sus detalles se han explicado anteriormente. Esta construcción, primaria y evolutiva, puede generarse, por ejemplo, cuando los individuos se reunen y siguen un protocolo. A este último llamamos *protocolo generativo* y propician la creación patrones estructurales implícitos entre los entes reunidos. Como la *iniciativa* puede generarse a partir de estos patrones, y además como su evolución continúa, estos patrones constituyen nuestros *patrones de adaptación*. Como también se ha explicado, estos patrones tienen un nivel alto de abstracción y son utilizados en un sentido arquitectónico.

Nuestros *patrones de adaptación* son prediseñados a partir de los servicios necesarios por una *iniciativa*, en base al objetivo final del sistema y concebidos para el refinamiento semántico posterior correspondiente. En el Capítulo 5 se describirán con detalle los dieciocho patrones de adaptación propuestos en esta tesis y que forman nuestro *lenguaje de patrones de adaptación*.

Como ejemplo de descripción y funcionamiento veamos un resumen de un par de ellos:

- el patrón *Façade* es de utilidad cuando se necesita interactuar con una organización primaria que aún no tenga una estructura definida, es así que un agente ha de representar a la propia *iniciativa* en su papel de *fachada*, que redirige cualquier comunicación recibida;

- el patrón *Surveyor* es aplicable cuando es necesario controlar el crecimiento de la *iniciativa* mientras está emergiendo. Un agente cumplirá esa función y además tendrá las capacidades, entre otras, de agregar o eliminar miembros del grupo que se está formando, generar una lista de las funciones de cada miembro o informar quienes son los integrantes de la organización preliminar.

Sirvan estos dos patrones como muestra de la identificación que se ha realizado para este trabajo y que será ampliada en el Capítulo 5. Con la breve descripción anterior se procura dejar claro el carácter arquitectónico de los patrones, ya que representan una visión de alto nivel.

Está claro que un patrón proporciona una solución a un problema recurrente y que puede reutilizarse, pero es importante destacar que no describe un diseño real, y que por su solo uso se tenga garantizado que el problema de diseño podrá resolverse como es requerido. Esto mismo debe tenerse en cuenta con la utilización de nuestros *patrones de adaptación*, ya que estos representan un modelo abstracto que habrá de aplicarse en base a restricciones, competencias y habilidades de las entidades participantes, requerimientos originales y cambios posteriores, roles actuales y nuevos, etc.

4.3.4. Organizaciones Auto-organizadas

Los objetivos describen el propósito de la organización a un nivel alto. Esto permite que puedan determinarse las tareas, los tipos de agentes involucrados, la asignación de los recursos entre los miembros, etc. También deberán tenerse en cuenta otros requerimientos, como por ejemplo el tiempo de respuesta, rendimiento, seguridad, etc. Como ya se ha explicado, la *iniciativa*, luego del proceso que hemos llamado *ciclo de vida*, da lugar a una organización "plena" basada en acuerdos. La estructura viene dada por los roles y por las interacciones y comunicaciones de los agentes, de igual manera que los controles y los protocolos utilizados permiten conseguir el comportamiento global esperado.

Las organizaciones tienen estructuras que pueden identificarse como patrones y que a largo plazo se corresponden con los tipos de organizaciones conocidos, basados en los modelos humanos. Como ejemplos, por un lado tendríamos el modelo puramente jerárquico, con un elemento que supervisa, controla, toma las decisiones y coordina las tareas y las comunicaciones con un grupo de subordinados; por otro lado, un modelo plano, "anárquico" organizacionalmente, sin estructura fija, los miembros no se controlan entre ellos aunque se conocen y pueden obtener información unos de otros [12]. Entre estos dos ejemplos existe una gran variedad de tipos de organizaciones.

El modelo de interés para este trabajo es el *equipo* [79], en el cual todos los agentes desempeñando sus roles cooperan con el fin de cumplir el objetivo global, la información es compartida, la coordinación puede no ser centralizada y las decisiones se toman en ba-

se a consenso o *acuerdos*. La recursividad organizacional, que también es de interés en nuestro enfoque, permite además que el equipo pueda formar parte de otros y así abordar problemas de mayor envergadura. Es decir que participando en *acuerdos* mayores puede integrar organizaciones mayores. De esta manera las organizaciones conseguidas tendrán una estructura flexible y con capacidades de modificarse, buscando la *auto-organización* que permita que la arquitectura del sistema pueda finalmente adaptarse ante la dinámica del entorno. Para este fin proponemos nuestros *patrones de adaptacion* detallados en el próximo Capítulo.

Algunos trabajos [206, 209, 109] plantean que la auto-organización tiene características *bottom-up*, es decir que la organización y su estructura emergen desde la interacción y coordinación de sus integrantes, de una manera descentralizada; a diferencia de la adaptación que se plantea con características *top-dowm*, de comportamiento y de una manera centralizada. Sin embargo, según [58], auto-organización y adaptación, en cierta forma son complementarios y se encuentran entrelazados. En nuestro enfoque las organizaciones auto-organizadas han llevado a cabo algún tipo de adaptación según la dinámica del entorno.

Se destaca la importancia de la auto-organización debido a la construcción de estructuras que contienen tanto agentes como otras organizaciones. Al sufrir cambios, las organizaciones existentes tienen que adaptarse al nuevo comportamiento. Primero deben notar que ha ocurrido un cambio, esto es que un cambio puede emerger de una manera intrínseca [186], y luego tienen que adaptarse.

Más adelante, en el Capítulo 6, se plantea un caso de estudio para una prueba de concepto para describir el uso de los patrones de adaptación propuestos en esta tesis. Se trata de un ejemplo hipotético pero basado en situaciones reales y en un escenario del dominio de las emergencias médicas.

Básicamente, una vez que la *iniciativa* ha dado lugar a una organización ya establecida y está en pleno funcionamiento se han identificado cuatro alternativas de adaptación que impulsan la auto-organización.

- Caso 1: sin modificar el objetivo principal de la organización: se realiza una búsque-

da dentro de la propia organización, intentando localizar un servicio similar al que ya no está disponible. La idea principal es un reemplazo directo del servicio. Por ejemplo, un enfermero de la ambulancia debe retirarse y es reemplazado por otro enfermero que también forma parte del equipo.

- Caso 2: sin modificar el objetivo principal de la organización: la búsqueda interna solo encuentra un servicio con un mínimo de similaridad con el que ya no está disponible. En este caso, el responsable del servicio encontrado deberá "aprender" a responder de la manera que lo hacía el anterior. Un proceso de aprendizaje es factible ya que estamos en un entorno orientado a SMA. El tiempo en esta tarea debe ser razonable, de acuerdo a las caracteristicas del escenario. Por ejemplo, el médico traumatólogo de la ambulancia debe retirarse, no hay otro con sus características pero sí un médico generalista, este último tendrá que "aprender" a responder como el traumatólogo ya que tiene un conocimiento similar.

- Caso 3: sin modificar el objetivo principal de la organización: si la búsqueda interna falla, la organización está habilitada a realizar una búsqueda externa. Este caso puede ser considerado como un cambio de estado de la organización. Vuelve al nivel de una *iniciativa*, la que es mantenida hasta que adquiera un estado más seguro en base a los acuerdos correspondientes. Por ejemplo, un médico especialista en quemaduras de la ambulancia debe retirarse porque ha resultado herido, como es el único médico de la ambulancia habrá que solicitar la presencia urgente de otro para completar el equipo, la organización en este momento está "incompleta" hasta que llegue el agente faltante, aunque puede seguir ofreciendo servicio mínimos.

- Caso 4: con cambio en el objetivo principal de la organización: en este caso la organización es "forzada" a modificar su objetivo, o dividirlo en metas parciales. No es posible ofrecer el servicio original. Por ejemplo, cuando la ambulancia que está trabajando en la zona de crisis es requerida en otra urgencia (talvez por su propias características o especialidad). En este caso cambia su objetivo principal ya que ahora tiene que atender en otra zona y a otros pacientes; si fuera posible, parte de su equipo puede quedar trabajando en la primera zona pero formando parte de otros equipos. Puede verse como una "división" en metas parciales, o también que algunos de sus agentes se unen a otros grupos/organizaciones ya establecidas y que están trabajando a la par.

Estos cuatro casos también representan parte de la propuesta de este trabajo, son estudiados para una real adaptación de las organizaciones debido a que no solo modifican la estructura sino también el tipo de elementos que lo constituyen. Con la aplicación de los patrones de adaptación se intenta no solo obtener organizaciones basadas en acuerdos a partir de la *iniciativa*, sino también su auto-organización y de esta manera conseguir que se adapten ante perturbaciones en el entorno.

4.4. Operaciones para lograr Adaptación

Para lograr la adaptación buscada la propuesta es obtener organizaciones basadas en acuerdos que generen un comportamiento emergente. La adaptación de las organizaciones es tenida en cuenta desde los inicios, como lo demuestra la utilización del concepto de *iniciativa*. Hemos visto que la adaptación busca mejorar, entre otras cuestiones, la eficiencia de la organización y para ello son necesarias una serie de tareas en distintos niveles.

En una revisión global de todas las funciones que consideramos necesarias para definir mecanismos adaptativos se han encontrado las que se detallan más adelante en esta sección. El listado que se presenta no intenta ser exhaustivo y además varias de ellas se encuentran ya implícitas en los mecanismos de los agentes, por ejemplo el descubrir los servicios. En nuestra propuesta no se intenta implementar todas las enumeradas sino que simplemente señalamos que un "mecanismo general" debería considerar a todas ellas. Intentamos que nuestro propio esquema proporcione las más complejas de las citadas.

Nuestra propuesta está orientada a servicios, basada en agentes y centrada en organizaciones, por lo que consideramos que las operaciones involucradas en los cambios tienen que estar representadas en esos tres niveles.

Partimos de que un servicio reúne un conjunto de operaciones, se describe con una interfaz estándar y comprende una secuencia de actividades. Asimismo está detallado por un perfil y un modelo explícito de procesos, los que a su vez pueden ser divididos en varios procesos menores.

Analizando en primer lugar los cambios que se producirían a nivel de los *servicios*,

consideramos relevantes las siguientes tareas:

- Crear un servicio
- Eliminar un servicio
- Añadir operaciones al servicio
- Retirar operaciones del servicio
- Especializar un servicio
- Refinar un servicio
- Descubrir un servicio
- Perder un servicio
- Continuidad en el ofrecimiento del servicio, aun con cambios en el entorno
- Optimizar el servicio, añadiéndole determinada calidad
- ...

No resulta difícil concebir que un agente software presente sus capacidades operacionales o funcionalidades como servicios. De manera similar al listado correspondiente a los servicios, las operaciones relacionadas a los cambios que podrían producirse a nivel de *agentes* pueden resumirse en la siguiente lista:

- Añadir un servicio al agente
- Retirar un servicio del agente
- Añadir un rol al agente
- Retirar un rol del agente
- Añadir canal de comunicación
- Eliminar canal de comunicación

- Agregado del agente
- Eliminacion del agente
- Pérdida del agente
- ...

El concepto de organización que estamos utilizando tiene un carácter recursivo que puede ser analizado desde dos puntos de vista: externamente una organización puede considerarse un contexto, un dominio de influencia o un "ámbito" en el que se le aplican un conjunto de normas y reglas a los individuos de la organización. Por otro lado, internamente una organización también puede ser considerada como un colección o la reunión de cierto grupo de individuos, que cumplirán con las normas establecidas y además desempeñarán los roles que les correspondan.

Las *organizaciones* también sufrirán cambios en base a la dinámica del entorno y las operaciones relacionadas a su nivel serían:

- Crear una organización
- Destruir una organización
- Cambia el objetivo de la organización
- Crear una relación entre dos organizaciones
- Eliminar una relación entre dos organizaciones
- Incorporar un agente a una organización
- Quitar un agente de una organización
- Mover un agente de una organización a otra
- Dividir una organización
- Dos organizaciones se unen en una mayor

- Cambia la topología de la organización
- ...

Las operaciones listadas pueden estar representadas en los patrones, utilizando uno o más de ellos durante el proceso de adaptación. Por el contrario, otras operaciones estarán reservadas para el nivel de infraestructura o también del *middleware*, como se detalló al explicar la arquitectura que da soporte a este trabajo (ver 3.4.1). Como ejemplo de uso de patrones para realizar operaciones tomemos al patrón *Surveyor* que con sus métodos *m1: add_member* y *m4: remove_member* permite agregar y quitar un agente de una organización. Se podrán ver este y otros casos similares en las descripciones de los *patrones de adaptación* en el Capítulo 5.

4.5. Esquema General de la Propuesta de Patrones

En este Capítulo estamos presentando nuestra propuesta y en las secciones anteriores hemos tratado los distintos temas que la componen. Ya hemos visto cómo obtener el comportamiento emergente del sistema por medio de acuerdos entre los integrantes del grupo. También cómo por medio de determinados controles y protocolos es posible favorecer la creación de una *iniciativa*, la que siguiendo un ciclo de vida puede consolidarse como una *organización completa basada en acuerdos*. Tanto los elementos del cambio, los agentes *changent* y *shifter*, como los patrones de adaptación y las operaciones involucradas facilitan que emerja el comportamiento buscado.

Antes de presentar el esquema general de nuestra propuesta, es importante conocer cuál ha sido el punto de partida. Los patrones que se proponen en este trabajo tienen como inspiración distintas fuentes. Por ejemplo, es sabida la existencia de diversos conjuntos de patrones conocidos, como ya se ha comentado con anterioridad [98, 38]. Normalmente se los utilizan como solución de diseño, pero queda claro también que pueden ser de mucha utilidad para especificar y listar comportamientos.

Hemos estudiado y analizado numerosos trabajos, y varios de ellos han tenido un interesante avance en la utilización de patrones para expresar esquemas de adaptación de sistemas, como por ejemplo [46, 191, 240], aunque por supuesto no han sido los únicos. Las soluciones que proponen resultan de gran atractivo, por lo que los hemos escogidos también como punto de partida para nuestra investigación sobre los patrones de adaptación. Al estudiar sus propuestas hemos notado que una de las cuestiones importantes es que agrupando los patrones para su uso se obtienen resultados más efectivos. De hecho su mayor utilidad viene asociada a los lenguajes (o sistemas) de patrones que normalmente pueden obtenerse con ellos.

Otra fuente interesante es el trabajo que se propone en [231, 226] y cuya solución es un lenguaje de cinco patrones para la adaptación de SMA. Aunque podemos considerar su enfoque algo limitado por su dominio, con sus patrones se plantea un ciclo de vida interesante.

A partir de estos análisis quedaba claro que nuestra propuesta de patrones debía formar un conjunto, generar un lenguaje de patrones y que este expresase un ciclo de vida del sistema. La idea de este ciclo de vida es el concepto que da el punto de partida de nuestro trabajo de investigación.

Teniendo como referencia los patrones ya conocidos, los trabajos anteriormente citados y varios más que han sido estudiados, la idea de utilizar patrones para obtener el comportamiento adaptativo estaba muy clara. Si nos atenemos a las clasificaciones más utilizadas, a los patrones de modelos conocidos o de tecnologías específicas, los patrones que serían de utilidad en nuestro trabajo pueden comprender a los patrones de diseño, arquitectónicos, de comportamiento, de integración y de mensajería.

En este punto ya estaba definido que se utilizarían los conceptos de *patrón* y un ciclo de vida expresado con un *lenguaje de patrones*.

El encuadre de este trabajo dentro del proyecto de las Tecnologías del Acuerdo (AT) nos permite plantear un entorno de SMA, aunque sus modelos son básicamente no adaptativos. En estas condiciones el objetivo es estudiar cómo un conjunto de agentes independientes pueden incorporarse al sistema y adaptarse a estructuras de organizaciones definidas en AT. Es decir, que el conjunto de agentes deriven hacia un esquema similar al de una organiza-

ción en AT y su posterior integración.

Como soporte a lo expresado hasta ahora, el despliegue de la propuesta concreta facilitará su comprensión. El esquema general es el siguiente:

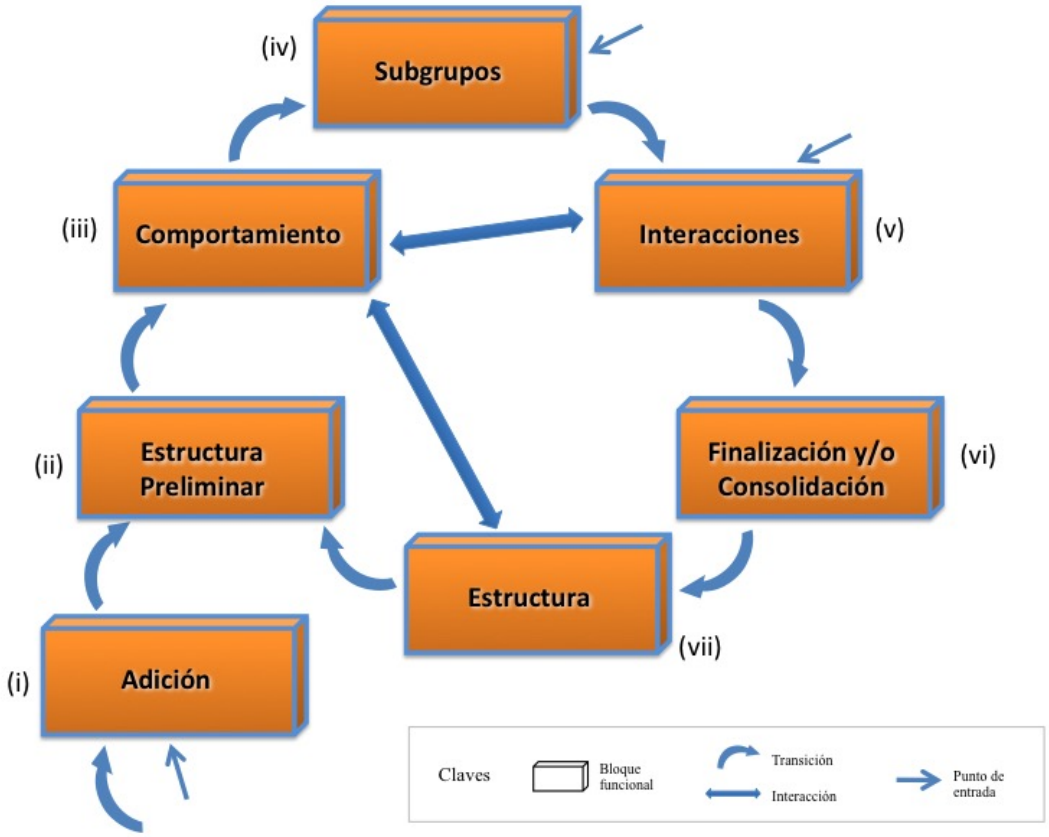

Figura 4.2: Esquema general de nuestra propuesta de patrones

En la Figura 4.2 se aprecia la idea general en forma de diagrama en el que están representadas las distintas etapas que describen el comportamiento global del sistema. Las fases que forman este ciclo se corresponden con determinados "bloques" de comportamiento y que en conjunto forman lo que posteriormente, en el Capítulo 5, veremos como nuestro *lenguaje de patrones*.

Puede observarse que presenta una serie de grandes grupos con varios puntos de entrada y una secuencia, además de ciertas interacciones entre dichos grupos.

Los bloques representan distintas funcionalidades, como la *adición* (i) de nuevos ele-

mentos al sistema, identificándose aquí uno de los puntos de entrada al ciclo. A continuación podrá obtenerse una *estructura preliminar* (ii), que también incluiría la existencia de un líder de grupo. Otros bloques importantes se corresponden con las funcionalidades del *comportamiento* (iii) tanto general como externo de la agrupación, y también de los elementos independientes que forman *subgrupos* (iv), en este último además se identifica otro punto de entrada al ciclo. Las *interacciones* (v) también están representadas como funcionalidades en bloque, siendo también un punto de entrada. De igual manera, la *finalización y/o consolidación* (vi) pueden dar paso al bloque *estructura* (vii), la que interactúa además con la ya citada *estructura preliminar* y con el bloque de *comportamiento*. Este último puede interactuar además con el bloque de *interacción*.

Cada uno de estos bloques realiza su función mediante un *patrón o conjunto de patrones* que globalmente proporcionan su comportamiento específico. Es decir que el esquema de este ciclo puede representar a toda una organización que se comporta como un sistema que tiene todos estos bloques ejecutándose funcionalmente.

Está claro también que por cada uno de los patrones propuestos se podrían incorporar otros con pequeños matices, pero consideramos que los desarrollados hasta ahora son los esenciales para hacer funcionar el sistema. Estos son los importantes para definir la estructura de nuestro ciclo de vida. A partir de aquí para el agregado de otros solo es cuestión de estudiar el comportamiento buscado y adecuar el conjunto de controles y protocolos para su realización.

En el capítulo siguiente, aparte del mecanismo de disparo de los patrones en 5.2.1, se describirán cada uno de los *patrones de adaptación* propuestos con sus respectivas características y se podrá ver cómo todos estos se ensamblan en un ciclo de vida definido. Este ciclo se explica con más detalle en la Subsección 5.7 y para mayor aclaración está representado en la Figura 5.36.

4.6. Conclusiones

A lo largo de este Capítulo se han descrito elementos y cuestiones relevantes que forman parte de nuestra propuesta. Si partimos de la base que para conseguir cierta adapta-

ción en los sistemas complejos su propia estructura debería presentar alguna flexibilidad, el enfoque que consideramos adecuado es el de proveer una construcción arquitectónica al problema de coordinación. La generalidad, abstracción y escalabilidad son características de este enfoque.

Al utilizarse agentes de software, servicios y organizaciones como partes fundamentales de la propuesta, bien puede plantearse la posibilidad de lograr comportamientos adaptativos de la arquitectura. El hecho de que el *acuerdo* entre entidades computacionales sea un concepto básico para la búsqueda de las estructuras adaptativas permite hacer un paralelo entre un *acuerdo global* entre integrantes de un grupo (de agentes) y el *comportamiento global emergente* del sistema (SMA).

Los mecanismos que pueden favorecer la coordinación emergente y que posteriormente permiten definir organizaciones, también emergentes y basadas en acuerdos, son los conocidos *controles* y *protocolos* que han sido descritos anteriormente. Como definen un amplio conjunto de regulaciones son tenidos en cuenta para conseguir que grupos de individuos puedan ser organizados en entructuras más o menos flexibles. La meta es conseguir que emerja una estructura adecuada que permita obtener la definición de diversas formas de arquitecturas. Estos elementos harán posible definir las principales estructuras internas a fin de lograr las *organizaciones basadas en acuerdos*.

Esta combinación de controles y protocolos es responsable de la generación de una estructura que da lugar a un grupo de individuos que crece con la dinámica del entorno. A esta unidad preliminar emergente llamamos *iniciativa*, aún no se halla plenamente establecida y continúa evolucionando. Eventualmente sus integrantes llegarán a un *acuerdo* y este grupo preliminar podrá consolidarse como una *organización*.

En el proceso, tanto de origen de la *iniciativa* como de posible consolidación en una *organización basada en acuerdos*, tienen un papel importante la presencia de ciertos *patrones* estructurales implícitos. Estos *patrones* participan de manera activa en la generación de los acuerdos emergentes que facilitan la creación de las estructuras adaptativas buscadas. Esta participación queda de manifiesto en el *ciclo de vida* de las estructuras que se descri-

bió previamente. Los patrones representan un fragmento de estructura "estática" que tiende a una dinámica, la *iniciativa*, la que puede transformarse finalmente en una *organización*.

Estos patrones, que llamamos *patrones de adaptación*, son definidos desde el punto de vista arquitectónico y prediseñados a partir de los servicios necesarios por una *iniciativa*, en base al objetivo del sistema y concebidos para un refinamiento posterior. En conjunto generan un *lenguaje de patrones*, el que podrá ser analizado con más detalle en el Capítulo siguiente. En estos patrones también participan los llamados agentes *changent* y *shifter*, que con sus respectivas características permiten que sean posibles las adaptaciones de la arquitectura.

El comportamiento global del sistema va desde la adición de nuevos elementos (agentes), hasta la obtención de una organización "estable", pasando por determinados estadios como la formación de una estructura preliminar y la aparición de interacciones y comportamientos importantes. En cada una de estas etapas los patrones de adaptación proporcionan la funcionalidad necesaria para lograr el comportamiento emergente de la arquitectura.

Como en este Capítulo se ha descrito la propuesta de trabajo, en el siguiente se podrán ver en detalle los patrones de adaptación, su dinámica de funcionamiento y como todos se ensamblan en un lenguaje de patrones.

Capítulo 5

Patrones de Adaptación

5.1. Introduccion

Una *iniciativa* es un grupo preliminar de agentes reunidos en base a un objetivo global, más allá de los objetivos (o metas) individuales de cada agente. En su definición, como hemos visto en 4.2.2, entran en juego una serie de *controles* y *protocolos*. Conjuntamente con estos, la *iniciativa* puede ser generada a partir de determinados patrones, a los que hemos llamado *patrones de adaptación*.

En Capítulos anteriores se han presentado conceptos relacionados con los patrones, y específicamente de patrones de software. En este Capítulo se harán breves referencias a los conceptos teóricos necesarios como introducción a los *patrones de adaptación*, que son los protagonistas importantes de este trabajo.

Como la arquitectura que da soporte a nuestra propuesta también está definida como *orientada a servicios* (ver 3.2), metodológicamente los primeros grupos, aunque todavía no presentan "estabilidad", deben ser concebidos como proveedores de ciertos servicios de alto nivel. Estos servicios también serán usados como punto de partida, conjuntamente con los patrones de software, para la definición de las organizaciones evolutivas que inclusive en su etapa inicial tratan de proporcionar esas funcionalidades.

Hasta el momento de escritura de esta tesis han sido identificados dieciocho patrones de adaptación. Si bien algunos nombres son típicos de los patrones para la programación

orientada a objetos [98], los que proponemos se definen en un contexto diferente, e incluso la descripción de las interacciones entre los elementos es pertinente a este nivel de abstracción. Son, en definitiva, patrones *arquitectónicos*: no describen una solución real sino que proporcionan un esquema genérico para la solución, describen sus elementos, responsabilidades, relaciones y la manera en que colaboran [38]. Nuestros *patrones de adaptación* se mencionan en la tabla 5.3.

En el apartado siguiente se presenta la plantilla (o *template*, en inglés) utilizada para uniformar y estructurar la información brindada por los patrones. Posteriormente se describen cada uno de estos patrones y en las secciones siguientes son analizados la clasificación, el catálogo y el funcionamiento de los mismos.

5.2. La Plantilla (o *Template*) para los Patrones de Adaptación

Un patrón tiene elementos básicos: su *nombre*, describe un *problema*, describe una *solución* y por último describe también las *consecuencias* de su uso. Estos elementos se organizan y presentan en una plantilla (*template*) que se utiliza para realizar la descripción de los patrones. El diseño de la plantilla debe representar de manera efectiva la información relacionada con los patrones, debe ser clara y lo menos compleja posible. De esta manera podrán compartirse más eficazmente las posibles soluciones de forma estructurada a problemas en contextos similares. Asimismo, ayuda a evitar errores, favorece las comparaciones y facilita la transmisión de conocimiento en el desarrollo de sistemas.

Los distintos enfoques que utilizan patrones usualmente suelen presentar sus propias plantillas, esquemas o estructuras con la información que mejor se adapta a sus propuestas. Por ejemplo, en los trabajos [38, 98, 165, 145, 190, 90] encontramos que cada uno tiene su plantilla particular, en ellas ciertos campos son comunes y otros son específicos para cada propuesta.

Para la descripción de los *patrones de adaptación* que proponemos en esta tesis también utilizamos una plantilla, con el objetivo de uniformar y dar estructura a la información,

además de facilitar la comprensión de su aplicación.

La plantilla utilizada en este trabajo se detalla a continuación y es similar en estilo a la de [98]. Cuenta con un formato consistente, incluso luego de haberla modificado porque se han prescindido, simplificado o cambiado algunos campos para denotar la dinámica presente en los sistemas adaptativos.

Plantilla de patrones

- Nombre del Patrón: El nombre identifica unívocamente al patrón e intenta describir brevemente su función en la arquitectura del sistema. La elección del nombre es importante porque debe transmitir su esencia y también debido a la cantidad de patrones que existen.
- Clasificación: La clasificación y categorización del patrón facilitan su organización teniendo en cuenta su propósito. Realizar una correcta clasificación es importante ya que indica su aplicabilidad en términos de abstracción y de tareas de desarrollo.
- Intención: La intención del patrón define brevemente el problema recurrente para el cual se define la solución. La intención, o problema, es una parte fundamental del patrón porque es la razón por la que fue creado.
- Contexto: Describe la situación, condiciones y contexto apropiados para la utilización del patrón. Puede describir una configuración particular del problema y proveer las bases para definir los factores de aplicabilidad del patrón.
- Motivación: Una descripción del escenario, las metas y objetivos del sistema que motivan la aplicación del patrón. Esto también ayudará a entender las descripciones siguientes.
- Esquema: Un esquema gráfico para representar a los agentes que intervienen en el patrón, así como también las organizaciones e interacciones de los elementos intervinientes.
- Participantes: Describe los elementos que participan en el patrón, sus colaboraciones y responsabilidades, los que forman parte de la solución propuesta.

- Comportamiento: Provee una descripción del comportamiento del patrón, un escenario que puede estar representado con un ejemplo de un diagrama de secuencia.
- Consecuencias: Describe los efectos de la aplicación del patrón, cómo se consiguen sus objetivos y sus resultados. También pueden describirse las ventajas y desventajas de la utilización del patrón.
- Restricciones: Describe las restricciones que deben ser satisfechas, que son relevantes para el problema particular y para utilizar el citado patrón en ese contexto. Es decir, que pueden determinar la aplicabilidad o no del patrón.
- Patrones relacionados: Describe otros patrones que podrían utilizarse en conjunto. Por ejemplo, pueden ser patrones que inducen el uso del actual o que son utilizados posteriormente, o también patrones que sean una variante o alternativa al mismo.

En secciones posteriores de este Capítulo esta plantilla será ampliamente utilizada para la descripción detallada de cada uno de los *patrones de adaptación* que proponemos en este trabajo de tesis.

La plantilla debe complementarse con algún modelo de interacción – en los patrones de adaptación los protocolos son de particular importancia: esta es la razón por la que se usará una especificación en *Cálculo-*π [207]. Se utiliza este cálculo debido a su expresividad pero no es necesario para la codificación de los patrones, de hecho, se podrían utilizar otros mecanismos (o incluso formalismos) a estos efectos. Esta especificación esencialmente utiliza el *Cálculo-*π poliádico estándar, con algunas y convenientes alteraciones. Se utiliza la notación CSP (siglas en inglés de *Communicating Sequential Processes*[1]) en lugar de la original. Para una rápida introducción al Cálculo-π puede consultarse [234].

5.2.1. Mecanismo de disparo de los patrones de adaptación

Todos los patrones de adaptación, cuya plantilla acabamos de describir, tienen asociados ciertas condiciones. Estas condiciones no han sido especificadas en el contexto de la

[1]En ciencias de la computación CSP es un lenguaje formal para describir patrones de interacción en sistemas concurrentes.

plantilla pero sí serán detallados en cada patrón cuando posteriormente se los describa. En 4.5 ya hemos visto el esquema general del funcionamiento, por lo que con los párrafos siguientes podrán aclararse las situaciones en que los patrones deben desplegarse y también cuándo detenerse.

La precondición que estamos citando es lo que llamamos *condición de disparo (triggering)*, y lo que significa es que cuando en el sistema se produzca la situación descrita por esa condición el patrón se *activa*. En ese caso puede existir un conjunto de agentes que se vean afectados, que ocupan los roles definidos en el patrón y despliegan el protocolo que corresponde a dicho patrón. Estos protocolos pueden ser continuos definiendo un comportamiento continuado, o también ser breves en el sentido que cuando el comportamiento correspondiente se acaba el protocolo vuelve a ejecutarse.

Básicamente, el disparo de los patrones se produce en función de una condición positiva, cuando esta se detecta el patrón se despliega según lo previsto. Mientras se está ejecutando ese patrón no vuelve a dispararse con los mismos elementos, sin embargo cuando el comportamiento definido ha llegado a su fin, si la condición prevalece el sistema dispara nuevamente el patrón.

Para el mecanismo de disparo que estamos considerando, en principio, una implementación probable sería realizarlo como un espacio de tuplas, aunque sería funcionalmente similar utilizar algún tipo de almacenamiento de datos común al que habría que tener acceso, como por ejemplo un espacio compartido o un espacio en el *middleware* del sistema. Una manera de desacoplar fácilmente la dinámica de los patrones del resto del sistema es utilizar ese espacio de tuplas (como una *pizarra*) para incluir la presencia de una condición como una tupla. Cuando el sistema detecta una condición se genera la tupla correspondiente y se almacena en ese espacio. Si por ejemplo un patrón se está ejecutando no es necesario que realice el control de la pizarra de manera continua, en caso contrario podría hacer ese control periódicamente para verificar su posterior disparo.

Como una cuestión alternativa de final de ejecución de los patrones el concepto de *inhibidores*, mencionado en capítulos y secciones anteriores, resulta especialmente adecuado para producir la detención de su comportamiento. Podemos considerarlos como condicio-

nes negativas, es decir, que ante su presencia el despliegue del patrón finaliza, se *inhibe*, si esto no sucediera podría ejecutarse indefinidamente. Utilizando el mismo enfoque que con las condiciones positivas, un inhibidor bien puede ser una tupla de distinto signo o de contenido diferente. La presencia de esta en el espacio de tuplas no permitirá que el patrón afectado realice el disparo de la condición positiva para su nuevo lanzamiento, y en ese caso se producirá la finalización de la ejecución del patrón de una manera moderada, se ejecutará hasta su final en el momento previsto.

5.3. Lenguaje de Patrones

El concepto de *lenguaje de patrones*, así como la noción propia de *patrón*, fue definida originalmente dentro del campo de la arquitectura de edificios por el arquitecto Christopher Alexander [8, 7]. El concepto ya conocido de *patrón de diseño* [98, 38] puede ser resumido como *la descripción de una plantilla de una solución a un problema en un contexto adecuado*.

De manera similar, y a grandes rasgos, la noción de lenguaje de patrones puede ser definida como una red de patrones, es decir, *un conjunto de patrones que trabajan juntos para definir una actividad compleja o una estrategia de diseño*. En [241] se presenta un análisis de las complejas relaciones entre los patrones de diseño, organización y categorización de los mismos como base para el desarrollo de un lenguaje de patrones.

Como se ha dicho en el Capítulo 4, la definición de Alexander concibe al lenguaje de patrones como un método estructurado para describir buenas prácticas en la resolución de los problemas comunes de diseño. *"Cada patrón [está] conectado con otros patrones, de tal manera que se obtiene una colección (...) como un todo, como un lenguaje, dentro del cual se puede lograr una infinita varidad de combinaciones"*. Asimismo afirma que *"ningún patrón es una entidad aislada. Cada patrón puede existir en el mundo, pero solo en la medida que es apoyado por otros patrones"* [8]. Los lenguajes de patrones no son concebidos como definiciones cerradas, así como los patrones individuales deben ser adaptados a escenarios específicos, los lenguajes pueden ser fácilmente ampliados con el agregado de nuevos patrones y nuevas relaciones entre ellos.

Además del lenguaje que él ha definido también han sido de utilidad otras prácticas que ha propuesto: por ejemplo que toda secuencia de patrones no solo debe ser lógica, sino que debe aportar valor; o que el campo de uso de patrones puede superponerse ya que ellos pueden proponer soluciones distintas al mismo problema [8].

Es interesante destacar que según Buschmann et al. [38] llamar "lenguaje de patrones" a un conjunto de patrones es, de alguna manera, excesivo. Para estos autores hablar de lenguaje implica que sus patrones constituyentes cubren todos los aspectos de importancia en un dominio particular. Y si lo aplicamos a arquitectura de software, el lenguaje debería ser computacionalmente completo: al menos un patrón debe cubrir los aspectos constructivos y de implementación del sistema [38]. Siguiendo este razonamiento nuestro conjunto de patrones de adaptación sería un sistema antes que un lenguaje, sin embargo continuaremos utilizando el término *lenguaje* porque consideramos que ofrece mayor expresividad y claridad, a pesar de no ser completo debido a que el enfoque es evolutivo y el lenguaje puede modificarse agregando o quitando patrones.

No obstante, y pese a llamarlo lenguaje de patrones, este *no* es un clásico lenguaje (como sería, por ejemplo, uno de programación), sino más bien una red de construcciones de conocimiento para el diseño. Puede ser visto como un lenguaje, ya que tiene un *vocabulario* (el conjunto de patrones), una *sintaxis* (las relaciones entre los mismos), y una *gramática* (es decir, cómo operan para resolver el problema, tanto individualmente como en conjunto). Pero de hecho no es un lenguaje para la comunicación sino que está concebido para una actividad compleja diferente: el *diseño* de una arquitectura.

Desde sus orígenes, el concepto de lenguaje (o sistema) de patrones ha sido aplicado a muchas soluciones complejas, en particular a un nivel arquitectónico [38]. También ha sido aplicado en el contexto específico de arquitecturas orientadas a agentes, como en [226, 165, 102], entre otros.

El propósito de nuestro *lenguaje de patrones* es representar los pasos para definir el ciclo de vida que se ha visto en el Capítulo anterior y que permita obtener las propiedades adaptativas que se buscan. Los dieciocho patrones y sus relaciones tratan de cubrir las distintas alternativas posibles, aunque es muy probable que no se agoten en todas las que se

proponen. Haciendo referencia a la categorización que se verá posteriormente, el lenguaje de patrones propone cubrir las posibilidades que corresponden a la *creación* y *destrucción* de organizaciones, cómo *unirse* o *separarse* de ellas, facilitar sus *comunicaciones*, además de *observarlas*, lograr *coordinación* y también su *evolución*. Es de esperarse el crecimiento de este lenguaje agregando nuevos patrones si se identifican nuevos comportamientos, lo que puede considerarse parte del trabajo futuro relacionado con esta tesis.

5.4. Clasificación

Para hacer un uso eficiente de los patrones de software se requiere algún tipo de categorización o clasificación. La función del esquema de clasificación es facilitar la selección del patrón de acuerdo al problema que se intenta resolver, por lo que estas categorizaciones deben ser, en la medida de lo posible, amplias y concisas. Ser amplias para abarcar la mayoría de los problemas que pueden resolver, y concisas para evitar confusión cuando se realice la correspondiente selección del patrón a utilizar [165].

A estos efectos proponemos adoptar dos criterios para la clasificación de nuestros *patrones de adaptación*. El primero de ellos está presente en el trabajo [190] y el segundo es una categorización propuesta a medida para esta tesis.

Adoptando el criterio de clasificación de [190] es posible categorizar los patrones de la siguiente manera:

- Si nos basamos en los objetivos generales, los patrones pueden ser clasificados como *monitores* (monitoring, M), los que “observan” al sistema y a las condiciones del entorno para determinar que es factible llevar adelante una adaptación; los patrones *decisores* (decision-making, DM) pueden determinar el momento adecuado para la adaptación; y los patrones *reconfiguradores* (reconfiguration, R) serán los responsables de llevar adelante los cambios estructurales o de comportamiento.

- A su vez, los patrones M y DM también pueden ser clasificados como *creadores* (creational, C) o *estructurales* (structural, S), teniendo en cuenta definiciones similares a [98] adaptadas al enfoque de este trabajo.

- Por otro lado, los patrones R también pueden ser clasificados como *de comportamiento* (behavioral, B) y estructurales (S), ya que pueden especificar cómo modificar una arquitectura una vez que el sistema haya llegado a un estado seguro para la adaptación.

Esta clasificación está representada en la Tabla 5.1 y será utilizada cuando se describan cada uno de los patrones de adaptación en secciones posteriores.

Tabla 5.1: Clasificación 1 de los Patrones de Adaptación (inspirada en [190])

	C	S	B
M	Patrón_X	Patrón_Y	
DM	Patrón_W	Patrón_Z	
R		Patrón_P	Patrón_Q

Tabla 5.2: Clasificación 2 de los Patrones de Adaptación.

Categorías	Crear Org.	*Iniciativa* (CR-I)	**Gathering**		
		Org. Plena (CR-O)	**Consolider**		
	Destruir Org.	(DE) **Terminator**			
	Unión	*Directa* (UN-D)	**Gathering**		
			Full Mesh		
		Indirecta (UN-I)	**Thru Mediator**		
	Separación	(SE) **Retirement**			
	Comunicación	*Interna*	*Directa* (CM-ID)	**Gathering**	
				Full Mesh	
			Indirecta (CM-II)	**Mediator**	**New Mediator**
					Asign Mediator
		Externa	*Directa* (No permitida)		
			Indirecta (CM-EI)	**Façade**	**Update Façade**
	Coordinación	*Planificado* (CO-P)	**Planner**		
		Ad-hoc (directa) (CO-D)	**Surveyor**	**Elect Surveyor**	
				Change Surveyor	
	Observación	*Interior* (OB-I)	**Monitor**		
		Exterior (OB-E)	**Environment**		
	Evolución	(EV) **Transformer**			

Como segundo criterio de clasificación proponemos categorizar nuestros patrones de adaptación haciendo referencia a las posibilidades que pretende cubrir el lenguaje de patro-

nes. Tenemos los siguientes casos: *creación* de una *iniciativa* o de una organización plena; *destrucción* de una organización; la *unión* directa o indirecta a los grupos (u organizaciones); también cómo *separarse* de ellas; facilitar la *comunicación* interna y externa; lograr *coordinación*, planificada o ad-hoc; permitir la *observación* tanto interior como exterior; y también considerar su *evolución*.

En la Tabla 5.2 puede observarse esta última categorización, la que también será utilizada en las descripciones posteriores de los patrones.

5.5. Catálogo de los *Patrones de Adaptación*

En esta sección se presenta el catálogo de los *patrones de adaptación* identificados en este trabajo.

La lista de los patrones puede verse en la Tabla 5.3 acompañados de una breve descripción.

Tabla 5.3: Patrones de adaptación: patrones de diseño a nivel arquitectónico.

Nombre del Patrón	**Descripción**
Gathering	Describe el inicio de la creación de un grupo preliminar, el que se obtiene por la reunión de un conjunto de agentes que se encuentran en un mismo lugar, que no están previamente relacionados y que tendrán la posibilidad de generar interacciones y conversaciones.
Elect Surveyor	Describe la manera en que un líder puede ser seleccionado entre los agentes que se hallan reunidos en determinado lugar por el patrón *Gathering*. Dependiendo de las circunstancias, pueden ser utilizados diferentes algoritmos de elección en sistemas distribuidos adaptándolos al contexto correspondiente [147, 95], como por ejemplo el algoritmo de anillo.

Tabla 5.3: Continuación Tabla Patrones de adaptación.

Surveyor	Describe el comportamiento básico del agente líder, funciona básicamente como un coordinador debido a que la organización aún no está en pleno funcionamiento y al menos un agente debe controlar su crecimiento y redirigir la mayoría de los mensajes hacia los agentes correspondientes. Una vez que un *surveyor* ha sido definido podemos considerar que la reunión de agentes puede transformarse en una *iniciativa*.
Change Surveyor	Describe la manera en que puede lanzarse un protocolo para sustituir (potencialmente) al actual *surveyor*. Las razones del cambio van desde proponer un nuevo candidato hasta el propio deseo del *surveyor* de renunciar a su puesto. En caso de decidirse el cambio se debe generar el proceso de elección.
Façade	Define una fachada, o lo que es lo mismo, define un agente "encargado" de llevar adelante cualquier interacción con elementos del exterior, por ejemplo todos aquellos que no pertenecen a la *iniciativa*. Esto hace posible concebir externamente a esta última como una entidad única. Es análogo al clásico patrón *Façade* de [98].
Update Façade	Describe el proceso para relevar al agente fachada actual, de manera que sus responsabilidades y funciones sean asignados al nuevo agente fachada.
Mediator	Describe cuando a un agente se le provee con la capacidad de actuar como una entidad inteligente intermediaria ante otros agentes, que de otra manera no podrían comunicarse, por ejemplo, por tratarse de agentes heterogéneos. Facilita la interacción y coordinación entre los integrantes del grupo preliminar.

Tabla 5.3: Continuación Tabla Patrones de adaptación.

Assign Mediator	Describe el protocolo para la asignación de otro *mediator* en sustitución del actual. Puede ser seleccionado, por ejemplo, porque posee nuevos conocimientos que son necesarios para continuar con el crecimiento de la *iniciativa*.
New Mediator	Describe el proceso para la asignación de un nuevo *mediator* ante el hecho de que el actual debe renunciar a su rol o ha dejado de funcionar como tal, por ejemplo por la necesidad de abandonar el grupo preliminar.
Thru Mediator	Establece la conexión entre un *mediator* ya determinado y un nuevo agente, al que se provee con el potencial de interactuar con el resto de la *iniciativa*.
Full Mesh	Provee una conexión *full-mesh* ("todos-con-todos") entre un agente que ingresa y el resto de elementos de la *iniciativa*. Es el opuesto semánticamente al patrón anterior: el nuevo agente tiene la capacidad para conectarse directamente con todos, sin requerir un mediador.
Retirement	Describe el proceso que debe seguir un agente para abandonar la *iniciativa*, volviéndose un "agente libre" nuevamente.
Consolider	Describe el proceso para consolidar y registrar una *iniciativa*, transformándola en una organización completa con todas las características de las organizaciones ya existentes.
Planner	Describe cuando un agente adquiere la capacidad de planificar una respuesta inteligente ante una situación. Es un planificador estándar, pero debe tener en cuenta los recursos que se encuentran presentes en la *iniciativa*.
Transformer	Describe el proceso que permite "transformar" el comportamiento de un agente, es decir, que el agente logre pasar de su rol definido a jugar otro.

Tabla 5.3: Continuación Tabla Patrones de adaptación.

Monitor	Las interacción relevantes dentro de la *iniciativa* pueden ser capturadas y filtradas de acuerdo a las suscripciones en este monitor. Hace posible que el *surveyor* descubra las capacidades actuales en la organización cambiante.
Terminator	Dispara el final de la actual *iniciativa*, sin que llegue a formarse una organización. Es lo opuesto al patrón *Consolider*, es lanzado una vez que haya solo un miembro en el grupo.
Environment	Su objetivo es permitir la observación hacia el exterior de la *iniciativa*, funcionalmente es más proactivo.

A continuación se realizará la descripción de cada uno de los patrones de adaptación utilizando la plantilla presentada anteriormente.

5.5.1. Patrón *Gathering*.

- Nombre del Patrón: **Gathering**.
- Clasificación:
  - *Clasificación 1*: Monitor (M), Creador (C)
  - *Clasificación 2*: CR-I (Creación Iniciativa), UN-D (Unión Directa), CM-ID (Comunicación Interna Directa)
- Intención: monitoriza el "espacio" que contiene elementos computacionales (agentes), canaliza los datos observados, facilita la interacción y, en última instancia, proporciona alguna forma de coordinación.
- Contexto: el patrón Gathering puede ser utilizado cuando:
  - aún no se tiene una *iniciativa* formada
  - los agentes que se reúnen en determinado lugar no se conocen, por lo que aún no tienen la capacidad para interactuar (conversar)

  - todavía no se han definido servicios de alto nivel proporcionando una organización

- Motivación: el sistema necesita coordinarse como un ecosistema de servicios en el que los agentes puedan entrar o abandonar el entorno dinámicamente. En un punto de encuentro se hallan agentes que no están previamente relacionados, los que en conjunto deberán trabajar (colaborar y cooperar) para brindar los servicios de alto nivel por los que fueron reunidos. El patrón Gathering facilita la interacción de los agentes en ese punto de encuentro. Luego de la ocurrencia de un evento en este punto, los agentes se encuentran capacitados para interactuar directamente entre sí, de esta manera puede dispararse el proceso de creación de una *iniciativa*.

- Esquema: en la Figura 5.1 se representa la interacción entre el sistema, el agente *venue* y dos agentes diferentes. Estos últimos en principio se reúnen en un punto de encuentro, no se conocen y deben interactuar.

- Participantes: Gathering utiliza dos o más agentes y un agente *venue* (representando el lugar de encuentro). La infraestructura subyacente debe cumplir el rol de conector.

- Comportamiento: la Figura 5.2 representa el comportamiento del patrón, según el cual el Agente_1 cercano a un punto de encuentro puede conocer al Agente_2, para que posteriormente pueda generarse conversación entre ellos.

- Consecuencias:

  - permite monitorizar a los agentes que pueblan el "espacio", a los que se les facilitan canales para conocerse y comunicarse.
  - dirige los datos monitorizados a los respectivos agentes.
  - facilita la interacción entre los agentes del "espacio" hasta tanto puedan comunicarse por sí mismos.
  - facilita la coordinación al intermediar entre agentes que se reúnen en el punto de encuentro.

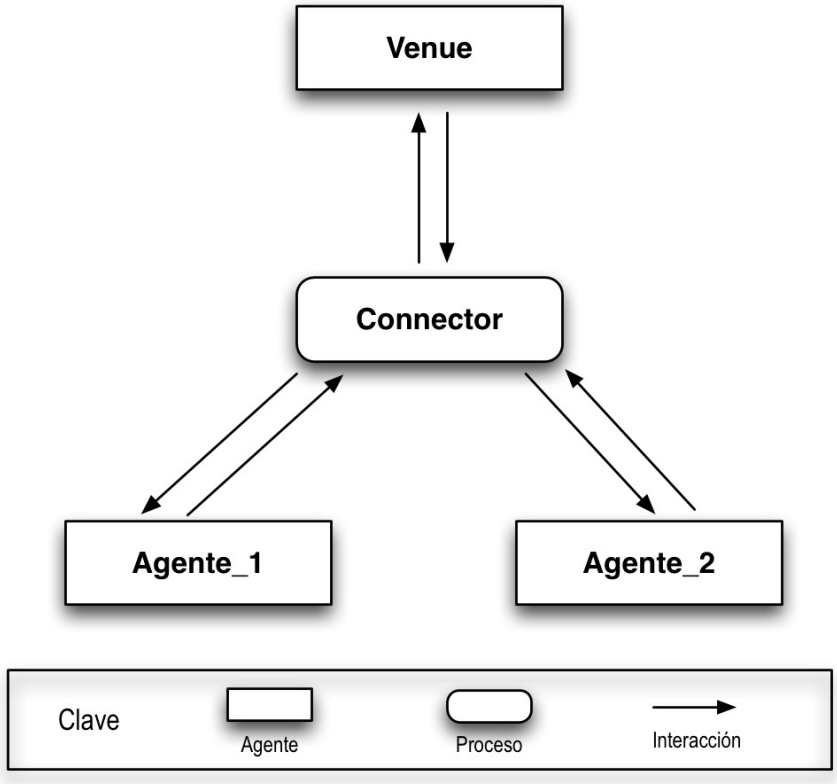

Figura 5.1: Esquema del patrón *Gathering*.

- Restricciones: no podrá usarse Gathering cuando todos los agentes se conocen y ya intervienen en conversaciones. Tampoco podrá utilizarse cuando la *iniciativa* ya está formada.

- Patrones relacionados: el patrón Elect Surveyor puede utilizarse para lanzar la elección de un surveyor una vez que los agentes ya se conocen y pueden realizar conversaciones como resultado del Gathering. Este último ya cuenta con la lista de agentes reunidos, a efectos de lanzar la elección.

Modelo de interacción en Cálculo-π.

Aclaración: AgentX será creado por otro proceso, el Creator_Agent y Venue será creado por Space, lo que se verá más adelante y en los patrones correspondientes.

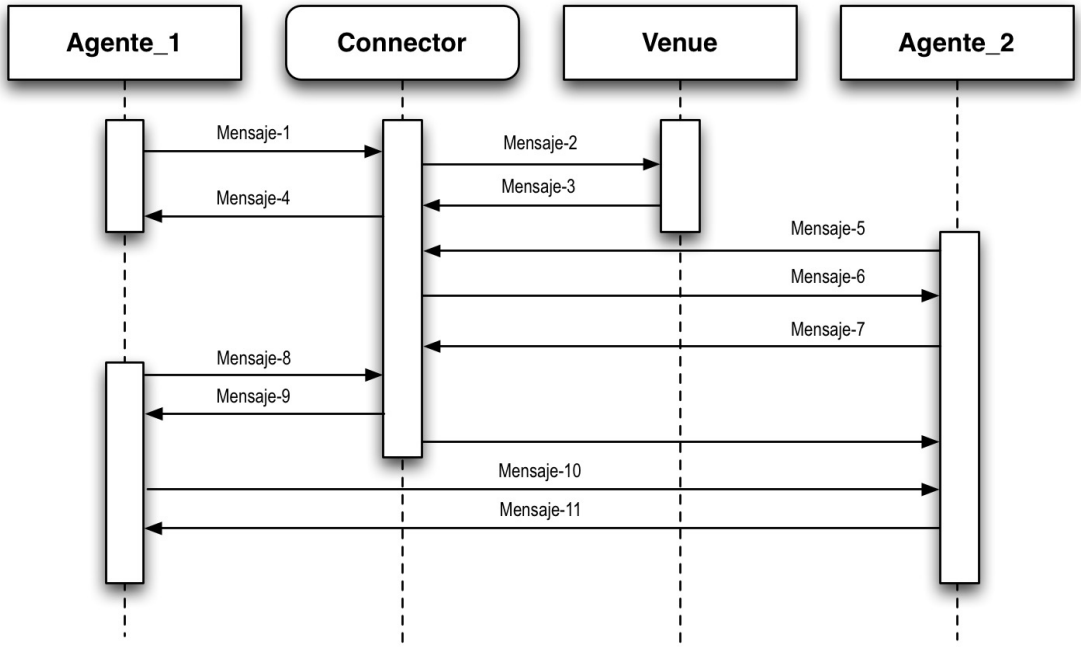

Figura 5.2: Ejemplo de comportamiento del patrón *Gathering*.

El símbolo asterisco ($*$) representa la operación de replicación; se asume una construcción (;) para una composición secuencial, y se incluye además el condicional (if A then B). Ambos se codifican convencionalmente de una manera estándar [207]. Se agregan también dos funciones simples, fácilmente implementables, por conveniencia algorítmica: **near** (detecta proximidad) y **count** (para contar elementos).

A continuación se presenta un extracto de la especificación en Cálculo-π y una breve explicación de su funcionamiento.

$$
\begin{aligned}
&\mathrm{AgentX}(\mathrm{sys\_where}) \triangleq \ldots \\
&\quad + (\nu\ hi)\,\mathrm{sys\_where!}\langle hi\rangle.\ \mathrm{sys\_where?}(\mathrm{where}).\ \tau.\ \mathrm{where?}(\mathrm{event}). \\
&\qquad (\nu\ me, \mathrm{ret}) * [\mathrm{event!}\langle me, \mathrm{ret}\rangle.\ \mathrm{ret?}(you).\ \tau.\ (me!\langle \mathrm{data}\rangle |\ you?(\mathrm{info}))]; \\
&\qquad \mathrm{AgentX}\langle \mathrm{sys\_where}\rangle \\
&\quad + \ldots
\end{aligned}
$$

$$
\begin{aligned}
&\text{Venue}(\text{sys\_place}) \triangleq \ldots \\
&\quad + (\nu\ \text{here}, r)\ \text{sys\_place}!\langle \text{here}, r\rangle.\ \tau. \\
&\qquad * [r?(hi).\ \underline{\text{if}}\ \textbf{count}(r) \geq 2\ \underline{\text{then}}\ (\nu\ \text{event})\ \text{here}!\langle \text{event}\rangle. \\
&\qquad\quad * [\text{event}?(\text{ag\_x}, ragx).\ (\\
&\qquad\qquad * [\text{event}?(\text{ag\_y}, ragy).ragx!\langle \text{ag\_y}\rangle]\ |\ \text{event}!\langle \text{ag\_x}, ragx\rangle)\ ]\ ]; \\
&\qquad\qquad \text{Venue}\langle \text{sys\_place}\rangle \\
&\quad + \ldots
\end{aligned}
$$

$$
\begin{aligned}
&\text{Connector}(\text{sys\_place}, \text{sys\_where}) \triangleq \ldots \\
&\quad + \text{sys\_place}?(\text{here}, r). \\
&\qquad * [\text{sys\_where}?(hi).\ \underline{\text{if}}\ \textbf{near}(\text{here}, hi)\ \underline{\text{then}}\ (\text{sys\_where}!\langle \text{here}\rangle.\ r!\langle hi\rangle)\ ]; \\
&\qquad \text{Connector}\langle \text{sys\_place}, \text{sys\_where}\rangle \\
&\quad + \ldots
\end{aligned}
$$

El proceso AgentX describe la manera en que un agente estándar debe comportarse en el patrón. En primer lugar, es consciente de su existencia (hi) y pregunta al sistema dónde se encuentra (sys_where). Si está cerca de un punto de encuentro (venue) se le envía su nombre (where). Cuando algo (event) sucede allí, el agente envía su propio nombre (me) y un canal de respuesta (ret). Por este canal, recibe el nombre de otro agente (you).

Por lo tanto, los dos agentes pueden ahora interactuar directamente, y no volver a usar el venue. Esta última parte es un sub-proceso repetitivo ($*$[A]), que se repite indefinida y concurrentemente – lo que significa que el agente puede comunicarse con muchos otros, dos a dos.

El proceso Venue describe la influencia del "punto de encuentro" en los agentes. Primero informa al sistema de su existencia (here), y provee un canal de respuesta (r), en el que los agentes eventualmente podrán registrarse. Una vez que haya dos o más agentes, el evento se dispara y se retransmite dentro de un proceso repetitivo. El canal event es utilizado para recibir el nombre (ag_x) y devolver el canal a todos los agentes cercanos, encontrar una pareja potencial (ag_y) y comunicarse con ella, dentro de un proceso paralelo. Por supuesto hay otras maneras más sofisticadas para definir un protocolo de difusión (broadcast): éste es solo un ejemplo.

El proceso Connector actúa, como su nombre lo indica, de conector definiendo el comportamiento de la porción de *middleware* que aglutina al conjunto.

Como comentario, podemos ver que cada proceso AgentX es un agente *shifter* y el Venue es un agente *changent* (4.3.2).

El patrón *Gathering* presenta sus resultados como si se tratara de una "alineación" (*alignment*) de los agentes involucrados. Esto es, una vez que hayan pasado por los procesos del patrón estarán alineados, habrán llegado a un acuerdo (*agreement*) para conversar e interactuar, compartiendo ontología, metas, estándares, lugar, etc. Todo esto puede verse como un simple ejemplo de coordinación para homogeneizar el comportamiento del grupo.

5.5.2. Patrón *Elect Surveyor*.

- Nombre del Patrón: **Elect Surveyor**.
- Clasificación:
  - *Clasificación 1*: Reconfigurador (R), Estructural (S)
  - *Clasificación 2*: CO-D (Coordinación Directa)
- Intención: provee los mecanismos para que se realice la elección de un líder organizacional (*surveyor*) entre los agentes que se hayan reunidos previamente por el patrón Gathering. Dependiendo del contexto pueden ser utilizados diferentes algoritmos de elección en sistemas distribuidos [147, 95].
- Contexto: el patrón Elect Surveyor puede ser utilizado cuando:
  - aún no se tiene una *iniciativa* formada
  - es necesario contar con un líder organizacional
- Motivación: un grupo de agentes se hallan reunidos en cierto lugar para proveer determinados servicios. La organización preliminar se encuentra emergiendo, la *iniciativa* todavía no existe aunque los agentes ya se conocen y pueden interactuar. Es importante la presencia de un líder, el que será elegido de entre los agentes que se hallan

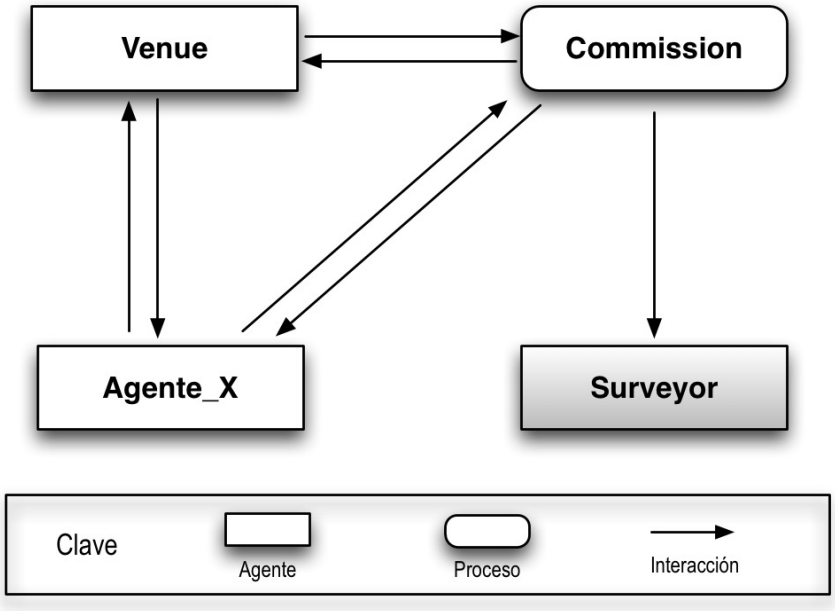

Figura 5.3: Esquema del patrón *Elect Surveyor*.

en una lista de participantes. El patrón definirá el protocolo (uno entre varios) para realizar esa elección.

- Esquema: en el esquema de la Figura 5.3 el agente Venue y los demás agentes interactúan con el proceso Commission. Utilizando determinado algoritmo de elección definido según el dominio se elige al líder de la *iniciativa*: el agente Surveyor.

- Participantes: el patrón utiliza dos o más agentes de tipo *shifter*, ya que ahora cuentan con capacidades de interacción y conversación después de haber pasado por Gathering. Uno de ellos puede ser elegido como Surveyor. También participa un agente *Venue*.

- Comportamiento: el comportamiento detallado se desarrolla en el modelo de interacción hecho a continuación en Cálculo-π. La Figura 5.4 representa un comportamiento genérico de un algoritmo de anillo para un algoritmo de elección determinado [147]: el ejemplo trata de agentes sanitarios que atienden urgencias de distinto tipo y el criterio de elección es que el líder será aquel que atienda urgencias de mayor cate-

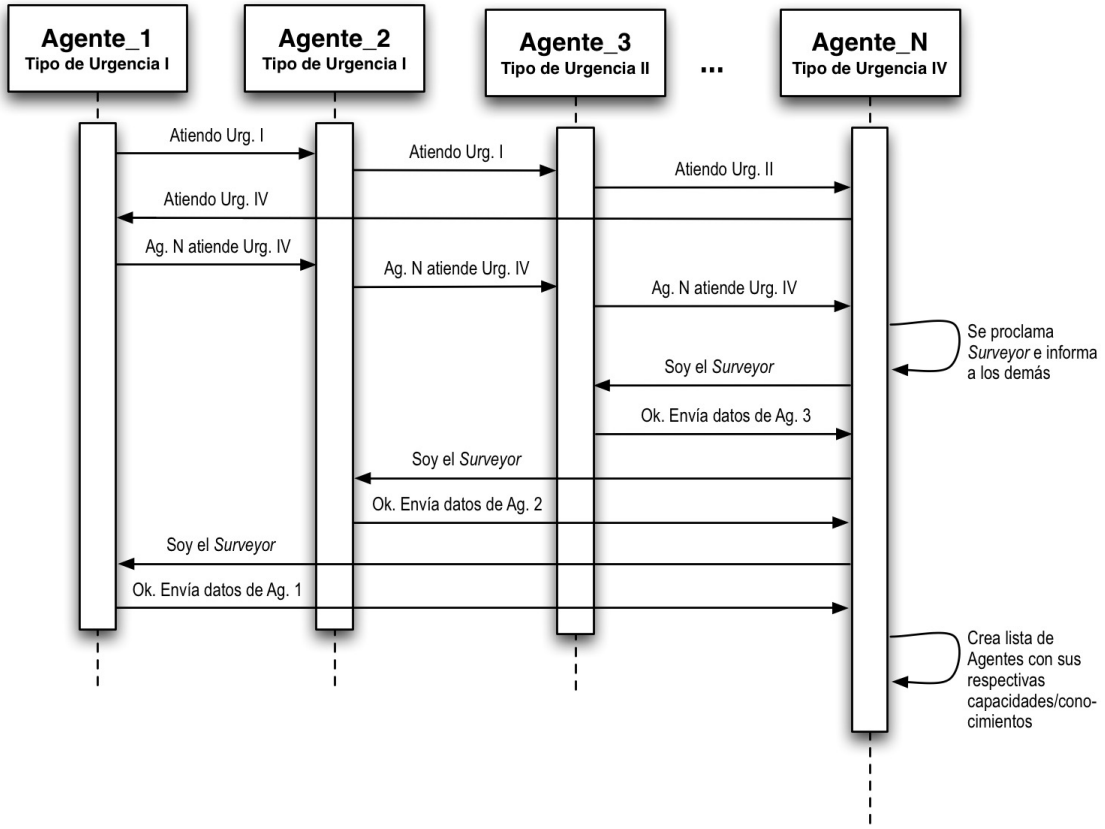

Figura 5.4: Ejemplo de algoritmo de elección basado en anillo.

goría. En este caso el agente N (Tipo IV) se proclama líder, informa a los demás y los incluye en la lista de integrantes del grupo preliminar.

- Consecuencias:
  - un agente es elegido líder organizacional.
  - la lista de agentes reunidos genera el grupo preliminar.
  - permite que la *iniciativa* pueda continuar creciendo.
- Restricciones: no podrá utilizarse Elect Surveyor si los agentes aún no se conocen y por lo tanto no pueden interactuar. El patrón no podrá ser usado si no se cuenta con la lista (censo) de los agentes reunidos y candidatos para la elección. Tampoco

se usará si ya existe una *iniciativa* formada o si ya se ha elegido un líder (surveyor), a menos que haya que cambiarlo por uno nuevo. Debe definirse el algoritmo de elección adecuado al contexto.

- Patrones relacionados: Gathering y Change Surveyor son patrones que pueden utilizarse con anterioridad a Elect Surveyor con el fin de realizar la elección del líder (o nuevo líder, según el caso). Surveyor es un patrón que puede dispararse con posterioridad una vez que se ha elegido líder, el cual llevará adelante su comportamiento básico correspondiente según el patrón.

Modelo de interacción en Cálculo-π.

Creator_Agent $() \triangleq$

$(\nu\, dni, try, \textbf{\textit{no\_med}}, \textbf{\textit{ch\_med}}, log)$(AgentX $\langle dni, try, \textbf{\textit{no\_med}}, \textbf{\textit{ch\_med}}, log, \dots \rangle$ |

$(\nu\, nulo)\textbf{\textit{no\_med}}!\, \langle nulo \rangle\, .\emptyset)$

Venue $(\dots, \textbf{\textit{sys\_place}}, \textbf{\textit{sys\_elect}}, \textbf{\textit{sys\_poll}}, \textbf{\textit{sys\_vote}}, \dots) \triangleq$

$\dots$

$+\, \textbf{\textit{sys\_elect}}?(ballot).(\nu\, \widehat{census}, poll)($

$timeout!\, \langle poll, 3000 \rangle$ |

Census_List $\langle \widehat{census}, \rangle$ |

$*\, [\textbf{\textit{sys\_poll}}!\, \langle poll \rangle\, .poll?(dni).$

$(\nu\, r)\textbf{\textit{census}}_o!\, \langle r \rangle\, .r?(\widetilde{x}).\textbf{\textit{census}}_i!\, \langle dni, \widetilde{x} \rangle\,]$ |

$timeout?(c, \rangle\, .$

if $(c = poll)$ then

$\textbf{\textit{sys\_vote}}!\, \langle ballot, \widehat{census} \rangle$

else

$timeout!\, \langle c, 0 \rangle\,);$

Venue $\langle \dots, \textbf{\textit{sys\_place}}, \textbf{\textit{sys\_poll}}, \textbf{\textit{sys\_vote}}, \dots \rangle$

AgentX $(dni, try,$ ***no_med***$,$ ***ch_med***$, log, \ldots,$ ***sys_elect***$,$ ***sys_poll***$,$ ***sys_proclaim***$,$ ***sys_office***$, \ldots) \triangleq$

$(\nu\, die, \widehat{who\_surveyor})(\ldots$

$dni?(ret).\tau\,(f).ret!\,\langle f, try,$ ***no_med***$,$ ***ch_med***$, log\rangle\,.$

AgentX $\langle dni, try,$ ***no_med***$,$ ***ch_med***$, log, \ldots\rangle$

$+\,(\nu\, ballot)$***sys_elect***$!\,\langle ballot\rangle\,.$

AgentX $\langle dni, \ldots,$ ***sys_elect***$,$ ***sys_poll***$,$ ***sys_proclaim***$,$ ***sys_office***$, \ldots\rangle$

$+$ ***sys_poll***$?(poll).$

$\underline{\text{if}}$ **near** $(poll, dni)$ $\underline{\text{then}}$

$poll!\,\langle dni\rangle\,.$

AgentX $\langle dni, \ldots,$ ***sys_elect***$,$ ***sys_poll***$,$ ***sys_proclaim***$,$ ***sys_office***$, \ldots\rangle$

$+$ ***sys_proclaim***$?(winner).$

$\underline{\text{if}}\ (winner = dni)\ \underline{\text{then}}$

$(\nu\, boss)$(Surveyor $\langle boss\rangle$ $\mid$

$*\,[$***sys_office***$!\,\langle boss\rangle\,]);$

AgentX $\langle dni, \ldots,$ ***sys_elect***$,$ ***sys_poll***$,$ ***sys_proclaim***$,$ ***sys_office***$, \ldots\rangle$

$+$ ***sys_office***$?(boss).$***who_surveyor***$_i!\,\langle boss\rangle$

AgentX $\langle dni, \ldots,$ ***sys_elect***$,$ ***sys_poll***$,$

sys_proclaim$,$ ***sys_office***$, \ldots\rangle\,)$

Commission $(\textit{sys\_vote}, \textit{sys\_proclaim}) \triangleq$

$\textit{sys\_vote}?(ballot, \widehat{census}).$

$(\nu\, today)(\text{ELECTION\_ALGORITHM}\ \langle today, \widehat{census}\rangle\ ;$

$today?(winner).$

$\textit{sys\_proclaim}!\ \langle winner\rangle\ .\textit{sys\_office}?(surveyor).$

$(\nu\, ret)surveyor!\ \langle ret\rangle\ .ret?(\textit{meth}_o).$

$(\nu\, r)\textit{meth}_o!\ \langle r\rangle\ .r?(\widetilde{m}).$

$m_2!\ \langle\widehat{census}\rangle\ .$

Commission $\langle \textit{sys\_vote}, \textit{sys\_proclaim}\rangle$

Census_List $(\widehat{census}, \widetilde{c}) \triangleq$

$\textit{census}_i?(x).$Census_List $\langle\widehat{census}, \widetilde{x}\rangle$

$+\ \textit{census}_o?(r).r!\ \langle\widetilde{c}\rangle\ .$Census_List $\langle\widehat{census}, \widetilde{c}\rangle$

Aclaraciones: con el fin de simplificar y abreviar se utiliza la variable colectiva $\hat{x}$ que reúne en un array y que se usa siempre que sea posible para responder; $\hat{x}$ está compuesta de **x1**: f (la función principal interna del agente), **x2**: try (para intentar una comunicación), **x3**: no_med (para informar que se elimina mediador), **x4**: ch_med (para informar al adjunto de su nuevo mediador), y **x5**: log (información utilizada por el patrón Monitor, 5.5.16). Excepto f, los demás consisten en parámetros.

Census_List es una lista con *in* y *out* para agregar elementos a la lista y también para enviar dicha lista cuando la soliciten.

La notación con tilde ($\widehat{ejemplo}$) se utiliza para representar el par de canales entrada-salida (*in* y *out*).

Se utilizan listas y vectores como intercambiables y *timeout* es un "canal especial", un temporizador.

add_list consiste en $(\nu r)census\_o!\langle r\rangle.r?(\tilde{x}).census\_i!\langle dni, \tilde{x}\rangle =$ **add_list**$\langle\widehat{census}, dni\rangle$. A partir de ahora se asume esta abreviatura y que se trabaja con arrays. Funciones similares también se definen de esta manera (**search_list**, **insert_list**, entre otras).

A continuación se describen brevemente las interacciones.

El proceso *Creator_Agent* es el *"bootstrap"*, se crean la variables que utilizará *AgentX*. Se invoca a este y en paralelo, y en el comienzo, el agente se crea sin mediadores (no_med).

El proceso *AgentX* es invocado con todos los parámetros que corresponden a este patrón, entre ellos los de sistema *sys_elect, sys_poll, sys_proclain, sys_office* que serán utilizados en la elección del surveyor. Luego de crear variables locales se recibe por el canal identidad otro canal (ret), por este puede enviar su propia función (f) y los demás parámetros. A continuación se lanza el bucle nuevamente. En el primer subproceso llama a la votación (ballot); con el segundo si el agente está cerca del lugar de votación puede participar; con el tercero si el agente involucrado es el ganador él mismo invoca al proceso Surveyor y en paralelo y con una replicación se informa a los demás agentes; finalmente, en el último subproceso ese agente es proclamado líder.

El proceso *Venue* al inicio convoca a elecciones, se crea la lista del censo (vacía) y se lanza un "timeout" para generar el censo (se asume el retardo 3000, solo como un ejemplo); se genera el censo y si el agente está identificado participará de las elecciones (poll); si timeout ha acabado corresponde llamarse a elecciones con ese censo.

Commission (o "junta electoral") llama a elecciones, utilizando un algoritmo externo de elección se obtiene un resultado (winner), se proclama al ganador, éste acepta asumir su rol y a continuación con el método **m2** del Surveyor la lista del censo pasa a ser la lista de la *iniciativa*.

5.5.3. Patrón *Surveyor*.

- Nombre del Patrón: **Surveyor**.
- Clasificación:
  - *Clasificación 1*: Monitor (M), Estructural (S)
  - *Clasificación 2*: CO-D (Coordinación Directa)
- Intención: un agente surveyor controla el crecimiento de la *iniciativa* y determina el momento en que esta queda conformada. Funciona como un coordinador tanto

para decidir cuándo se unen nuevos elementos (agentes), como para asignar roles, definir en qué momento la *iniciativa* se estabiliza como organización plena, entre otras funciones importantes.

- Contexto: el patrón Surveyor puede ser utilizado cuando:
  - la *iniciativa* continúa en crecimiento
  - es necesario un control e inspección de la organización emergente para comprobar si podrá cumplir el objetivo inicial
  - es necesario agregar o remover miembros en el grupo preliminar en función del objetivo inicial
  - el líder organizacional ya ha sido elegido
  - es necesario tener acceso a la biblioteca de patrones para ser utilizado el que corresponda según cambios en el contexto
- Motivación: el sistema debe proveer los servicios de alto nivel que dispararon la adaptación. Se necesita inspeccionar constantemente a la organización emergente para comprobar si será capaz de ofrecer esos servicios controlando el crecimiento de la *iniciativa*. En caso de ser necesario, debido a cambios en el entorno o en el servicio de alto nivel a ser provisto, debe dispararse la activación de otro patrón de adaptación. El patrón Surveyor tiene acceso a la biblioteca de patrones. El patrón describirá el comportamiento básico del agente líder, esto es, redirigir la mayoría de los mensajes hacia los agentes que correspondan. Una *iniciativa*, para nuestro enfoque, consiste en un agente surveyor con una lista de los demás agentes y sus respectivos conocimientos y capacidades.
- Esquema: en la Figura 5.5 el agente Surveyor interactúa con todos los integrantes de la *iniciativa*.
- Participantes: dos o más agentes, uno de los cuales ha sido elegido líder de la *iniciativa* en crecimiento, puede ser de tipo *changent*. Los demás agentes (pueden ser tipo *shifter*) tienen sus propias metas y objetivos que en forma conjunta proveerían los servicios de alto nivel.

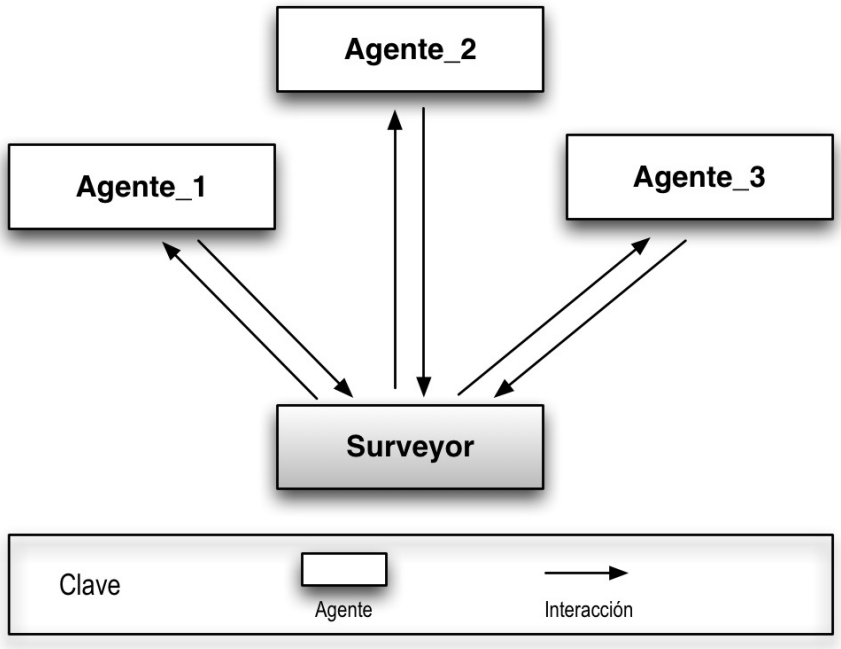

Figura 5.5: Esquema del patrón *Surveyor*.

- Comportamiento: en la Figura 5.6 se representa un ejemplo de comportamiento para este patrón. Un agente surveyor, líder de la *iniciativa*, agrega agentes a su lista de integrantes utilizando el método *add_member*, como se verá a continuación en la especificación en Cálculo-π. Una vez que los agentes están en la lista Surveyor puede tener otro tipo de interacciones con ellos.

- Consecuencias:

  - los agentes que integran la *iniciativa* pueden interactuar con el líder.
  - nuevos elementos (agentes) pueden ser agregados.
  - pueden ser removidos elementos (agentes) que no sean necesarios.
  - se pueden asignar roles a los integrantes de la *iniciativa*.
  - será posible definir el momento en que se ha formado la organización plena, capaz de ofrecer el servicio de alto nivel requerido.

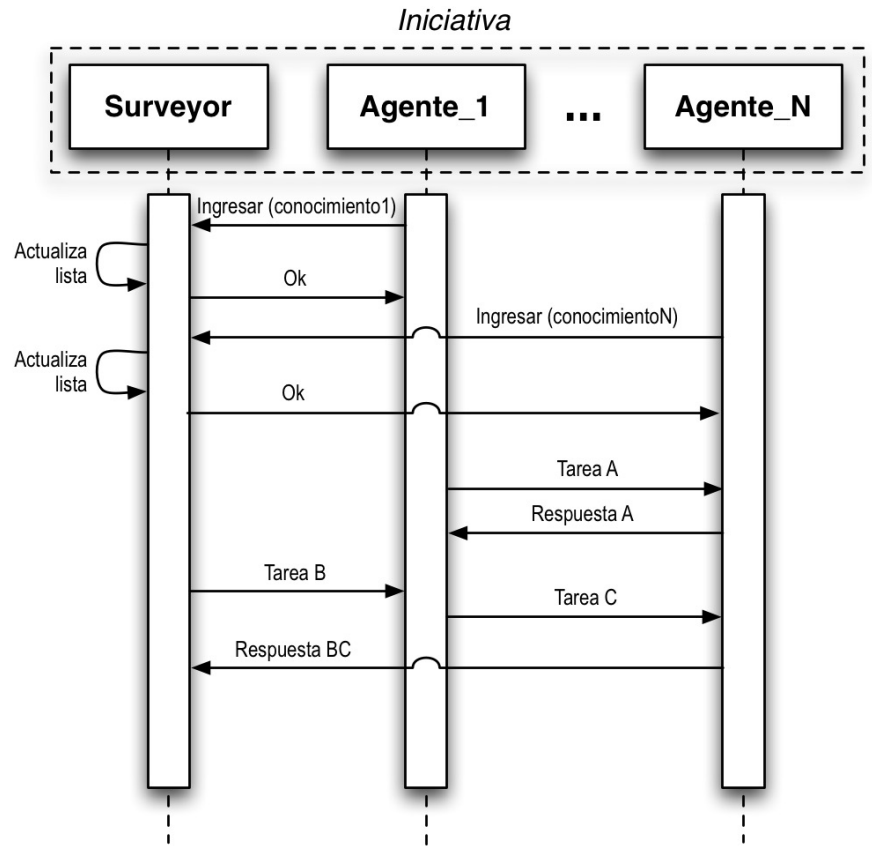

Figura 5.6: Ejemplo de comportamiento del patrón *Surveyor*.

- será posible decidir en qué momento cierto patrón de adaptación es el adecuado y realizar su activación.

- Restricciones: Surveyor no podrá ser utilizado cuando aún no esté elegido el líder organizacional de la *iniciativa*.

- Patrones relacionados: el patrón Surveyor se relaciona con prácticamente todos los demás patrones de nuestro lenguaje de patrones, según el estado en que se encuentre la *iniciativa*. La excepción es el patrón Gathering, ya que en este caso los agentes aún no pueden interactuar entre sí y además no se ha elegido líder o Surveyor.

Modelo de interacción en Cálculo-π.

AgentX $(dni, try, \textit{no\_med}, \textit{ch\_med}, log) \triangleq$

$dni?(ret).\tau\,(f).ret!\,\langle f, try, \textit{no\_med}, \textit{ch\_med}, log\rangle\,.$

AgentX $\langle dni, try, \textit{no\_med}, \textit{ch\_med}, log\rangle$

Method_List $(\widehat{meth}, \widetilde{c}) \triangleq$

$\textit{meth}_i?(x).$

$+\ \textit{meth}_o?(r).r!\,\langle\widetilde{c}\rangle$ .Method_List $\langle\widehat{meth}, \widetilde{c}\rangle$

Group_List $(\widehat{gr}, \widetilde{c}) \triangleq$

$\textit{gr}_i?(x)$.Group_List $\langle\widehat{gr}, \widetilde{x}\rangle$

$+\ \textit{gr}_o?(r).r!\,\langle\widetilde{c}\rangle$ .Group_List $\langle\widehat{gr}, \widetilde{c}\rangle$

Cap_List $(\widehat{gr}, \widetilde{c}) \triangleq$

$\textit{cap}_i?(x)$.Cap_List $\langle\widehat{cap}, \widetilde{x}\rangle$

$+\ \textit{cap}_o?(r).r!\,\langle\widetilde{c}\rangle$ .Cap_List $\langle\widehat{gr}, \widetilde{c}\rangle$

Surveyor $(me) \triangleq$

$(\nu\,\widehat{meth}, \widehat{group}, \widetilde{vl}, \textit{add\_member}, \textit{my\_group}, \textit{who\_group}, \textit{remove\_member}, \textit{what\_list}, \ldots,$

$\textit{die\_fac}, planner, \textit{die\_planner}, \ldots, bye)$

(Method_List $\langle\widehat{meth}, \widetilde{m}\rangle$ |

MySurveyor $\langle me, \widehat{meth}, \widehat{group}, \widetilde{vl}, \textit{die\_fac}, planner, \textit{die\_planner}, \widetilde{m}\rangle$

| Create_Planner $\langle planner, \textit{die\_planner}\rangle$)

MySurveyor $(me, \widehat{meth}, \widehat{group}, \widetilde{vl}, \textbf{\textit{die\_fac}}, planner, \textbf{\textit{die\_planner}}, \widetilde{m}) \triangleq$

$me?(ret).ret!\,\langle \textbf{\textit{meth}}_o \rangle\,.$
MySurveyor $\langle me, \widehat{meth}, \widehat{group}, \widetilde{vl}, \textbf{\textit{die\_fac}}, planner, \textbf{\textit{die\_planner}}, \widetilde{m} \rangle$

$\{\mathbf{m1}\}\ +\ \textit{add\_member}?(x).\textbf{\textit{add\_list}}\,\langle \widehat{group}, x \rangle\,.$
MySurveyor $\langle me, \widehat{meth}, \widehat{group}, \widetilde{vl}, \textbf{\textit{die\_fac}}, \ldots, \widetilde{m} \rangle$

$\{\mathbf{m2}\}\ +\ \textit{my\_group}?(\widehat{g}).$
$(\nu\, r)g_o!\,\langle r \rangle\,.r?(\widetilde{l}).\textbf{\textit{group}}_i!\,\langle \widetilde{l} \rangle\,.$
MySurveyor $\langle me, \widehat{meth}, \widehat{group}, \widetilde{vl}, \textbf{\textit{die\_fac}}, \ldots, \widetilde{m} \rangle$

$\{\mathbf{m3}\}\ +\ \textit{who\_group}?(ret).ret!\,\langle \widehat{group} \rangle\,.$
MySurveyor $\langle me, \widehat{meth}, \widehat{group}, \widetilde{vl}, \textbf{\textit{die\_fac}}, \ldots, \widetilde{m} \rangle$

$\{\mathbf{m4}\}\ +\ \textit{remove\_member}?(x).\textbf{\textit{remove\_list}}\,\langle \widehat{group}, x \rangle\,.$
MySurveyor $\langle me, \widehat{meth}, \widehat{group}, \widetilde{vl}, \textbf{\textit{die\_fac}}, \ldots, \widetilde{m} \rangle$

$\{\mathbf{m5}\}\ +\ \textit{what\_list}?(ret).$
$(\nu\, r)\textbf{\textit{group}}_o!\,\langle r \rangle\,.r?(\widetilde{l}).$
$(\nu\, \widetilde{c}, vcb, end, r)(\text{Cap\_Loop}\,\langle \widetilde{l}, vcb, end \rangle\ \mid$
$\text{C\_Loop}\,\langle 1, \widetilde{c}, vcb, end, r \rangle\ \mid$
$r?(\widetilde{z}).ret!\,\langle \widetilde{z} \rangle\,);$
MySurveyor $\langle me, \widehat{meth}, \widehat{group}, \widetilde{vl}, \textbf{\textit{die\_fac}}, \ldots, \widetilde{m} \rangle$

$\{\mathbf{m6}\}\ +\ \textit{who\_facade}?().\textit{who\_facade}!\,\langle fac \rangle$
MySurveyor $\langle me, \widehat{meth}, \widehat{group}, \widetilde{vl}, \textbf{\textit{die\_fac}}, \ldots, \widetilde{m} \rangle$

$\{\mathbf{m7}\}$ $+$ *create_facade*?().((ν *ret_list*)
what_list! $\langle$*ret_list*$\rangle$.*ret_list*?($\widetilde{\textit{capa\_list}}$).
τ ($\widetilde{\textit{capa\_list}}$, $\widetilde{\textit{exp\_list}}$).
(ν *die_fac*)Facade $\langle fac$, *exp_list*, *die_fac*$\rangle$ |
MySurveyor $\langle me, \widehat{meth}, \widehat{group}, \widetilde{vl}$, *die_fac*, $\ldots, \widetilde{m}\rangle$)

$\{\mathbf{m8}\}$ $+$ *kill_facade*?().
die_fac! $\langle\rangle$.
MySurveyor $\langle me, \widehat{meth}, \widehat{group}, \widetilde{vl}$, *die_fac*, $\ldots, \widetilde{m}\rangle$

$\{\mathbf{m9}\}$ $+$ *update_facade*?().(
kill_facade! $\langle\rangle$.*create_facade*! $\langle\rangle$ |
MySurveyor $\langle me, \widehat{meth}, \widehat{group}, \widetilde{vl}$, *die_fac*, $\ldots, \widetilde{m}\rangle$)

$\{\mathbf{m10}\}$ $+$ *full_mesh* $\ldots$

$\{\mathbf{m11}\}$ $+$ *create_mediator* $\ldots$

$\{\mathbf{m12}\}$ $+$ *replace_mediator* $\ldots$

$\{\mathbf{m13}\}$ $+$ *request_mediator* $\ldots$

$\{\mathbf{m14}\}$ $+$ *generate_plan* $\ldots$

$\{\mathbf{m15}\}$ $+$ *register_org* $\ldots$

$\{\mathbf{m16}\}$ $+ \mathit{bye}?().\mathit{die\_fac}!\,\langle\rangle\,.\mathit{die\_plan}!\,\langle\rangle\,.$
$(\nu\, r)\mathit{vdiem}_o!\,\langle r\rangle\,.r?(\widetilde{vd}).(\nu\, \mathit{end})$
$(\text{D\_Loop}(\widetilde{vd}, \mathit{end}) \mid \mathit{end}?().\emptyset)$

D_Loop $(\widetilde{d}, \mathit{end}) \triangleq$
$\underline{\text{if}}$ (**null** $\widetilde{d}$) $\underline{\text{then}}$ $\mathit{end}!\,\langle\rangle\,.\emptyset$
$\underline{\text{else}}$
$d_1!\,\langle\rangle\,.\text{D\_Loop}((\textbf{cdr}\ \widetilde{d}), \mathit{end})$

Cap_Loop $(\widetilde{l}, \mathit{vcb}, \mathit{end}) \triangleq$
$\underline{\text{if}}$ (**null** $\widetilde{l}$) $\underline{\text{then}}$ $\mathit{end}!\,\langle\rangle\,.\emptyset$
$\underline{\text{else}}$
$(\nu\, r)l_1!\,\langle r\rangle\,.r?(\widetilde{x}).\mathit{vcb}!\,\langle x1\rangle\,.$
Cap_Loop $\langle(\textbf{cdr}\ \widetilde{l}), \mathit{vcb}, \mathit{end}\rangle$

C_Loop $(n, \widetilde{c}, \mathit{vcb}, \mathit{end}, r) \triangleq$
$\mathit{vcb}?(cn).\text{C\_Loop}\,\langle n+1, \widetilde{c}, \mathit{vcb}, \mathit{end}, r\rangle$
$+\ \mathit{end}?().r!.\,\langle\widetilde{c}\rangle\,.\emptyset$

WhoBoss_Cell $(\widehat{\mathit{who\_boss}}, b) \triangleq$
$\mathit{who\_boss}_i?(x).\text{WhoBoss\_Cell}\,\langle\widehat{\mathit{who\_boss}}, x\rangle$
$+\ \mathit{who\_boss}_o?(s).s!\,\langle b\rangle\,.$
WhoBoss_Cell $\langle\widehat{\mathit{who\_boss}}, b\rangle$

Aclaraciones: Method_List, Group_List y Cap_List son listas con *in* y *out* para permitir agregar un elemento y también enviar listas. WhoBoss_Cell se utiliza para preguntar quién es el Surveyor de la *iniciativa*.

La variable colectiva $\tilde{vl}$ se utiliza también para abreviar, contiene las variables locales fac (la fachada), $\widehat{vmed}$ (la lista de mediadores), $\widehat{vind}$ (la lista de índices de los mediadores), $\widehat{vadj}$ (la lista de adjuntos de un mediador) y $\widehat{vdiem}$ (la lista para eliminar los mediadores).

Las variables *vcb* y *end* son las usuales para controlar las iteraciones en una estructura de datos. De manera similar ***cdr*** y ***car*** para obtener ciertos elementos de una lista.

A continuación se explica brevemente la especificación del patrón:

El proceso *AgentX* es invocado por el proceso *Creator_Agent*, que es su *"bootstrap"*. En su canal identidad recibe otro por el cual podrá enviar todos los parámetros que le corresponden. A continuación tiene el bucle usual para este tipo de procesos recursivos.

El proceso *Surveyor* también tiene características de ser el "arranque" para otros procesos. En él se crean todas las variables que son necesarias para la mayoría de las funciones involucradas en los demás patrones. Es decir que sus funciones son invocadas por los demás e intervienen en sus definiciones. Con Method_List incluye la lista de sus métodos $\tilde{m}$, en paralelo invoca al proceso recursivo *MySurveyor*. Asimismo también en paralelo se invoca el proceso para crear un planificador (Planner), conjuntamente con su canal de eliminación.

MySurveyor es el que realmente utiliza los métodos propios del patrón y que se explican a continuación. Este proceso, luego de ser invocado, a través del canal ret puede enviar la lista de métodos (meth_o).

Las funciones del surveyor vienen dadas por sus métodos. Para facilitar la identificación, y solo para abreviar, se los ha renombrado con **m** y un número. También se los ha incluido en un nombre colectivo, el vector $\tilde{m}$, para utilizarlo en las invocaciones de los procesos. La línea final de todos los métodos invoca la recursión de MySurveyor, con excepción del último (**m16**). Los métodos son:

{**m1**} add_member: cuando se recibe un dato en este canal con la función predefinida **add_list** se agrega un nuevo agente al grupo.

{**m2**} my_group: en este canal, la primera vez el surveyor recibirá la lista de los integrantes de su grupo, por un canal (r) se recibe la lista y por el otro (group_i) se envía.

{**m3**} who_group: por este canal se solicita el grupo, que se envía utlizando ret.

{**m4**} remove_member: al recibir el dato (x) pr ese canal, utilizando la función, también

predefinida **remove_list** se quita ese miembro del grupo.

{**m5**} what_list: cuando se invoca este método se recibe el canal ret. Con la función Cap_Loop se obtiene la lista de capacidades de los integrantes del grupo, C_Loop es una función auxiliar obtener ese resultado. Se ha incluido Cap_Loop también como un modelo de cómo se definirían funciones para tratar listas, siguiendo este formato se pueden predefinir otras, como por ejemplo **add_list** y similares. Por el canal ret se envía $\tilde{z}$ conteniendo el listado de funciones de los agentes.

{**m6**} who_facade: utilizado para responder quién es la fachada (ver Façade, 5.5.5).

{**m7**} create_facade: método para crear la fachada (ver 5.5.5).

{**m8**} kill_facade: se usa este método para eliminar la fachada (ver 5.5.5).

{**m9**} update_facade: se utiliza para actualizar la fachada (ver 5.5.6).

{**m10**} full_mesh: permite que otros agentes puedan comunicarse con la *iniciativa* (ver 5.5.11).

{**m11**} create_mediator: para crear un mediador (ver 5.5.9).

{**m12**} replace_mediator: usado para reemplazar un mediador (ver 5.5.8).

{**m13**} request_mediator: con este un agente externo solicita un mediador para interactuar con la *iniciativa* (ver 5.5.10).

{**m14**} generate_plan: utilizado para generar planificaciones (ver 5.5.14).

{**m15**} register_org: se lo utiliza para registrar una organización a partir de una *iniciativa* consolidada (ver 5.5.13).

{**m16**} para finalizar el surveyor. En el canal bye se espera recibir la orden de eliminación del surveyor, en ese caso también deben eliminarse la fachada, el planificador y todos los mediadores, esto último utilizando la función D_Loop que lo va haciendo uno tras otro hasta que no quede ninguno. Finalmente el surveyor se elimina.

5.5.4. Patrón *Change Surveyor*.

- Nombre del Patrón: **Change Surveyor**.
- Clasificación:
  - *Clasificación 1*: Reconfigurador (R), Estructural (S)

- *Clasificación 2*: CO-D (Coordinación Directa)

- Intención: lanza el protocolo para que el agente en el rol de Surveyor pueda ser cambiado por otro agente.

- Contexto: el patrón Change Surveyor puede ser utilizado cuando:

  - la *iniciativa* está en crecimiento
  - ya existe un agente surveyor, líder de la organización preliminar
  - el actual surveyor debe sustituirse por otro

- Motivación: la organización debe continuar emergiendo y se está utilizando el patrón Surveyor. Para ello el sistema debe poder realizar el cambio del agente surveyor. El patrón *Change Surveryor* describe el protocolo para degradar al actual y trasladar el conocimiento al próximo. Como causas del cambio se tienen, por ejemplo, agregar un nuevo agente más capacitado hasta el propio deseo del surveyor de abandonar su puesto. En caso de decidirse el cambio debe dispararse el proceso de elección.

- Esquema:

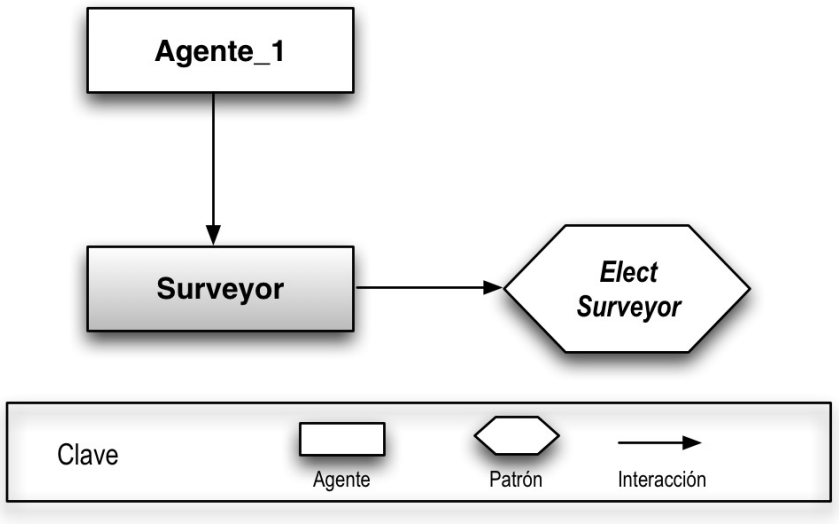

Figura 5.7: Esquema del patrón *Change Surveyor*.

- Participantes: utiliza dos (o más) agentes de tipo *changent*, uno de los cuales es el actual surveyor y el otro el nuevo agente que reemplazará al anterior. Agentes de otro tipo también pueden estar integrando la *iniciativa*.

- Comportamiento: el comportamiento se verá en detalle en el modelo de interacción hecho posteriormente en Cálculo-π. La Figura 5.8 presenta un ejemplo de comportamiento del patrón. El Agente plantea una *cuestión de confianza* para el cambio de Surveyor.

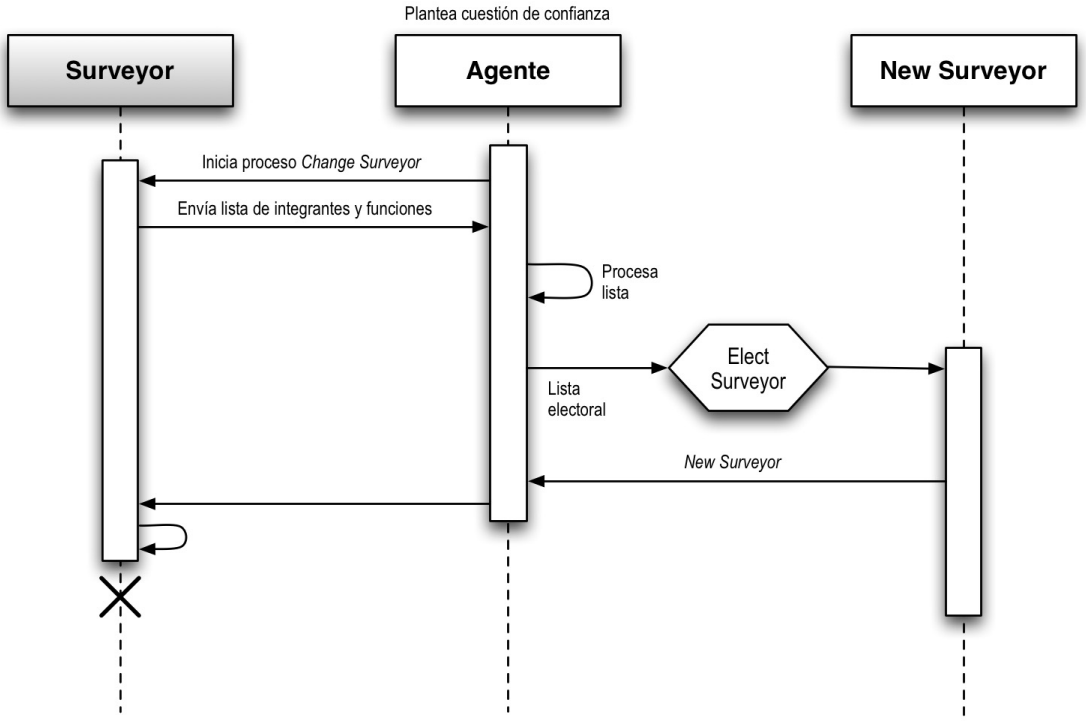

Figura 5.8: Ejemplo de comportamiento del patrón *Change Surveyor*.

- Consecuencias:

  - un nuevo agente surveyor reemplaza al anterior
  - el anterior surveyor traslada su conocimiento al nuevo
  - la *iniciativa* puede continuar su crecimiento

- Restricciones: Change Surveyor podrá ser utilizado cuando ya exista un surveyor liderando la *iniciativa*. No podrá ser utilizado a menos que se produzca un pedido de cambio de líder. Tampoco si los agentes no se conocen y carecen de la capacidad de conversar.

- Patrones relacionados: es necesario que Surveyor esté en uso para que posteriormente puede lanzarse el Change Surveyor, según sea necesario. Elect Surveyor puede lanzarse posteriormente para la elección de un nuevo surveyor en caso que el actual no sea reemplazado por un agente en particular. El patrón Monitor puede utilizarse en paralelo para capturar la interacción solicitando el cambio de surveyor.

Modelo de interacción en Cálculo-π.

AgentX $(dni, \textit{sys\_change}, \dots) \triangleq$

$\dots$

$+ \textit{who\_surveyor}?(surveyor).\textit{who\_surveyor}!\langle surveyor\rangle .$

$\tau .\textit{sys\_change}!\langle surveyor\rangle .$

AgentX $\langle dni, \textit{sys\_change}, \dots\rangle$

Confidence $(\textit{sys\_change}, \textit{sys\_vote}) \triangleq$

$\textit{sys\_change}?(surveyor).(\nu\, decide)$

(DECISION_ALGORITHM $\langle surveyor, decide\rangle$;

if $(decide = true)$ then

$(\nu\, ballot)surveyor!\langle\rangle .$

$surveyor?(\textit{meth}_o).$

$(\nu\, r)\textit{meth}_o!\langle r\rangle .r?(\widetilde{m}).$

$\textit{who\_group}!\langle\rangle .\textit{who\_group}?(\widehat{g}).$

$\textit{sys\_vote}!\langle ballot, \widehat{g}\rangle$);

Confidence $\langle \textit{sys\_change}, \textit{sys\_vote}\rangle$

El funcionamiento de este patrón puede verse como un cuestionamiento a la confianza

del actual Surveyor. Se pide un cambio de líder y la política para ese cambio puede ser externa al patrón.

Para realizar el cambio del Surveyor se utiliza el canal de sistema *sys_change*, que es otro parámetro del proceso *AgentX*. Por el canal who_surveyor del proceso se espera recibir al surveyor, seguidamente por ese canal también puede enviarse el dato del surveyor. Posterior a una función interna (τ), por el canal sys_change se enviará el (nuevo) surveyor. Seguidamente se invoca el bucle respectivo.

La cuestión de confianza que se mencionaba al comienzo estaría representada por el proceso Confidence, que recibe su parámetros respectivos. Por el canal sys_change se recibirá surveyor, con este y la variable *decide* puede utilizarse un algoritmo de decisión, ante una decisión positiva de cambio de surveyor se informa al actual su desplazamiento, se espera recibir la lista de métodos ($\tilde{m}$) en r y los integrantes de la *iniciativa* en who_group. Con estos integrantes se procede a realizar la elección del surveyor enviando *ballot* y $\hat{g}$ por el canal de sistema sys_vote. Seguidamente se fuerza la secuencialidad y se invoca nuevamente Confidence.

5.5.5. Patrón *Façade*.

- Nombre del Patrón: **Façade**.
- Clasificación:
  - *Clasificación 1*: Monitor (M), Estructural (S)
  - *Clasificación 2*: CM-EI (Comunicación Externa Indirecta)
- Intención: define un agente fachada (o agente "encargado") para llevar adelante las interacciones necesarias con el mundo exterior, por ejemplo con todos aquellos (agentes u organizaciones) que no pertenecen a la *iniciativa*.
- Contexto: el patrón Façade puede ser utilizado cuando:
  - la *iniciativa* continúa su crecimiento
  - es necesario interactuar con el exterior de la *iniciativa*

- se necesita un representante del grupo preliminar

- Motivación: se necesita generar interacción con el exterior de la organización preliminar. Reducir la complejidad minimizando la comunicación entre agentes, por lo que el agente fachada canaliza los mensajes hacia los agentes internos del grupo. Definir una vista simple de la organización al ser el agente fachada un único punto de acceso, lo que unificará, por ejemplo, servicios y mensajes. No es necesario que funcione también como supervisor o coordinador.

- Esquema: en la Figura 5.9 el *Agente Façade* representa a la *iniciativa* para facilitar la interacción con los agentes (o también sistemas) que no pertenecen a ella.

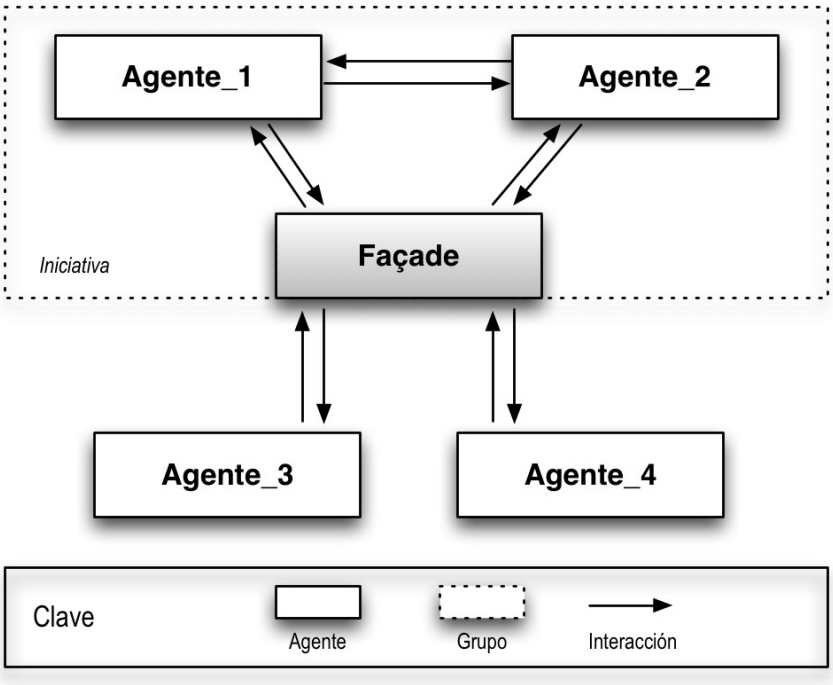

Figura 5.9: Esquema del patrón *Façade*.

- Participantes: este patrón utiliza un agente fachada, que representa a la *iniciativa*, canaliza los mensajes entrantes hacia los agentes correspondientes dentro del grupo, y hacia el exterior a otros agentes.

- Comportamiento: la Figura 5.10 es el diagrama de secuencias que representa las interacciones entre el agente fachada y los agentes *Agente_3* y *Agente_4*, que no forman parte de la *iniciativa*.

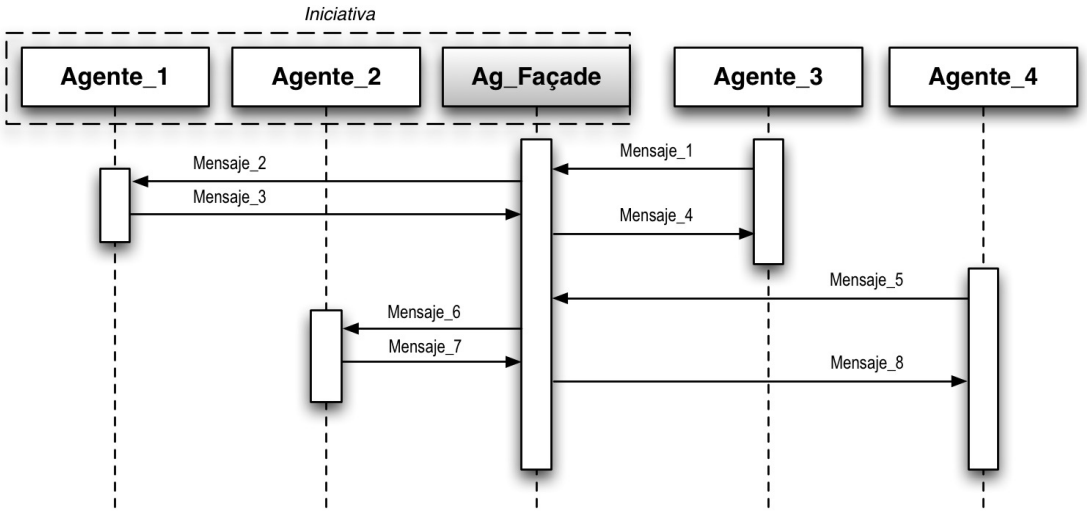

Figura 5.10: Ejemplo de comportamiento del patrón *Façade*.

- Consecuencias:
  - el agente fachada representa a la organización preliminar
  - puede canalizar la comunicación hacia el exterior por un solo punto de acceso
  - se comporta como interfaz única y simplificada de la *iniciativa*
  - permite concebir externamente al grupo preliminar como una entidad
  - facilita que no haya sobrecarga en la comunicación entre el exterior y los agentes que forman la *iniciativa*
- Restricciones: no se utilizará Façade a menos que sea necesario un representante del grupo preliminar para comunicaciones con el exterior de la *iniciativa*.
- Patrones relacionados: el patrón Surveyor está en funcionamiento y puede lanzar Façade con posterioridad según sea necesario. El patrón Update Façade puede ser lanzado posteriormente si fuera necesario una actualización del agente fachada. El

patrón Monitor puede utilizarse en paralelo para capturar la información que el agente fachada puede utilizar y a la que se ha suscripto.

Modelo de interacción en Cálculo-π.

Surveyor $(me) \triangleq$

$(\nu\ \ldots, fac, \textit{die\_fac}, \ldots, bye)$

$\ldots$

MySurveyor $(me, \ldots, fac, \textit{die\_fac}, \ldots, \widetilde{m}) \triangleq$

$me?(ret).ret!\ \langle \textit{meth}_o \rangle\ .$

MySurveyor $\langle me, \ldots, fac, \textit{die\_fac}, \ldots, \widetilde{m} \rangle$

$\{\mathbf{m6}\}\ +\ \textit{who\_facade}?().\textit{who\_facade}!\ \langle fac \rangle$

MySurveyor $\langle me, \ldots, fac, \textit{die\_fac}, \ldots, \widetilde{m} \rangle$

$\{\mathbf{m7}\}\ +\ \textit{create\_facade}?().((\nu\ \textit{ret\_list})$

$\textit{what\_list}!\ \langle \textit{ret\_list} \rangle\ .\textit{ret\_list}?(\widetilde{capa\_list}).$

$\tau\ (\widetilde{capa\_list}, \widetilde{exp\_list}).$

$(\nu\ \textit{die_fac})$Facade $\langle fac, \widetilde{exp\_list}, \textit{die\_fac} \rangle\ \mid$

MySurveyor $\langle me, \ldots, fac, \textit{die\_fac}, \ldots, \widetilde{m} \rangle\ )$

$\{\mathbf{m8}\}\ +\ \textit{kill\_facade}?().$

$\textit{die\_fac}!\ \langle\rangle\ .$

MySurveyor $\langle me, \ldots, fac, \textit{die\_fac}, \ldots, \widetilde{m} \rangle$

Facade $(me, \widetilde{clist}, bye) \triangleq$

$(\nu\ \widehat{cap}, \textit{my\_API})(\textit{Cap\_List}\ \langle \widehat{cap}, \widetilde{clist} \rangle\ \mid$

MyFacade $\langle me, \widehat{cap}, \textit{my\_API}, bye \rangle\ )$

$$
\begin{aligned}
&\text{MyFacade } (me, \widehat{cap}, \textit{my\_API}, bye) \triangleq \\
&\quad me?(ret).ret!\,\langle \textit{my\_API} \rangle\,. \\
&\quad \text{MyFacade } \langle me, \widehat{cap}, \textit{my\_API}, bye \rangle \\
&\quad + \textit{my\_API}?(ret). \\
&\quad (\nu\, r)\textit{cap}_o!\,\langle r \rangle\,.r?(\widetilde{cl}).ret!\,\langle \widetilde{cl} \rangle\,. \\
&\quad \text{MyFacade } \langle me, \widehat{cap}, \textit{my\_API}, bye \rangle \\
&\quad + bye?().\emptyset
\end{aligned}
$$

El patrón Façade hace uso de tres métodos del patrón Surveyor.

El método **m6** o *who_facade* es utilizado para preguntarle al Surveyor quién es la fachada de la *iniciativa* y se responde con ese dato por la misma interfaz. A continuación se invoca el bucle MySurveyor.

El segundo método, **m7** o *create_facade* se utiliza para crear la fachada. La creación puede ser lanzada tanto por el Surveyor como también ser solicitada por un tercero. Como parte del método se solicita al Surveyor la lista de capacidades de los integrantes de la *iniciativa*; la respuesta la envía el método **m5** del Surveyor. De esta última se asume la creación de la lista de funciones a exportar por la fachada. Seguidamente se invoca el proceso de creación propiamente dicho. En esta invocación se incluye un canal de "finalización" para utilizarlo en el momento adecuado. En paralelo, aún cuando la fachada se está creando, el Surveyor continúa funcionando con su versión recursiva.

El tercer método es **m8** o *kill_facade*, que utiliza una variable (die_fac) para enviar una orden de eliminación de la fachada que será llevada adelante por el Surveyor.

El proceso Facade, que tiene la forma de un *"bootstrap"* como en otros patrones, recibe sus parámetros correspondientes a su identidad, la lista de elementos de la fachada y el utilizado para su eliminación. Se crean variables para la lista de capacidades y la interfaz completa de la *iniciativa*. Seguidamente se crea la lista de capacidades y se lanza el bucle MyFacade.

El proceso MyFacade, lanzado en Facade y con sus respectivos parámetros, espera re-

cibir en su canal identidad uno por el cual enviar la interfaz (my_API) y a continuación se lanza el bucle MyFacade.

Con my_API primero se pide el envío de la lista de capacidades por un canal (r), por ese canal se espera recibir dicha lista que posteriormente se enviará por otro canal (ret). Seguidamente el bucle recursivo.

Por último, con bye se solicitará la finalización de la fachada.

5.5.6. Patrón *Update Façade*.

- Nombre del Patrón: **Update Façade**.
- Clasificación:
  - *Clasificación 1*: Monitor (M), Estructural (S)
  - *Clasificación 2*: CM-EI (Comunicación Externa Indirecta)
- Intención: permite realizar el relevo de las responsabilidades del actual agente con el rol de fachada. Dichas responsabilidades se asignarán al nuevo agente fachada.
- Contexto: el patrón Update Façade puede ser utilizado cuando:
  - la *iniciativa* continúa su crecimiento
  - existe un representante del grupo preliminar
  - es necesario relevar al agente fachada actual
- Motivación: la *iniciativa* cuenta con un representante para las comunicaciones con el exterior. Es necesario que el agente fachada actual sea relevado, por ejemplo, si debe abandonar el lugar donde está asignado. El patrón facilita el traspaso de las responsabilidades al nuevo agente fachada.
- Esquema: la Figura 5.11 presenta el esquema de las interacciones entre un Agente y el Surveyor para que haya un cambio de rol fachada.
- Participantes: participan un agente surveyor, un agente fachada y al menos otro agente, el que podrá reemplazar al fachada actual.

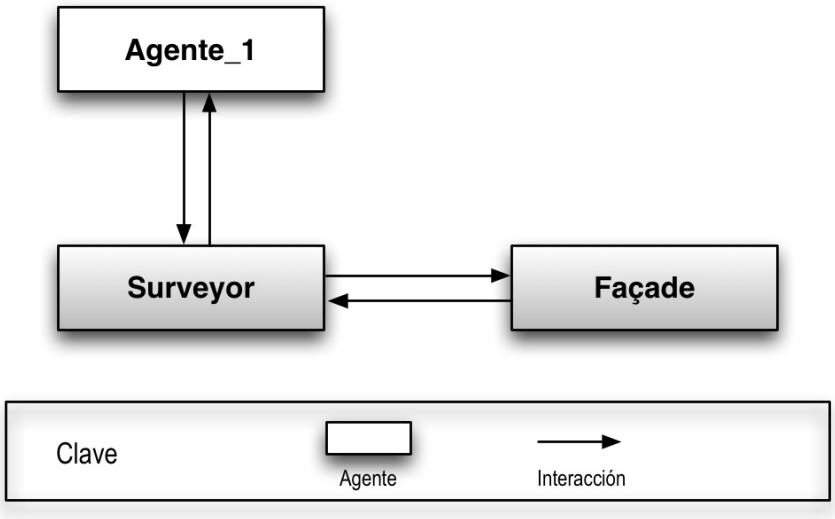

Figura 5.11: Esquema del patrón *Update Façade*.

- Comportamiento: la Figura 5.12 representa un diagrama de secuencia del patrón. Ante determinadas circunstancias, un agente solicita el cambio de la fachada. El agente surveyor analiza la solicitud y selecciona al nuevo Façade. Este recibe la lista actualizada de integrantes y acepta las nuevas responsabilidades. El anterior agente fachada queda liberado del rol pudiendo continuar (o no) su trabajo en la *iniciativa*.

- Consecuencias:

  - la *iniciativa* tiene nuevo representante.
  - el anterior agente fachada es relevado del cargo

- Restricciones: el patrón no podrá ser utilizado a menos que sea necesario un relevo del representante de la *iniciativa*. Tampoco podrá ser utilizado si la *iniciativa* no cuenta con un agente fachada.

- Patrones relacionados: Façade es utilizado con anterioridad para que a un agente de la *iniciativa* le sea asignado el rol de fachada. Con el patrón Surveyor se facilita tanto la selección del agente fachada como su relevo. El patrón Monitor puede utilizarse en paralelo para capturar la interacción solicitando la actualización del agente fachada.

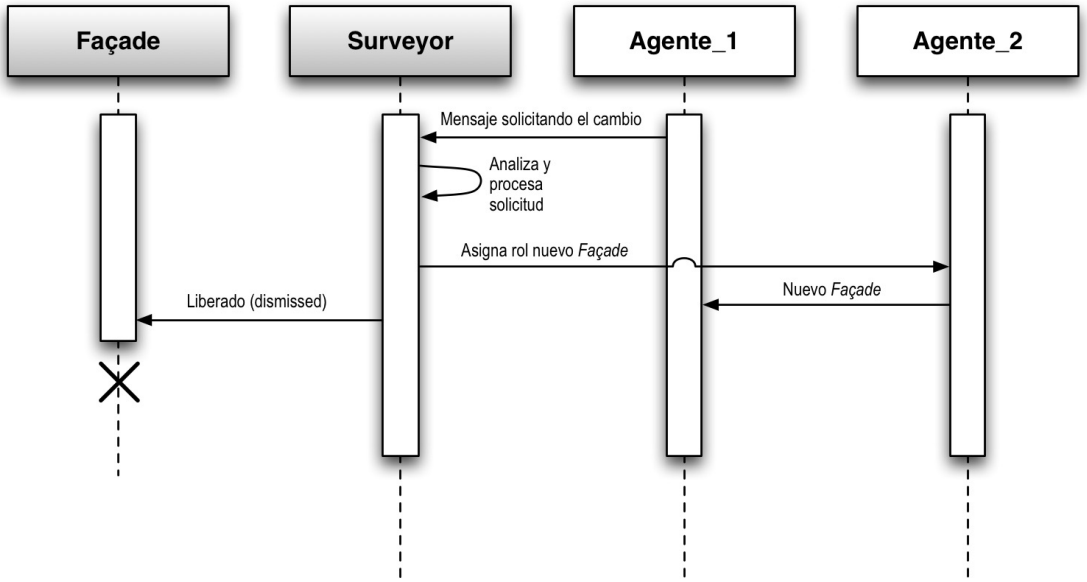

Figura 5.12: Ejemplo de comportamiento del patrón *Update Façade*.

Modelo de interacción en Cálculo-π.

$$\text{MySurveyor}\ (me, \widehat{meth}, \widehat{group}, \widetilde{vl}, \textit{die\_fac}, \ldots, \widetilde{m}) \triangleq$$
$$me?(ret).ret!\ \langle \textit{meth}_o \rangle\ .$$
$$\text{MySurveyor}\ \langle me, \widehat{meth}, \widehat{group}, \widetilde{vl}, \textit{die\_fac}, \ldots, \widetilde{m} \rangle$$

$$\{\mathbf{m9}\}\ + \textit{update\_facade}?().($$
$$\textit{kill\_facade}!\ \langle\rangle\ .\textit{create\_facade}!\ \langle\rangle\ \mid$$
$$\text{MySurveyor}\ \langle me, \widehat{meth}, \widehat{group}, \widetilde{vl}, \textit{die\_fac}, \ldots, \widetilde{m} \rangle\ )$$

$$+ \ldots$$

Se asume que el Surveyor es el encargado de realizar la actualización (*update*) de la fachada de la *iniciativa* a solicitud de un tercero.

Se utiliza el método **m9** o *update_facade* del Surveyor. Este método recibe la solicitud de *update* y a continuación se eliminará la anterior fachada con el método **m8** (kill_facade). Seguidamente se envía la orden para la creación de una nueva, método **m7** (create_facade). En paralelo se invoca el proceso recursivo MySurveyor para que Surveyor pueda continuar funcionando aún cuando la nueva fachada se esté creando.

5.5.7. Patrón *Mediator*.

- Nombre del Patrón: **Mediator**.
- Clasificación:
  - *Clasificación 1*: Reconfigurador (R), de Comportamiento (B)
  - *Clasificación 2*: CM-II (Comunicación Interna Indirecta)
- Intención: define a un agente como intermediario interno, que funciona dentro de la *iniciativa*. Facilita la interacción y la coordinación entre los agentes integrantes del grupo preliminar.
- Contexto: Mediator puede ser utilizado cuando:
  - la *iniciativa* continúa su crecimiento
  - existe mucha heterogeneidad entre los agentes integrantes
  - las interdependencias aún no están totalmente estructuradas
  - los servicios de datos no funcionen adecuadamente
  - los servicios de alto nivel aún no están siendo provistos
  - es necesario concentrar las comunicaciones internas y desde el exterior
- Motivación: la *iniciativa* aún no se ha consolidado como organización plena. Es necesario que un elemento intermediario concentre las comunicaciones internas de forma que facilite la interacción entre agentes que de otra manera no podrían comunicarse. De igual manera con agentes desde afuera hacia adentro del grupo preliminar. El patrón Mediator facilita la designación de un agente como mediador interno a la

iniciativa. Además el agente facilitaría las traducciones semánticas necesarias ante la heterogeneidad de agentes.

- Esquema: en la Figura 5.13 se representa al *Agente Mediator*, que actúa como intermediario entre los agentes de la *iniciativa*, y también hacia el exterior. Asimismo tiene la posibilidad de acceder al Surveyor.

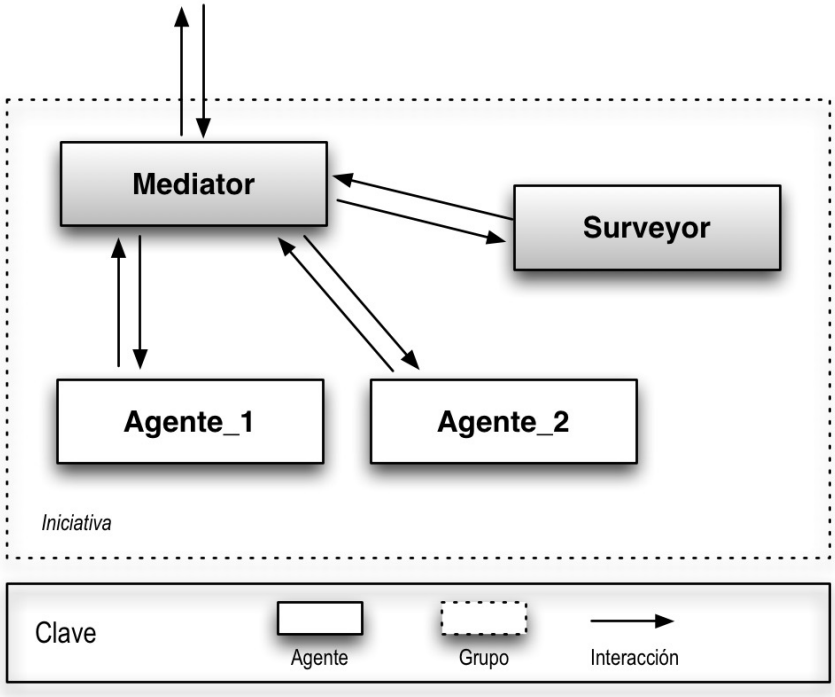

Figura 5.13: Esquema del patrón *Mediator*.

- Participantes: el patrón Mediator utiliza un agente de tipo mediador, funciona como intermediario interno y también con el exterior de la *iniciativa*. Facilita el comportamiento cooperativo, conoce a los demás agentes de la *iniciativa* y les permite comunicarse a través de él realizando las interpretaciones o traducciones necesarias. También participan uno (o más) agentes de otro tipo, los que conocen al mediador, proveen sus propios servicios a medida que van integrando el grupo preliminar, se comunican entre ellos a través del mediador.

- Comportamiento: la Figura 5.14 representa un ejemplo de comportamiento e interacciones entre el mediador y dos agentes, todos dentro de la *iniciativa*, que de otra forma no podrían comunicarse. Asimismo el mediador puede acceder al agente líder.

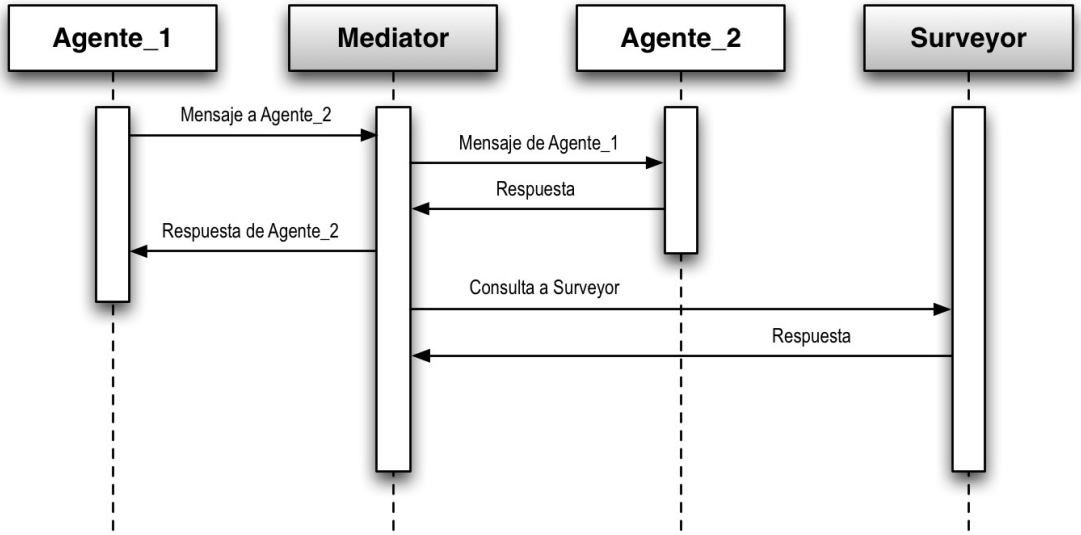

Figura 5.14: Ejemplo de comportamiento del patrón *Mediator*.

- Consecuencias:
  - permite que la *iniciativa* pueda crecer.
  - permite la comunicación interna entre agentes heterogéneos.
  - permite canalizar la comunicación con el exterior de la *iniciativa*.
  - posibilita el acceso a las fuentes de datos/servicios de datos en formación.
  - simplificar la interacción con entidades externas.
  - permite realizar traducciones semánticas.
  - facilita el trabajo cooperativo entre los integrantes del grupo preliminar.
- Restricciones: no será necesario utilizar Mediator cuando todos los agentes de la *iniciativa* se conozcan y tengan la capacidad de conversar sin necesitar intérprete o traductor. Tampoco sería necesario si no se precisaran interacciones con el exterior.

- Patrones relacionados: el patrón Surveyor es el encargado de seleccionar al agente mediador, en base a sus habilidades, conocimiento y capacidades. Con el patrón Assign Mediator se asignará un mediador (ya existiendo un actual) de entre los agentes de la *iniciativa*, por ejemplo por poseer mayores conocimientos. Con el patrón New Mediator seleccionará un nuevo mediador, por ejemplo, ante la inexistencia de uno. El patrón Thru Mediator permite establecer la conexión de un agente con la *iniciativa*, posiblemente desconectado y que necesite traducción.

Modelo de interacción en Cálculo-π.

Mediator $(me, iniciativa, \textbf{\textit{who\_adj}}, bye) \triangleq$

$(\nu\, \widehat{adj}, forward)$

MyMediator $\langle me, iniciativa, \widehat{adj}, \textbf{\textit{who\_adj}}, forward, bye\rangle$

MyMediator $(me, \ldots, bye) \triangleq$

$me?(\textbf{\textit{new\_adj}}, ret).ret!\,\langle forward\rangle\,.$

$\textbf{\textit{add\_list}}\,\langle \widehat{adj}, \textbf{\textit{new\_adj}}\rangle\,.$

MyMediator $\langle me, bye\rangle$

$+\ \textbf{\textit{who\_adj}}?(ret).ret!\,\langle \widehat{adj}\rangle\,.$

MyMediator $\langle me, bye\rangle$

$+\ forward?(who, what, ret).$

$(\nu\, r1)iniciativa!\,\langle r1\rangle\,.r_1?(\widehat{\textbf{\textit{ini\_list}}}).$

$(\nu\, destino)\textsc{Select\_Med}(who, what, \widehat{\textbf{\textit{ini\_list}}}, destino).$

$(\nu\, inter)destino!\,\langle what, inter\rangle\,.$

$(inter?(respuesta).ret!\,\langle respuesta\rangle\ \mid$

MyMediator $\langle me, \ldots, bye\rangle\,)$

$+\ bye?().\emptyset$

$$\text{Adjunct\_List } (\widehat{adj}, \widetilde{c}) \triangleq$$
$$\quad \textit{adj}_i?\,\langle x\rangle\,.\text{Adjunct\_List}\,\langle \widehat{adj}, x\rangle$$
$$\quad + \textit{adj}_o?(r).r!\,\langle \widetilde{c}\rangle\,.\text{Adjunct\_List}\,\langle \widehat{adj}, \widetilde{c}\rangle$$

Aclaración: el patrón Mediator utiliza dos listas (arrays) como estructuras de datos internas que son administradas por el Surveyor. Se tratan de *vmed*, la lista de los mediadores (ya que pueden ser varios), y *vind*, la lista de índices con los que se indica lo que realiza cada mediador. Se utilizan los índices para acceder a los mediadores.

El mediador funciona como un intermediario entre integrantes de la *iniciativa* que no pueden interactuar directamente. Tiene que ser capaz de recibir las peticiones, en base a estas se realizará la búsqueda en la lista e invocar al mediador correspondiente. Recibe la lista de la *iniciativa* cada vez que algún integrante necesita su mediación para verificar si se han producido cambios en ella.

A continuación se describen brevemente las interacciones.

Este patrón, al igual que otros, también utiliza un proceso como "arranque": el Mediator, que luego lanza el proceso MyMediator que es el que mantiene el bucle (la recursión).

Los parámetros del proceso Mediator son me, iniciativa, who_adj y bye; la segunda (iniciativa) se corresponde con el método **m3** o *who_group* del Surveyor que se usa para pedir la lista de integrantes. Como es usual se crean las variables locales necesarias ($\widehat{adj}$ y forward), la primera es la lista de "adjuntos" o elementos que van a ser intermediados y la última es el método para solicitar la intermediación.

A continuación se lanza MyMediator con sus respectivos parámetros. Es invocado y recibe al nuevo adjunto y su canal (ret) por donde realizará los pedidos de intermediación utilizando el correspondiente método. Con **add_list** se añade el nuevo adjunto a la lista de adjuntos (como ya se ha explicado, esta función agrega un elemento a una lista o array). Seguidamente se continúa el bucle.

Con who_adj se responde a la pregunta de quiénes son los adjuntos de ese mediador.

El método para la intermediación es forward. Se recibe la identidad del que necesita un mediador, el dato y su canal de respuesta. Se crea una variable local (r1), un canal privado, por el que se recoge la lista de la *iniciativa*. Seguidamente se crea el canal destino y se invoca el proceso SELECT_MED. Este proceso, en base a quién lo pidió y qué ha pedido, selecciona entre los integrantes del grupo el destino a dónde enviar el dato (what). En realidad SELECT_MED puede ser más complejo de lo que se expone, habría que analizarlo con más detalle pero para este contexto lo importante es su objetivo. Seguidamente se envía un mensaje a destino (what, inter) y se espera en el canal inter la respuesta. Una vez recibida se la reenvía. Para una nueva petición se realizará todo el proceso nuevamente. En paralelo se lanza MyMediator con sus parámetros. Esto último a efectos de que si el elemento remoto no responde en un tiempo razonable, el mediador siga intermediando a otros.

Con bye se recibe la petición para que el mediador sea eliminado.

Adjunct_List representa la lista de canales para solicitar a un mediador su lista de adjuntos o miembros de la *iniciativa* con los que hace de mediador.

5.5.8. Patrón *Assign Mediator*.

- Nombre del Patrón: **Assign Mediator**.
- Clasificación:
  - *Clasificación 1*: Reconfigurador (R), de Comportamiento (B)
  - *Clasificación 2*: CM-II (Comunicación Interna Indirecta)
- Intención: el actual mediador de la *iniciativa* debe ser cambiado por otro elemento del grupo que presenta mayores habilidades y conocimientos, por lo que se lanza el protocolo para que el nuevo sea escogido entre los candidatos potenciales.
- Contexto: el patrón Assign Mediator puede ser utilizado cuando:
  - la *iniciativa* continúa su crecimiento

- existe un agente mediador dentro del grupo preliminar
- un agente dentro del grupo cuenta con mayores habilidades, conocimientos y/o capacidades que el actual mediador
- es necesario un mediador con mayores capacidades ante cambios del entorno

- Motivación: el grupo preliminar está en crecimiento y aún no se ha consolidado. Se cuenta con la presencia de un agente mediador, pero ante la necesidad de nuevas capacidades se asigna el rol de mediador a otro agente que cumple con las condiciones requeridas de conocimiento y habilidades ante cambios en el entorno.

- Esquema: en la Figura 5.15 se esquematizan las interacciones entre un Agente que solicita al Surveyor una reasignación del rol mediador. Surveyor posibilita que un nuevo mediador sea asignado e informa al antiguo del cambio.

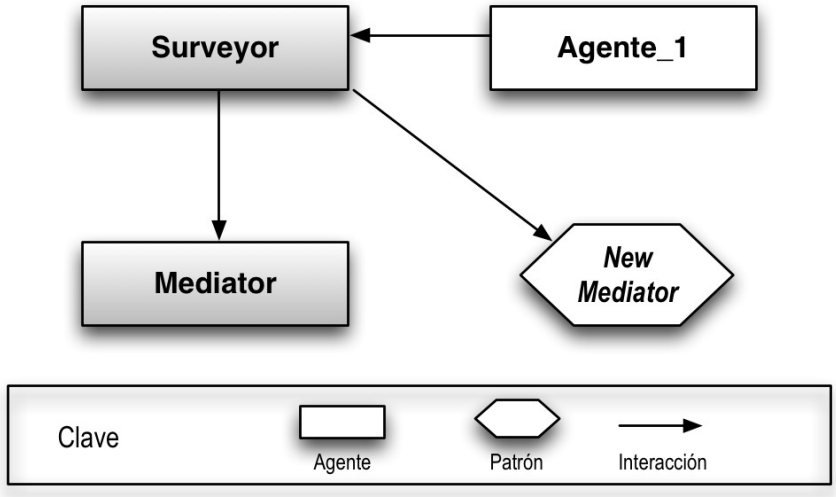

Figura 5.15: Esquema del patrón *Assign Mediator*.

- Participantes: en el patrón participan un agente surveyor, un agente mediador y al menos otro agente con mayores capacidades que el mediador actual.

- Comportamiento: el comportamiento detallado puede verse posteriormente en la especificación del modelo de interacción en Cálculo-π. En la Figura 5.16 se tiene un

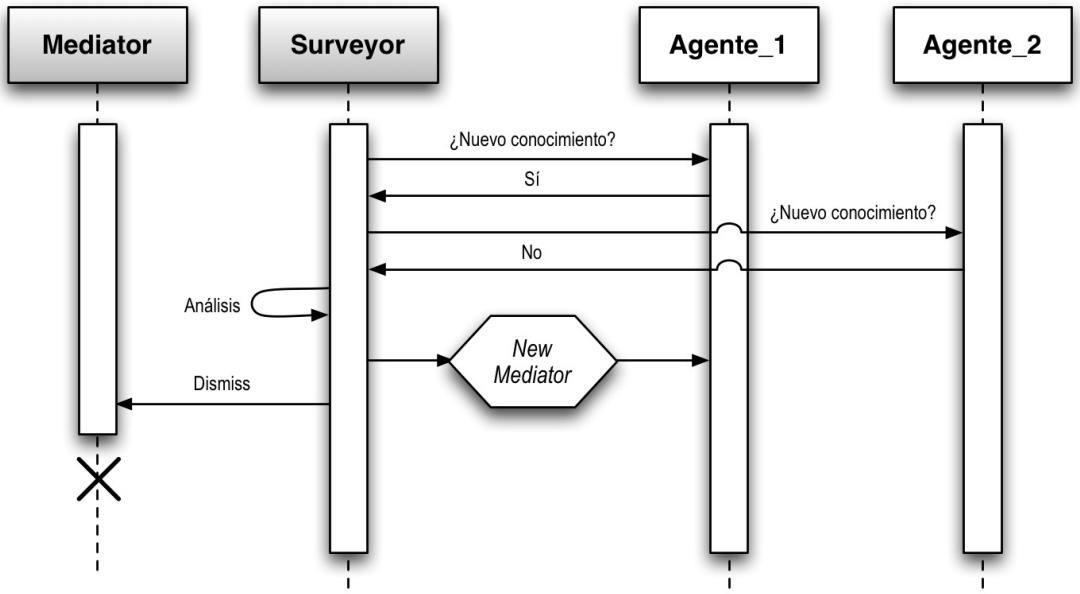

Figura 5.16: Ejemplo de comportamiento del patrón *Assign Mediator*.

diagrama de secuencia de un ejemplo de comportamiento del patrón. El agente surveyor, ante la necesidad de nuevas habilidades para el mediador, pregunta a dos agentes respecto de sus conocimientos. Determina que el Agente_1 es el adecuado y es elegido para asignarle el rol de agente mediador.

- Consecuencias:
  - el actual agente mediador deja de serlo
  - se asigna el rol de mediador a un agente con mayores capacidades
  - la *iniciativa* aumenta su capacidad de interacción
- Restricciones: el patrón no será utilizado a menos que sea necesario un agente mediador más capacitado como respuesta a los cambios del entorno y que exista uno con esas características.
- Patrones relacionados: el patrón Surveyor se utiliza en paralelo, el agente surveyor asignará el rol de mediador al agente correspondiente. El patrón Mediator también es utilizado en paralelo teniendo en cuenta que el agente mediador ya existía y ahora

debe ser cambiado. El patrón Monitor puede utilizarse en paralelo para capturar la interacción solicitando el cambio de surveyor.

Modelo de interacción en Cálculo-π.

$\{\mathbf{m12}\}\ + \textit{replace\_mediator}?(suitor, dat).$
$\tau\,(dat, index).(\nu\, i)\textbf{\textit{.search\_list}}\,\langle\widehat{vind}, index, i\rangle\,.$
$(\nu\, r)\textit{vadj}_o!\,\langle r\rangle\,.r?(\widetilde{lw}).(\nu\, s)\textit{lw}_i!\,\langle s\rangle\,.s?(\widetilde{la}).$
$(\nu\, \textit{die\_suitor}, \textit{adj\_suitor})$
(Mediator $\langle suitor, \textit{who\_group}, \textit{adj\_suitor}, \textit{die\_suitor}\rangle$
$|\ (\nu\, end)$(A_Loop $\langle suitor, \widetilde{la}, end\rangle$
$|\ end?().$
$(\nu\, t)\textit{vdiem}_o!\,\langle t\rangle\,.t?(\widetilde{ldm}).$
$\textit{ldm}_i!\,\langle\rangle\,.$
$\textbf{\textit{insert\_list}}\,\langle\widehat{vmed}, suitor, i\rangle\,.$
$\textbf{\textit{insert\_list}}\,\langle\widehat{vadj}, \textit{adj\_suitor}, i\rangle\,.$
$\textbf{\textit{insert\_list}}\,\langle\widehat{vdiem}, \textit{die\_suitor}, i\rangle\,.$
MySurveyor $\langle me, \ldots\rangle\,))$

A_Loop $(suitor, \widetilde{la}, end) \triangleq$
if (**null** $\widetilde{la}$) then $end!\,\langle\rangle\,.\emptyset$
else $(\nu\, ret)(la_1!\,\langle ret\rangle\,.ret?(\widetilde{x}).$
$x_4!\,\langle suitor\rangle\,.$
A_Loop $\langle suitor, (\textbf{cdr}\ \widetilde{la}), end\rangle\,)$

Para este patrón se utiliza el método **m12** o *replace_mediator* del patrón Surveyor. Se solicita el reemplazo del mediador por alguna razón importante, como por ejemplo que el nuevo candidato tenga mayores conocimientos en ontologías.

Se invoca el proceso *replace_mediator* con suitor (el reemplazador) y dat (el motivo del reemplazo). La invocación la puede realizar el propio agente que quiere reemplazar a un mediador. Con el resultado de la función interna (τ) se utiliza la función **search_list** para buscar en la lista de índices la posición del mediador (el lugar i). "index" tiene el carácter de etiqueta de contenido y sirve para ubicar el índice que se corresponde con los datos de petición.

Con **search_list** se asume una función de búsqueda convencional en listas o arrays.

Después la búsqueda se recibirán las listas de los adjuntos de los mediadores ($\widetilde{lw}$), la lista de adjuntos del mediador i ($\widetilde{lw_i}$) y la lista de adjuntos del antiguo mediador ($\widetilde{la}$). Seguidamente se invoca al proceso Mediator con los parámetros correspondientes para crear al nuevo mediador con el candidato (suitor), entre ellos su canal de eliminación (die_suitor) y el canal para responder a la pregunta de quiénes son los adjuntos de ese mediador (adj_suitor).

Antes de eliminar al antiguo mediador hay que reasignar los adjuntos al nuevo, el objetivo es que Surveyor pueda saber quién es el nuevo y a quiénes afecta. Con A_Loop se logra esa reasignación de adjuntos al nuevo (suitor). Una vez hecho esto se está en condiciones de eliminar al antiguo, se recibe la lista de canales de "muerte" de los mediadores ($\widetilde{ldm}$) y se envía la orden en el canal correspondiente ($\widetilde{ldm_i}$).

Con la función **insert_list** se inserta a suitor en la lista de mediadores en la posición i. De nuevo, esta es una función convencional de inserción de un elemento en una lista o array. De igual manera se insertan las variables adj_suitor en $\widehat{vadj}$ (la lista de canales para pedirle a un mediator su lista de adjuntos) y die_suitor en la propia.

Seguidamente el bucle de MySurveyor como es usual.

Con la función A_Loop se reasignan los adjuntos en el nuevo mediador (suitor). Se le informa a cada adjunto quién es ahora el nuevo mediador. En el canal $\widetilde{la_1}$ (correspondiente al adjunto actual, que es el dni de un *AgentX*) se recibe el vector $\tilde{x}$ y por **x4** (o *ch_med* del agente) se envía el suitor al agente para que ya puedan interactuar. La explicación de $\tilde{x}$ y **x4** puede verse en la especificación el patrón Elect Surveyor 5.5.2.

5.5.9. Patrón *New Mediator*.

- Nombre del Patrón: **New Mediator**.

- Clasificación:
  - *Clasificación 1*: Reconfigurador (R), de Comportamiento (B)
  - *Clasificación 2*: CM-II (Comunicación Interna Indirecta)
- Intención: este patrón lanza la elección de un nuevo agente mediador ante el hecho de que el actual debe renunciar a su rol o ha dejado de funcionar como tal, por ejemplo por la necesidad de abandonar el grupo preliminar.
- Contexto: el patrón New Mediator puede ser utilizado cuando:
  - la *iniciativa* continúa su crecimiento
  - existe un agente mediador dentro del grupo preliminar
  - el actual agente mediador debe renunciar a su rol
  - el actual agente mediador ha dejado de funcionar como tal
  - el actual agente mediador debe abandonar el grupo preliminar
  - el grupo preliminar no cuenta con un agente mediador
- Motivación: para continuar con el crecimiento y emergencia de la nueva organización el sistema necesita un nuevo elemento mediador para poder acceder, por ejemplo a fuentes de datos internas, y realizar las traducciones necesarias para sus integrantes. O también porque el grupo preliminar ha perdido (o perderá) a su actual mediador por lo que es necesario que otro agente, con los servicios mínimos para funcionar como tal, deba ser seleccionado como nuevo agente mediador.
- Esquema: en la Figura 5.17 se esquematizan las interacciones en el caso de un agente que solicita un nuevo mediador al Surveyor, este invoca a uno que ya puede interactuar con el primer agente.
- Participantes: utiliza un agente surveyor y dos (o más) agentes. Es posible también que participe otro agente en el papel de antiguo mediador en caso de que la *iniciativa* ya cuente con uno. El patrón lanza el protocolo correspondiente para elegir el nuevo mediador.

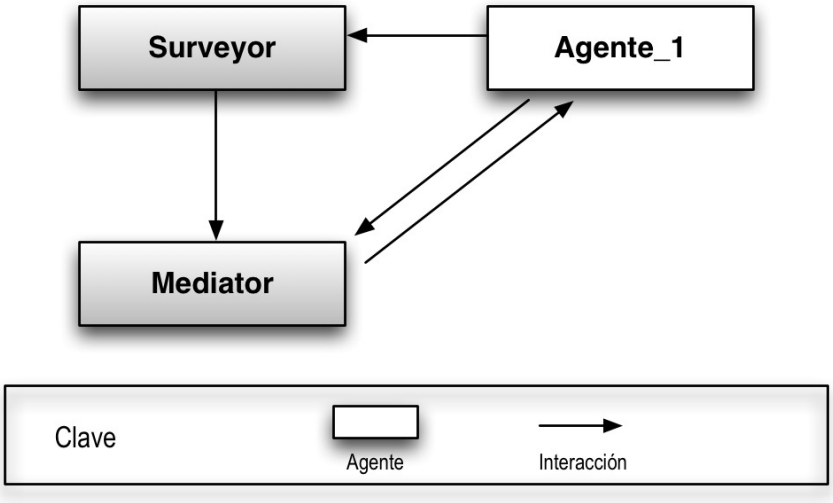

Figura 5.17: Esquema del patrón *New Mediator*.

- Comportamiento: en la Figura 5.18 se puede ver el diagrama de secuencia para el caso en el que el actual agente mediador debe abandonar su rol en el grupo preliminar. El agente surveyor pregunta a los otros agentes sobres los servicios mínimos para funcionar como mediador. El Agente_2 cumple con las condiciones y es seleccionado como tal. El antiguo mediador es liberado de su rol.

- Consecuencias:

  - permite elegir un nuevo mediador en caso de no existir uno
  - permite dirigir el cambio del mediador antiguo al nuevo agente mediador
  - se actualiza la lista con los datos de los agentes de la *iniciativa*
  - se informa a todos los agentes del nuevo mediador

- Restricciones: el patrón no será utilizado cuando todos los agentes de la *iniciativa* estén en condiciones de conversar e interactuar por sí mismos y sin traducciones. Tampoco cuando no sea necesario interactuar con el exterior.

- Patrones relacionados: el patrón Surveyor se utiliza en paralelo, el agente surveyor elige al agente que cuente con los servicios mínimos para funcionar como mediador

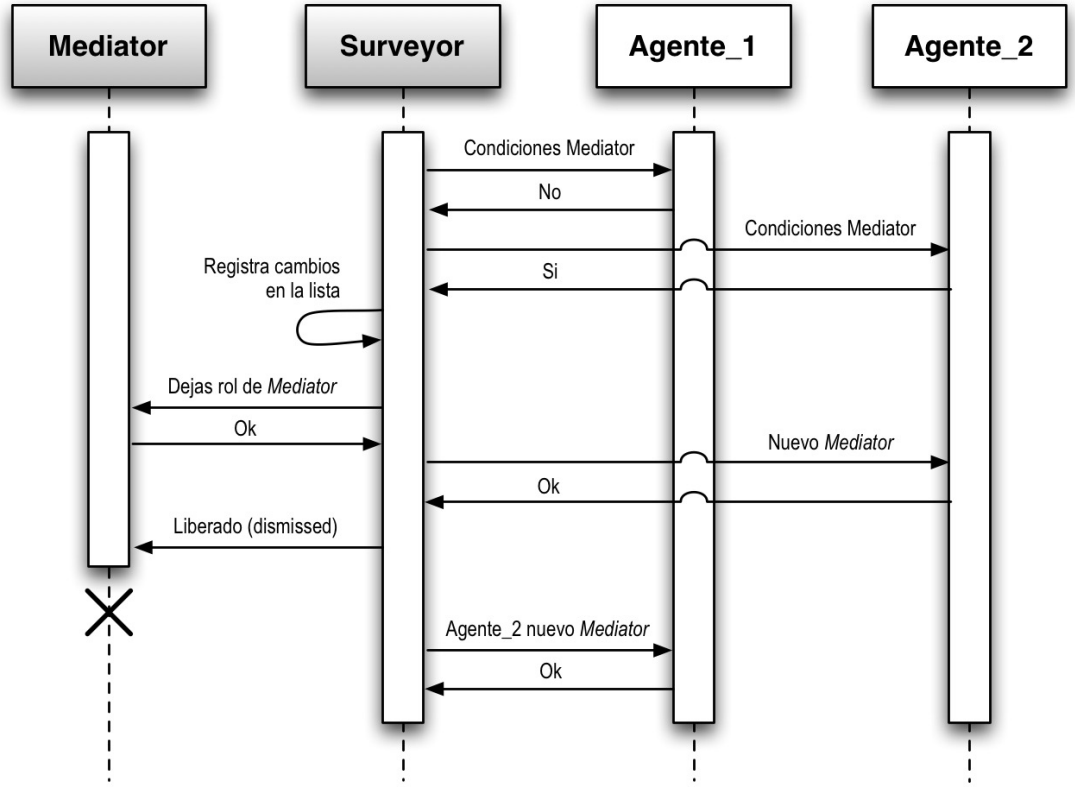

Figura 5.18: Ejemplo de comportamiento del patrón *New Mediator*.

y le asigna ese rol. El patrón Mediator también es utilizado en paralelo en caso de existir un agente mediador que dejará de funcionar como tal. El patrón Monitor puede utilizarse en paralelo para capturar la interacción solicitando el cambio de agente mediador.

Modelo de interacción en Cálculo-π.

Surveyor $(me) \triangleq$

$(\nu\, \widehat{meth}, \ldots, \widetilde{vl}, \ldots, \widetilde{m})$

$\ldots$

$$\text{MySurveyor}\ (me, \widehat{meth}, \ldots, \widetilde{m}) \triangleq$$
$$me?(ret)\ldots$$

$$+\ldots$$

$$\{\mathbf{m11}\}\ + \textit{create\_mediator}?(dat)$$
$$(\nu\ \textit{new\_med}, \textit{adj\_med}, \textit{die\_med})$$
$$(\textbf{\textit{add\_list}}\ \langle \widehat{vmed}, \textit{new\_med}\rangle\ .\tau\ (dat, index).$$
$$\textbf{\textit{add\_list}}\ \langle \widehat{vind}, index\rangle\ .$$
$$\textbf{\textit{add\_list}}\ \langle \widehat{vadj}, \textit{adj\_med}\rangle\ .$$
$$\textbf{\textit{add\_list}}\ \langle \widehat{vdiem}, \widehat{\textit{die\_med}}\rangle\ .$$
$$\text{Mediator}\ \langle \textit{new\_med}, \textit{who\_group}, \textit{adj\_med}, \textit{die\_med}\rangle\ )\ |$$
$$\text{MySurveyor}\ \langle me, \widehat{meth}, \ldots.\rangle$$

En este patrón se hace uso de cuatro listas: Mediator_List, Index_List, Adjunct_List y Diem_List, lista de mediadores, lista de índices, lista de adjuntos y lista de canales de "muerte" de los mediadores, respectivamente.

Se utiliza el método **m11** o *create_mediator* del Surveyor y el parámetro dat representa el motivo por el que se necesita crear el mediador. Se crean las variables necesarias para este método: new_med para un mediador, adj_med para el que va a ser mediado, y die_med para eliminar mediadores.

Como ya se ha explicado anteriormente, **add_list** se asume que trabaja con listas o arrays indistintamente. Con esta función se crean los mediadores y se agregan a la lista ($\widehat{vmed}$, new_med). La función τ es una acción interna del sistema, no es necesario colocarle parámetros, pero los incluimos solo para indicar que el índice que se genera para el nuevo mediador se puede calcular a partir de dat.

De manera similar, con **add_list** se agregan los índices, los adjuntos y los canales de muerte de los mediadores a sus respectivas listas ($\widehat{vind}$, $\widehat{vadj}$, $\widehat{vdiem}$).

A continuación se invoca al proceso Mediator con sus parámetros: new_med, who_group, adj_med y die_med. Como se considera que el Mediator no va a necesitar saber quién es el Surveyor que lo ha creado, solo tiene que estar informado de su creación y de cómo pedir la lista de integrantes. Por lo tanto recibirá el canal para pedir esa lista, que se corresponde con el método **m3** (who_group) del Surveyor.

En paralelo se tiene al proceso MySurveyor, necesario para el bucle como en los otros métodos del patrón Surveyor.

5.5.10. Patrón *Thru Mediator*.

- Nombre del Patrón: **Thru Mediator**.
- Clasificación:
  - *Clasificación 1*: Reconfigurador (R), de Comportamiento (B)
  - *Clasificación 2*: UN-I (Unión Indirecta)
- Intención: este patrón establece la conexión entre un mediador ya determinado y un nuevo agente, posiblemente desconectado, para que pueda interactuar con la *iniciativa*, o también para ingresar en ella.
- Contexto: el patrón Thru Mediator puede ser utilizado cuando:
  - la *iniciativa* continúa con su crecimiento
  - existe un agente mediador en el grupo preliminar
  - un nuevo agente, posiblemente desconectado, precisa comunicarse/interactuar con la *iniciativa*
  - un nuevo agente exterior precisa integrar el grupo preliminar
  - el nuevo agente exterior precisa traducciones para comunicarse/interactuar con la *iniciativa*
- Motivación: el grupo preliminar no se ha establecido aún como organización plena. Un nuevo agente precisa comunicarse con agentes de la *iniciativa* pero necesita

hacerlo a traves de un intermediario, el agente mediador. El patrón Thru Mediator establece esa conexión proveyendo al nuevo agente con el potencial para esas interacciones, y de esa manera le facilita el ingreso a la *iniciativa*.

- Esquema: la Figura 5.19 representa a un agente externo que puede interactuar con el mediador y de esa manera accede a los elementos de la *iniciativa*.

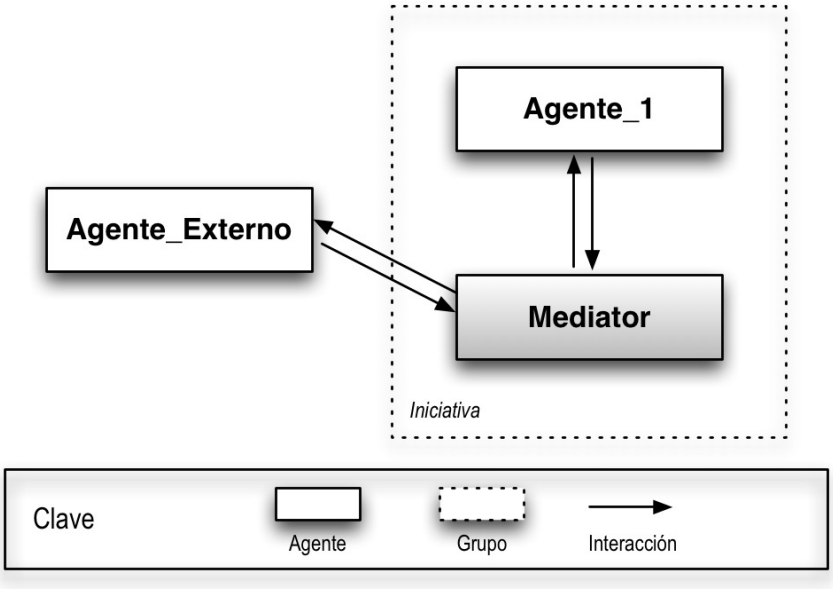

Figura 5.19: Esquema del patrón *Thru Mediator*.

- Participantes: participan un agente mediador y un agente externo. El mediador facilita la interacción entre el nuevo agente y los integrantes de la *iniciativa*.

- Comportamiento: en la Figura 5.20 puede verse el diagrama de secuencia en el caso de un agente externo que precisa interactuar con un agente (Agente_1) que pertenece a la *iniciativa*. Lo hace a traves del agente mediador, el que se encarga de realizar las traducciones semánticas necesarias.

- Consecuencias:

  - el agente externo puede interactuar con agentes del grupo preliminar

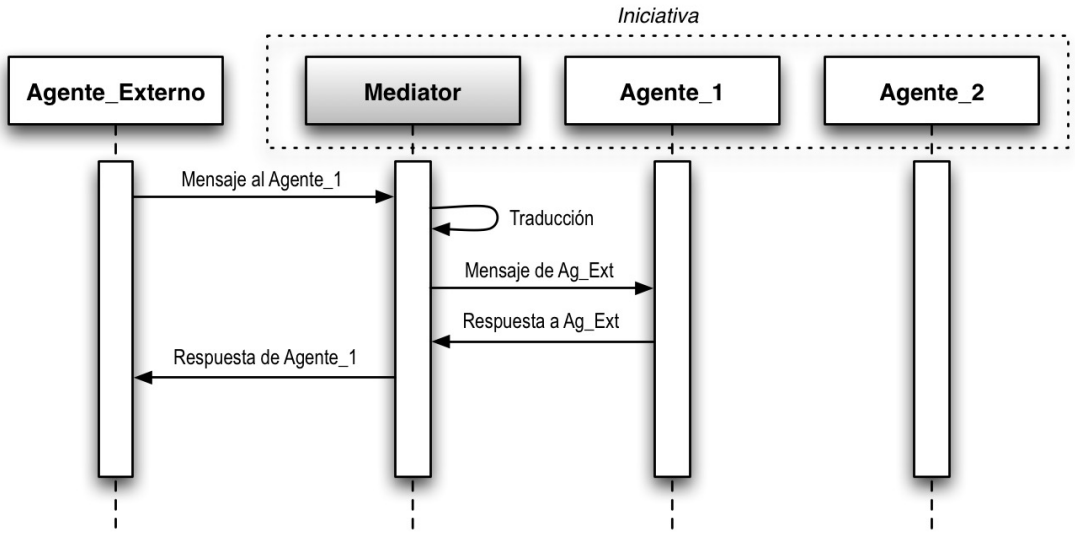

Figura 5.20: Ejemplo de comportamiento del patrón *Thru Mediator*.

- el agente externo estaría en condiciones de unirse a la *iniciativa*
- el agente externo utiliza al agente mediador para interactuar con el grupo preliminar

- Restricciones: este patrón no será utilizado a menos que exista un agente externo que necesite interactuar con el grupo preliminar y precise de un agente mediador para ello.

- Patrones relacionados: el patrón Mediator está funcionando en paralelo porque es necesario que el agente mediador permita la interacción del grupo preliminar con un agente externo. El patrón Monitor puede utilizarse en paralelo para capturar la interacción solicitando el ingreso del nuevo agente.

Modelo de interacción en Cálculo-π.

Surveyor $(me) \triangleq$
$(\nu\, \widehat{meth}, \ldots, \widetilde{m}) \ldots$
MySurveyor$(me, \widehat{meth}, \ldots, \widetilde{m})$

MySurveyor $(me, \widehat{meth}, \ldots, \widetilde{m}) \triangleq$

$me?(ret)$

$\ldots$

$\{\mathbf{m13}\}$ $+$ ***request_mediator***$?(what, ret).$

$\tau\,(what, index).$

$(\nu\, i)$.***search_list*** $\langle \widehat{vind}, index, i \rangle$.

$(\nu\, r)$***vmed***$_o!\, \langle r \rangle$ $.r?(\widetilde{lm}).$

$ret!\, \langle lmi \rangle$.

MySurveyor $\langle me, \ldots \rangle$

AgentX $(dni, try,$ ***no_med***$,$ ***ch_med***$, log) \triangleq$

$(\nu\ \ldots)\ldots$

$+\, try?($***request_here***$).\tau\,(what).$

$(\nu\, ret)($***request_here***$!\, \langle what, ret \rangle$.

$ret?(mediator).$

$(\nu\, r)mediator!\, \langle dni, r \rangle$ $.r?(send).$

$(\nu$ ***die_adj***$)($***no_med***$?($***bye_old***$).$***bye_old***$!\, \langle\rangle$.

no_med$!\, \langle$***die_adj***$\rangle$ $|$

Adjunct $\langle dni, send,$ ***die_adj***$\rangle$ $|$

AgentX $\langle dni, try,$ ***no_med***$,$ ***ch_med***$, log\rangle$)

$+$ ***ch_med***$?(mediator).$

$(\nu\, r)mediator!\, \langle dni, r \rangle$ $.r?(send).$

$(\nu$ ***die_adj***$)($***no_med***$?($***bye_old***$).$***bye_old***$!\, \langle\rangle$.

no_med$!\, \langle$***die_adj***$\rangle$ $|$

Adjunct $\langle dni, send,$ ***die_adj***$\rangle$ $|$

$Log!\, \langle$**String** $($*"ch mediator"*$, suitor,$ *"agent"*$, dni)\rangle$.

AgentX $\langle dni, try,$ ***no_med***$,$ ***ch_med***$, Log\rangle$)

$$
\begin{aligned}
&\text{Adjunct } (dni, send, bye) \triangleq \\
&\quad \tau\,(datos).(\nu\, ret). \\
&\quad send!\,\langle dni, datos, ret\rangle\,. \\
&\quad ret?(answer). \\
&\quad \text{Adjunct } \langle dni, send, bye\rangle \\
&\quad + bye?().\emptyset
\end{aligned}
$$

Este patrón utiliza el método **m13** o *request_mediator* del Surveyor. Básicamente, ante la petición de un agente exterior se le recomienda un mediador. Al invocarse **m13** se recibe el motivo de la solicitud del mediador (por ejemplo un especialista en un tema) y el canal para la respuesta. Una función interna devuelve index, el que utiliza la función **search_list** para buscar en la lista de índices la posición del mediador (i) correspondiente. Se responde con el mediador identificado (lm_i). Seguidamente el bucle de MySurveyor.

El proceso *AgentX* recibe *try* y *no_med* (en el principio no se tiene un mediador) como parámetros, variables que han sido creadas entre varias otras en su *"bootstrap"*: *Creator_Agent* (ver 5.5.2). Cuando un agente solicita mediador, se le pasa try y en él espera recibir un canal (request_here), que en realidad es el **m13** del Surveyor. El proceso continúa con el tema (what) y un canal de respuesta, por el que se recibe el mediador propuesto (el canal *mediator*). Esta parte sincroniza con *me* del proceso MyMediator. Seguidamente el agente externo se identifica como adjunto a ese mediador, el que le responde con un canal (send), el que se corresponde con el forward del mediador y por el que podrá enviar sus peticiones.

Se continúa con un sub-proceso, dentro de *AgentX*, que gestiona todo el proceso de comunicación con el que actualmente es el mediador, hasta que un nuevo mediador es asignado a ese adjunto. En ese momento se elimina el antiguo adjunto a traves del canal no_med. A continuación en ese canal se guarda el canal de "muerte" del adjunto en cuestión. En paralelo se invoca Adjunct, y también nuevamente *AgentX*.

Con ch_med se espera recibir al mediador y así se le dice al adjunto quién es su nuevo mediador. A continuación se realizan pasos similares que anteriormente con try. La dife-

rencia está en la introducción de un *Log*, que corresponde a información que puede ser utilizada por el patrón Monitor (ver 5.5.16), ya que el cambio de mediador es importante.

Adjunct tiene un ciclo de comportamiento, después de un tiempo variable se envía al nuevo adjunto y por el canal send la información para que pueda conversar con la *iniciativa* y recibe las respuestas por el canal ret. Con bye se termina la recursión.

5.5.11. Patrón *Full Mesh*.

- Nombre del Patrón: **Full Mesh**.
- Clasificación:
  - *Clasificación 1*: Reconfigurador (R), de Comportamiento (B)
  - *Clasificación 2*: UN-D (Unión Directa), CM-ID (Comunicación Interna Directa)
- Intención: el patrón provee una conexión de tipo *full-mesh*, es decir "todos-con-todos", entre un agente externo que ingresa en la *iniciativa* y el resto de sus integrantes.
- Contexto: el patrón Full Mesh puede ser utilizado cuando:
  - la *iniciativa* continúa su crecimiento
  - un agente externo precisa ingresar en el grupo preliminar
- Motivación: el grupo preliminar todavía no se ha establecido como organización plena. Un agente externo necesita ingresar en la *iniciativa* por lo que el patrón provee una conexión apropiada con todos los integrantes del grupo preliminar. El nuevo agente tiene la capacidad para conectarse directamente con todos, sin requerir un mediador.
- Esquema: en la Figura 5.21 puede verse como un agente externo puede interactuar directamente con los integrantes de la *iniciativa*, en este caso con el Surveyor y con el Agente_1.

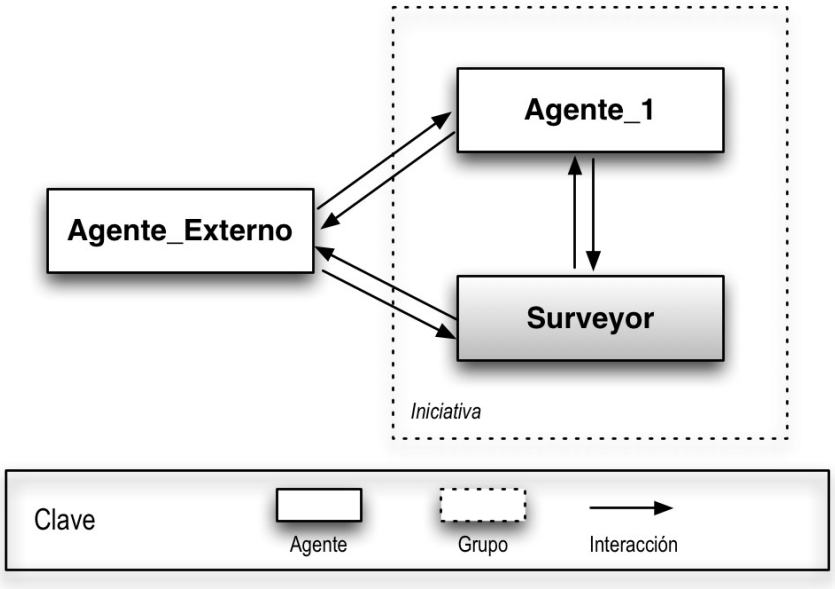

Figura 5.21: Esquema del patrón *Full Mesh*.

- Participantes: un agente surveyor que recibirá la solicitud de un agente externo para formar parte de la *iniciativa*.
- Comportamiento: la Figura 5.22 presenta el caso de las interacciones entre un agente surveyor (dentro de la *iniciativa*) y un agente externo que precisa integrarse al grupo preliminar. Este último envía el mensaje con la solicitud al surveyor, el que posteriormente le responde con la lista de integrantes actualizada para que pueda conectarse directamente con todos.
- Consecuencias:
  - uno o más agentes del exterior pueden formar parte del grupo preliminar
  - facilita la interacción entre los agentes de la *iniciativa* con los nuevos
  - se incrementa la lista de integrantes del grupo
- Restricciones: no se utilizará este patrón a menos que uno o más agentes externos soliciten formar parte del grupo preliminar. Solo se utilizará si los agentes del exterior tienen la capacidad para conectarse con todos los integrantes de la *iniciativa*.

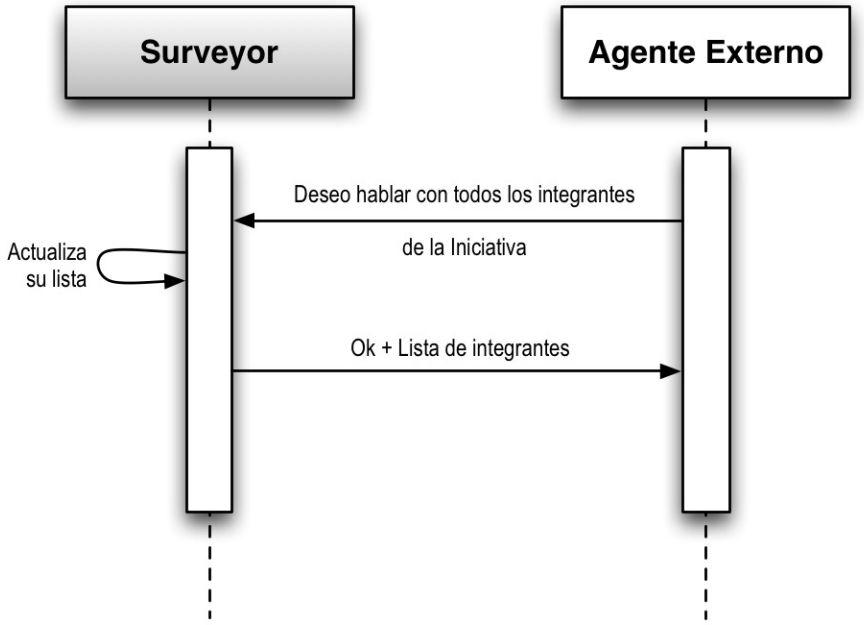

Figura 5.22: Ejemplo de comportamiento del patrón *Full Mesh*.

- Patrones relacionados: el patrón Surveyor funciona en paralelo porque el agente surveyor debe facilitar el ingreso del o los agentes del exterior. El patrón Monitor puede utilizarse en paralelo para capturar la interacción solicitando el ingreso de los nuevos agentes.

Modelo de interacción en Cálculo-π.

$$
\begin{aligned}
&\text{AgentX}\ (\ldots, chat, \ldots) \triangleq \\
&\qquad + chat?(\widetilde{l}).\tau\,. \\
&\qquad * [lx!\,\langle data\rangle \mid ly?(info)] \\
&\qquad \text{AgentX}\,\langle \ldots, chat, \ldots \rangle
\end{aligned}
$$

$$\text{MySurveyor}\ (me, \widehat{meth}, \widehat{group}, \ldots, (\ldots, \textit{full\_mesh}, \ldots)) \triangleq$$

$$\ldots$$

$$\{\mathbf{m10}\}\ +\textit{full\_mesh}?().$$

$$(\nu\, r)\textit{group}_o!\,\langle r\rangle\,.r?(\widetilde{l}).$$

$$(\nu\, end)(\text{Full\_Loop}\ \langle \widetilde{l}, \widetilde{l}, end\rangle$$

$$\mid end?().$$

$$\text{MySurveyor}\ \langle me, \widehat{meth}, \widehat{group}, \ldots\rangle\ )$$

$$\text{Full\_Loop}\ (\widetilde{l}, \widetilde{full}, end) \triangleq$$

$$\underline{\text{if}}\ (\mathbf{null}\ \widetilde{l})\ \underline{\text{then}}\ end!\,\langle\rangle$$

$$\underline{\text{else}}\ l_1!\,\langle\rangle\,.l_1?(chat).$$

$$chat!\,\langle\widetilde{full}\rangle\,.$$

$$\text{Full\_Loop}\ \langle(\mathbf{cdr}\ \widetilde{l}), \widetilde{full}, end\rangle$$

Para este patrón el proceso *AgentX* debe ser invocado con el parámetro *chat*, además de todos los otros que correspondan. En ese canal (chat) se espera recibir la lista de integrantes de la *iniciativa*. Seguidamente el sub-proceso repetitivo (denotado por $*$) significa que los agentes pueden interactuar e intercambiar datos (data e info).

Se utiliza el método **m10** o *full_mesh* del patrón Surveyor. Se invoca ese método, se recibe la lista de agentes y utilizando la función Full_Loop los agentes estarán habilitados para conversar sin la necesidad de intermediarios. A continuación se invoca a MySurveyor como es usual en estos casos.

Con la función Full_Loop cada agente involucrado (l_1) espera recibir un canal por donde conversar, ese canal es chat y por él se le envía la lista de los demás participantes.

5.5.12. Patrón *Retirement*.

- Nombre del Patrón: **Retirement**.
- Clasificación:
  - *Clasificación 1*: Reconfigurador (R), Estructural (S)

- *Clasificación 2*: SE (Separación)

- Intención: el patrón describe el proceso que debe seguir un agente para abandonar la *iniciativa*.

- Contexto: el patrón Retirement puede ser utilizado cuando:

  - la *iniciativa* continúa su crecimiento
  - un agente integrante del grupo preliminar debe abandonarlo

- Motivación: el grupo preliminar aún no se ha establecido como organización plena. Todo agente integrante de la *iniciativa* que deba abandonarla, por decisión propia o por otras razones, debe seguir el proceso de este patrón. De esta manera recupera su papel de "agente libre".

- Esquema: en este esquema, Figura 5.23, un agente abandona la *iniciativa*, lo que puede ocurrir por propia solicitud o por indicación del Surveyor.

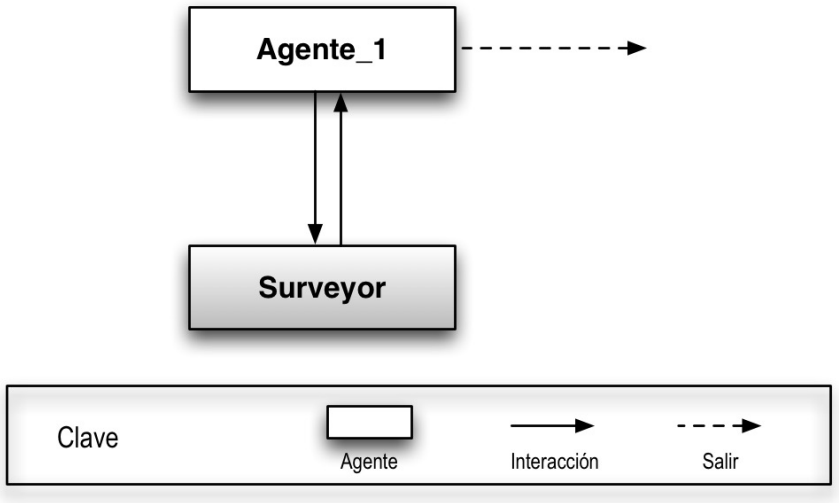

Figura 5.23: Esquema del patrón *Retirement*.

- Participantes: un agente surveyor y los agentes integrantes de la *iniciativa*.

- Comportamiento: en el caso de la Figura 5.24 puede verse un diagrama de secuencia que representa la salida del Agente_1 de la *iniciativa*. Para ello informa de este hecho al agente surveyor, el que posteriormente actualiza la lista de integrantes y libera al agente que abandona el grupo preliminar.

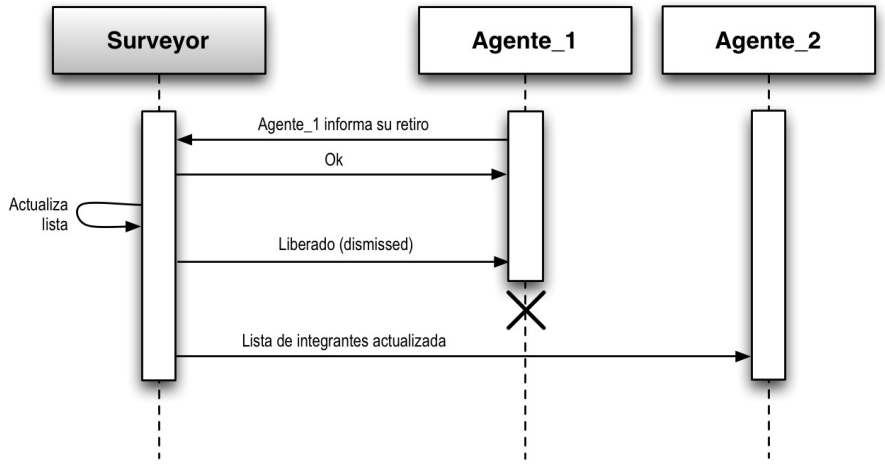

Figura 5.24: Ejemplo de comportamiento del patrón *Retirement*.

- Consecuencias:
  - la *iniciativa* pierde uno (o más) integrantes
  - puede modificarse el servicio de alto nivel que representaba el objetivo buscado de la organización
- Restricciones: este patrón no será utilizado a menos que haya algún agente integrante de la *iniciativa* que deba abandonar el grupo.
- Patrones relacionados: el patrón Surveyor funciona en paralelo porque el agente surveyor debe resolver el retiro del o los agentes que lo han solicitado o que deben abandonar el grupo.

Modelo de interacción en Cálculo-π.

$$
\begin{array}{l}
\text{AgentX}\ (dni, \ldots) \triangleq \\
\quad \ldots \\
\quad + \textit{who\_surveyor}?(surveyor).\textit{who\_surveyor}!\ \langle surveyor \rangle\ . \\
\quad surveyor!\ \langle\rangle\ .surveyor?(\textit{meth}_o). \\
\quad (\nu\, r)\textit{meth}_o!\ \langle r \rangle\ .r?(\widetilde{m}). \\
\quad\quad m_4!\ \langle dni \rangle\ . \\
\quad\quad \text{AgentX}\ \langle dni, \ldots \rangle
\end{array}
$$

El agente que tiene acoplado el proceso *AgentX* debe abandonar la *iniciativa*, por lo tanto el proceso MySurveyor recibirá el identificador (dni) del agente que debe salir del grupo.

Utilizando el método **m4**, que se corresponde con *remove_member*, procederá a quitarlo de la lista de la *iniciativa*.

5.5.13. Patrón *Consolider*.

- Nombre del Patrón: **Consolider**.
- Clasificación:
  - *Clasificación 1*: Decisor (DM), Creador (C)
  - *Clasificación 2*: CR-O (Creación Organización)
- Intención: este patrón describe el proceso por el cual se consolida y se registra una *iniciativa*, convirtiéndola en una organización completa o de pleno derecho.
- Contexto: el patrón Consolider puede utilizarse cuando:
  - no es necesario que la *iniciativa* continúe su crecimiento
  - el objetivo final de la organización puede ser alcanzado
  - todos los agentes integrantes de la organización han llegado al *acuerdo* necesario para resolver el problema

- Motivación: la *iniciativa* ha finalizado su crecimiento. La estructura del grupo se ha vuelto "estable" debido a que todos los integrantes han llegado al *acuerdo* necesario para resolver el problema o lograr el objetivo principal que había sido la causa de la unión inicial. La *organización* resultante posteriormente se registra como una unidad y es conceptualmente similar a las organizaciones de numerosos enfoques de SMA.
- Esquema: la Figura 5.25 presenta el caso de una *iniciativa* que se consolida como organización, por lo tanto el Surveyor debe realizar su registro, dando por finalizado también al grupo preliminar.

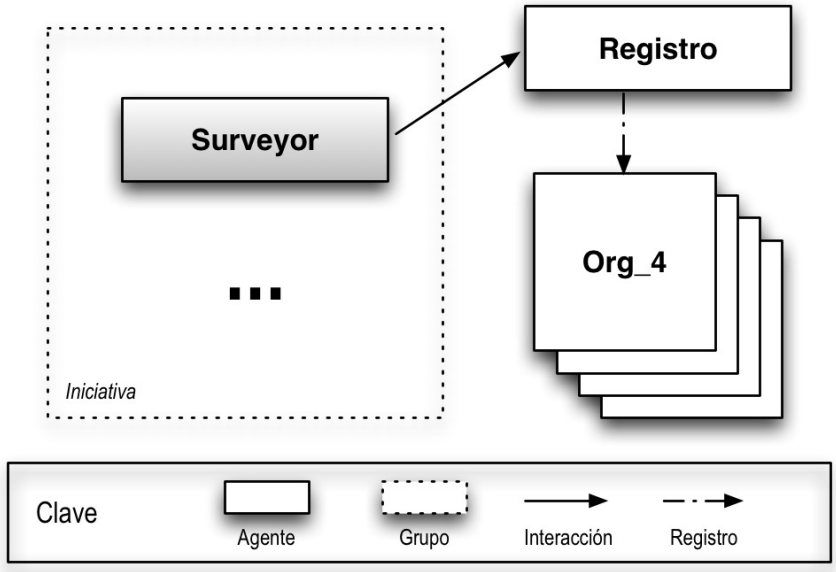

Figura 5.25: Esquema del patrón *Consolider*.

- Participantes: un agente surveyor más todos los agentes integrantes de la *iniciativa* que han llegado al *acuerdo* para formar la organización que llevará adelante la resolución del problema.
- Comportamiento: la Figura 5.26 presenta un diagrama de secuencia en el que una *iniciativa* es registrada como una organización plena. El agente surveyor informa a los agentes que pasan a formar parte de la organización consolidada y la *iniciativa* se da por finalizada.

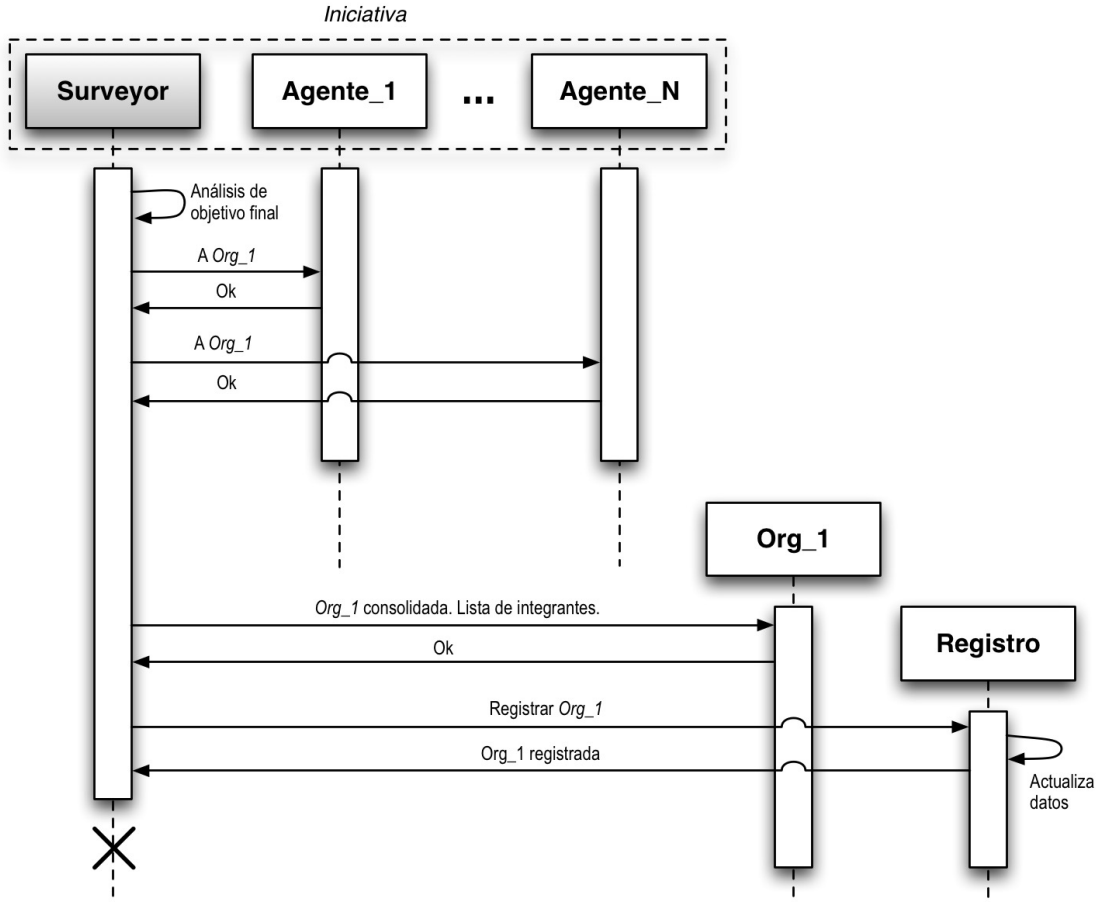

Figura 5.26: Ejemplo de comportamiento del patrón *Consolider*.

- Consecuencias:
  - la *iniciativa* ha detenido su crecimiento
  - se ha llegado al *acuerdo* entre todos los integrantes
  - se ha formado una *organización basada en acuerdos*
  - la organización ha sido registrada como organización plena
  - la organización ha alcanzado una estructura estable
  - la organización está en condiciones de lograr el objetivo principal, o resolver el problema original

- Restricciones: no se podrá utilizar el patrón si la *iniciativa* aún está en crecimiento. Tampoco se podrá utilizar si el objetivo principal no se puede alcanzar, o si no es posible conseguir el *acuerdo* entre los agentes integrantes.

- Patrones relacionados: el patrón Surveyor está funcionando en paralelo, el agente surveyor es el responsable de decidir la consolidación de la *iniciativa* en una organización de pleno derecho. Podrá participar en el disparo de Terminator en caso de no consolidación de la *iniciativa*; o en caso de consolidarse la organización y precisarse la finalización de todos los proceso del grupo preliminar.

Modelo de interacción en Cálculo-π.

Surveyor $(me) \triangleq$
$(\nu\ \ldots, \textit{register\_org}, \ldots)$
$(\textit{Method\_List}\ \langle \widehat{meth}, \widetilde{m} \rangle\ \mid$
MySurveyor $\langle me, \widehat{meth}, \widehat{group}, \ldots, \widetilde{m} \rangle$

MySurveyor $(me, \widehat{meth}, \widehat{group}, \ldots, \textit{register\_org}, \ldots) \triangleq$
$\ldots$
$+\ \textit{register\_org}?().$
$(\nu\, r)\textit{group}_o!\ \langle r \rangle\ .r?(\widetilde{l}).$
$\textit{sys\_registry}!\ \langle \widetilde{l} \rangle\ .\textit{bye}!\ \langle\rangle\ .$
MySurveyor $\langle me, \widehat{meth}, \widehat{group}, \ldots, \widetilde{m} \rangle$
$+ \ldots$

Dentro del proceso Surveyor la variable register_org debe haberse creado en el *"bootstrap"*, esa variable se utilizará para registrar la organización que se ha consolidado a partir de la *iniciativa*.

Se invoca al proceso MySurveyor incluyéndose esa variable entre sus parámetros. Su método **my** es el que se utiliza para registrar las organizaciones. Con la lista de los integrantes de la *iniciativa* y usando la variable de sistema *sys_registry* se registra a la organizacion,

posteriormente se invoca el proceso *bye* de MySurveyor para dar por finalizada la *iniciativa*. Cómo se toma la decisión para consolidar la *iniciativa* es una cuestión política y externa al patrón.

5.5.14. Patrón *Planner*.

- Nombre del Patrón: **Planner**.
- Clasificación:
  - *Clasificación 1*: Decisor (DM), Estructural (S)
  - *Clasificación 2*: CO-P (Coordinación Planificada)
- Intención: posibilita la planificación de una respuesta inteligente ante determinada situación. El agente encargado es un planificador estandard que debe tener en cuenta los recursos que se encuentran presentes en la *iniciativa*.
- Contexto: el patrón Planner puede ser utilizado cuando:
  - la *iniciativa* continúa su crecimiento
  - es necesario planificar determinadas acciones o cambios de acuerdo a la dinámica del entorno
- Motivación: la organización preliminar continúa en su crecimiento. En cierto punto del ciclo de vida de la *iniciativa* es necesaria una planificación de acciones y/o cambios. La planificación preparada para ser aplicada debe tener en cuenta los recursos que se encuentran presentes en el grupo preliminar. Ante la descripción de la situación el agente planificador retornará un plan que debería resolverla. Un clásico ciclo *MAPE*[2] puede ser de utilidad en este caso.
- Esquema: este esquema es un caso especial porque se plantea el trabajo en conjunto de los patrones *Planner* y *Transformer*, lo que será explicado con más detalle en el apartado de la especificación en Cálculoπ. En el esquema de la Figura 5.27 Surveyor

[2]MAPE: Monitor-Analyze-Plan-Execute [125]

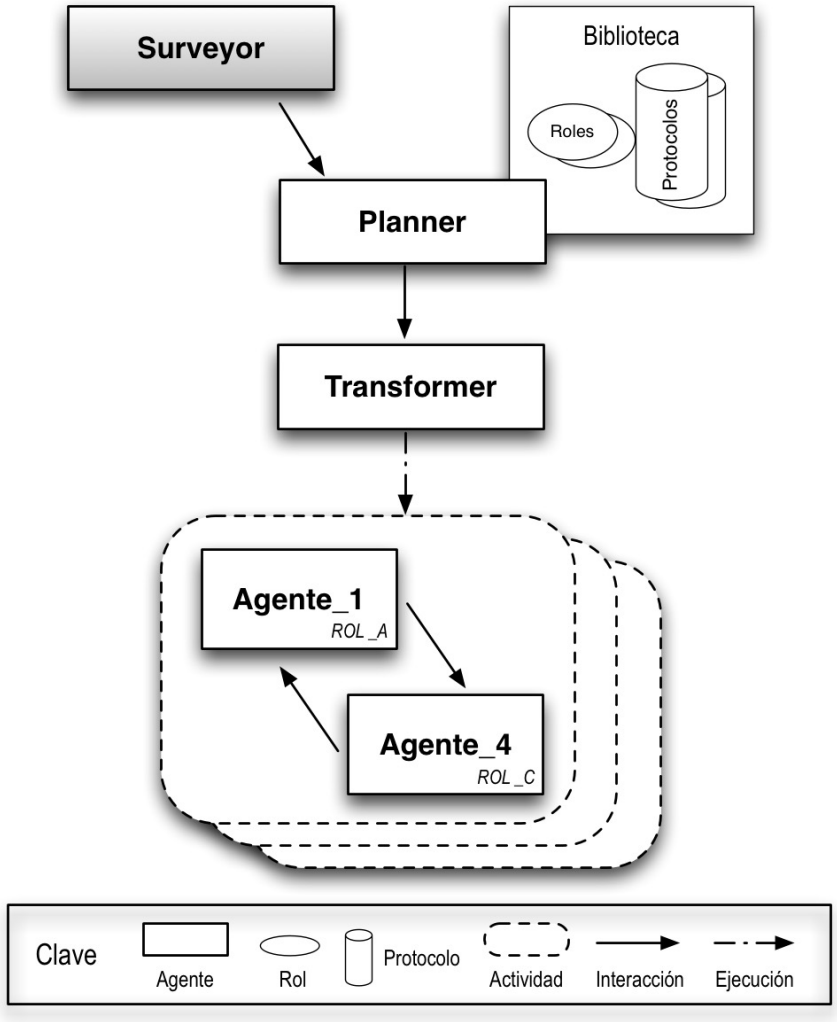

Figura 5.27: Esquema del trabajo en conjunto de los patrones *Planner* y *Transformer*.

interactúa con Planner al que puede solicitar determinado plan de acción. Este tiene acceso a bibliotecas de roles y protocolos que utilizará para determinar el o los planes. Posteriormente dichos planes serán ejecutados por Transformer, el que haciendo uso de los recursos previstos podrá llevar adelante las actividades necesarias.

- Participantes: el agente surveyor es el que determina que la situación precisa de una planificación de acciones y cambios. Un agente planificador es el responsable de realizar la planificación correspondiente, siempre teniendo en cuenta los recursos

disponibles en la *iniciativa*.

- Comportamiento: en la Figura 5.28 puede verse un ejemplo del comportamiento del patrón. El agente surveyor envía al agente planner la situación que debería ser planificada para resolverla. El planner solicita la lista de los recursos presentes en la *iniciativa*, y en base a ellos, dependencias, orden de tareas, acciones, etc. genera una planificación que después será puesta en funcionamiento por la *iniciativa*.

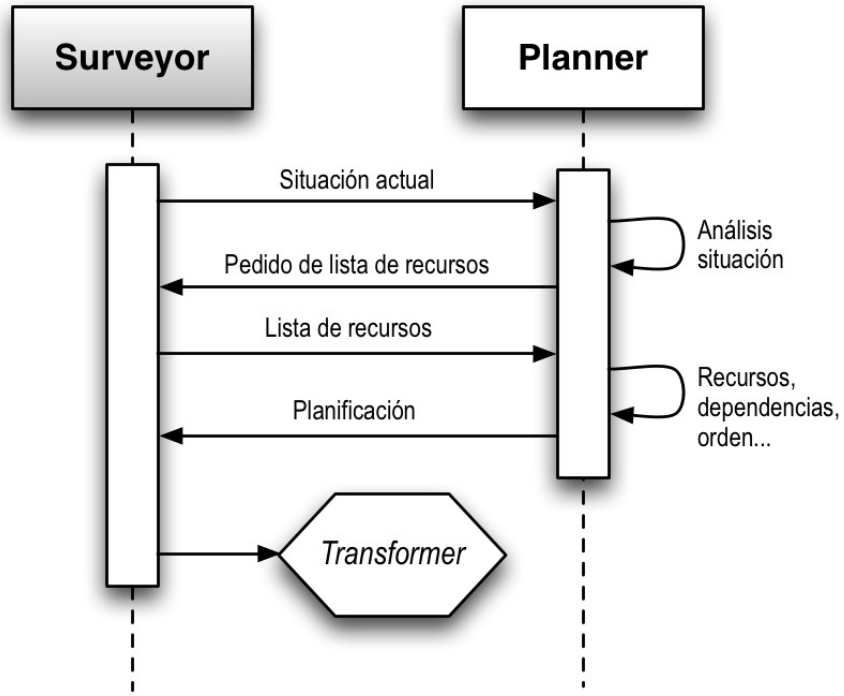

Figura 5.28: Ejemplo de comportamiento del patrón *Planner*.

- Consecuencias:
  - el agente planificador genera un plan para hacer frente una situación determinada
  - con la planificación de acciones y cambios propuestos Transformer podrá ejecutar las actividades correspondientes

- Restricciones: el patrón Planner solo será necesario si ante determinada situación se precisa de una planificación de acciones y cambios. Deberá tener en cuenta los recursos presentes en la *iniciativa*, es decir, no estará facultado para tomar la decisión de agregar más elementos o recursos.

- Patrones relacionados: el planificador puede ser el responsable de inducir transformaciones dentro de la *iniciativa*, los que podrían llevarse a cabo utilizando el patrón Transformer. El patrón Surveyor estará funcionando en paralelo ya que el agente surveyor tiene participación activa en una situación como la detallada. Transformer será el que ejecute el plan trazado. Trabajando conjuntamente con los patrones Monitor, Transformer y Surveyor podría definirse claramente un ciclo *MAPE-K*[3]

Modelo de interacción en Cálculo-π.

1. Producer/Consumer

$$\text{Producer } (dni, p, die) \triangleq$$
$$* [p! \langle datos \rangle].\tau \, (tiempo).Producer \langle dni, p, die \rangle$$
$$+ \, die?().\emptyset$$

$$\text{Consumer } (dni, p, die) \triangleq$$
$$p?(datos).\tau \, (datos).Consumer \langle dni, p, die \rangle$$
$$+ \, die?().\emptyset$$

Esquema Genérico

$$(\nu \, p, \textbf{\textit{die\_c}}1, \textbf{\textit{die\_c}}2, \textbf{\textit{die\_c}}3, \textbf{\textit{die\_p}})$$
$$(Producer \langle dni_1, p, \textbf{\textit{die\_p}} \rangle \mid$$
$$Consumer \langle dni_3, p, \textbf{\textit{die\_c}}1 \rangle \mid$$
$$Consumer \langle dni_8, p, \textbf{\textit{die\_c}}2 \rangle \mid$$
$$Consumer \langle dni_6, p, \textbf{\textit{die\_c}}3 \rangle)$$

[3]MAPE-K: Monitor-Analyze-Plan-Execute al que se le incorpora conocimiento (Knowledge).

Eventualmente:

$$\text{Kill\_All } (\widetilde{die\_l}) \triangleq$$
$$die\_p! \langle\rangle .die\_c1! \langle\rangle .die\_c2! \langle\rangle .die\_c3! \langle\rangle .\emptyset$$

---

2. Master / Slave

$$\text{Master } (dni, m, die) \triangleq$$
$$* [m! \langle orden \rangle].\tau \, (tiempo).Master \langle dni, p, die \rangle$$
$$+ die?().\emptyset$$

$$\text{Slave } (dni, p, die) \triangleq$$
$$m?(orden).\tau \, (orden).\tau \, (exe).Slave \langle dni, p, die \rangle$$
$$+ die?().\emptyset$$

Esquema Genérico:

$$(\nu \, m, die\_m1, die\_s1, die\_s2)$$
$$(Master \langle dni_2, m, die\_m1 \rangle \mid$$
$$Slave \langle dni_5, m, die\_s1 \rangle \mid$$
$$Slave \langle dni_3, m, die\_s2 \rangle)$$

Eventualmente:

$$\text{Kill\_All } (\widetilde{die\_l}) \triangleq$$
$$die\_m1! \langle\rangle .die\_s1! \langle\rangle .die\_s2! \langle\rangle .\emptyset$$

$$\text{Surveyor } (me) \triangleq$$
$$(\nu \, \widehat{meth}, \ldots, planner, die\_planner, \ldots)$$
$$(\text{Method\_List } \langle \widehat{meth}, \widetilde{m} \rangle \mid \text{MySurveyor } \langle me, \ldots, planner, die\_planner, \ldots \rangle \mid$$
$$\text{Create\_Planner } \langle planner, die\_planner \rangle)$$

MySurveyor $(me, \ldots, planner,$ ***die_planner***$, \ldots) \triangleq$

$\ldots$

$\{\mathbf{m14}\}$ $+$ ***generate_plan***$?().$

$(\nu\, s)$***group***$_o!\, \langle s \rangle\, .s?(\widetilde{g}).planner!\, \langle \widetilde{g} \rangle\, .$

MySurveyor $\langle me, \ldots, planner,$ ***die_planner***$, \ldots \rangle$

Create_Planner $(planner, bye) \triangleq$

$(\nu\, trans,$ ***die_trans***$)$

(MyPlanner $\langle planner, trans, bye,$ ***die_trans***$\rangle$

| MyTransformer $\langle trans,$ ***die_trans***$\rangle$)

MyPlanner $(planner, trans, bye,$ ***die_trans***$) \triangleq$

$planner?(\widetilde{l}).\tau\, (\widetilde{l}).$

GEN_PLAN$(\widetilde{l}, plan).$

$(\nu\, cp)trans!\, \langle cp \rangle\, .cp!\, \langle plan, \widetilde{l} \rangle\, .$

MyPlanner $\langle planner, trans, bye,$ ***die_trans***$\rangle$

$+\, bye?().$***die_trans***$!\, \langle \rangle\, .\emptyset$

MyTransformer $(trans, bye) \triangleq$

$trans?(cp).cp?(plan, \widetilde{l}).$

PROCESAR_PLAN$(plan).$

ASIGNAR_ROLES$(plan, \widetilde{l}).$

EJECUTAR_ESQUEMA$(\widetilde{l}).$

$\tau\, (\ldots).$***Kill_All***$(\widetilde{die\_l}).$

MyTransformer $\langle trans, bye \rangle$

$+\, bye?().\emptyset$

La mayoría de los patrones de adaptación en realidad llevan adelante gestiones, por lo que para que los agentes de la *iniciativa* puedan trabajar en conjunto es necesario facilitarles patrones de comunicación. En este contexto lo patrones Planner y Transformer son especialmente adecuados.

Cuando se pone en marcha la *iniciativa* también se inicia un proceso planificador. Este va a ser en principio un proceso independiente pero si hay que asociarlo con un agente sería el Surveyor. La idea es que Planner puede preguntar al Surveyor quienes son los integrantes del grupo y también pedirle la lista de capacidades. Con estos elementos ya es posible establecer básicamente dos protocolos bien conocidos: *productor/consumidor*, y *maestro/esclavo*. La definición de estos se realizan como patrones de procesos (*templates*), y si por ejemplo se necesita un acceso a base de datos se realizará una versión concreta para ese caso.

En este trabajo de tesis solo se definirán *templates* genéricos.

Para darle funciones a la *iniciativa* para que realice algo concreto se tendrá que facilitarle versiones concretas de esos procesos. De esta manera Planner puede generar estos esquemas y definir roles genéricos para que después Transformer decida sobre esos procesos genéricos qué procesos concretos tendrá que instanciar. Es decir, Planner genera la planificación, tiene una plantilla y Transformer la ejecuta, instancia esa plantilla con los agentes concretos. En base a estas características puede notarse que Transformer es un *changent* y los otros agentes serán *shifters*.

Planner invoca una función de planificación, genera un plan y Transformer lo concreta.

Los esquemas genéricos de Producer y Consumer son similares, la diferencia está en el tratamiento de los datos, el primero enviará datos (con una replicación para todos los consumidores) y el segundo recibirá los datos correspondientes. Cada uno tiene su canal die, más que nada para saber cuándo acaban su tarea. En el ejemplo son 3 consumidores y 1 productor, y la comprensión del despliegue del plan no presenta mayores dificultades. Eventualmente, cuando se termina el plan se eliminan a todos con Kill_All.

Los esquemas respectivos de Master y Slave son similares a los anteriores, solo que

ahora son órdenes del Master que Slave debe ejecutar. En el ejemplo son 2 slaves y 1 master, el despliegue se comprende fácilmente. Eventualmente también se pueden eliminar a todos los actores con Kill_All.

Las especificaciones en Cálculo-π:

En el proceso Surveyor, en paralelo a la invocación a MySurveyor se invoca la creación del planificador con Create_Planner.

Se utiliza el método **m14** con el que se generan los planes de un grupo de integrantes.

Create_Planner recibe el planner y el canal de fin, seguidamente invoca a MyPlanner pasando entre otros a un transformer (trans) y su canal de fin (die_trans). En paralelo se invoca el proceso MyTransformer.

MyPlanner se encarga de solicitar y recibir la lista de los integrantes de la *iniciativa* y con ella se genera el o los planes, utilizando un algoritmo planificador conveniente. Asimismo envía el plan a Transformer. Con bye finaliza y también ordena el fin del transformer.

El proceso MyTransformer recibe trans y bye, con el plan y los integrantes el Transformer está preparado para procesar el plan, asignar roles y ejecutar el esquema correspondiente. Con Kill_All elimina a todos los participantes. Con bye finaliza él mismo.

5.5.15. Patrón *Transformer*.

- Nombre del Patrón: **Transformer**.
- Clasificación:
  - *Clasificación 1*: Reconfigurador (R), de Comportamiento (B)
  - *Clasificación 2*: EV (Evolution)
- Intención: este patrón describe el proceso que permite "transformar" el comportamiento de un agente, es decir, que el agente logre pasar de su rol definido a jugar otro.
- Contexto: el patrón Transformer puede ser utilizado cuando:

  - la *iniciativa* aún continúa su crecimiento
  - es preciso que un agente sufra una transformación de roles en base a cambios en el entorno

- Motivación: el grupo preliminar todavía no se establecido como organización plena. Cambios en el entorno inducen a realizar los cambios necesarios en los individuos que correspondan dentro de la *iniciativa*. Una manera de hacerlo es con la transformación de ciertos roles, por lo que un agente jugando determinado rol debería pasar a jugar otro rol sin que el objetivo principal se viera alterado.

- Esquema: como se plantea el trabajo conjunto con *Planner* el esquema correspondiente a ambos es el de la Figura 5.27. El comportamiento detallado también puede verse en la especificación del citado patrón (5.5.14).

- Participantes: el agente surveyor, el planner y un agente al que por ejemplo se le añade un rol para la ejecución del plan de actividades previsto por planner.

- Comportamiento: la Figura 5.29 representa el caso en el que el agente surveyor, ante determinado contexto, dispara el patrón Planner. Este último, como parte de su planificación, requiere por ejemplo un cambio de rol en el Agente_X. Se le añade a este el rol correspondiente y posteriormente se informa al surveyor.

- Consecuencias:

  - roles genéricos definidos por planner son aplicados a agentes concretos
  - se ejecutan las acciones previstas en la planificación

- Restricciones: el agente jugando un rol definido para pasar a jugar otro rol debe tener las capacidades mínimas necesarias para el segundo rol. El patrón no será utilizado si todos los agentes de la *iniciativa* están jugando el rol que les corresponde, si no hay posibilidades de que los agentes puedan cambiar de roles, o si no existe planificación que lo justifique.

- Patrones relacionados: el patrón Surveyor funciona en paralelo, dispara la necesidad de planificación, la que será obtenida por Planner para su ejecución.

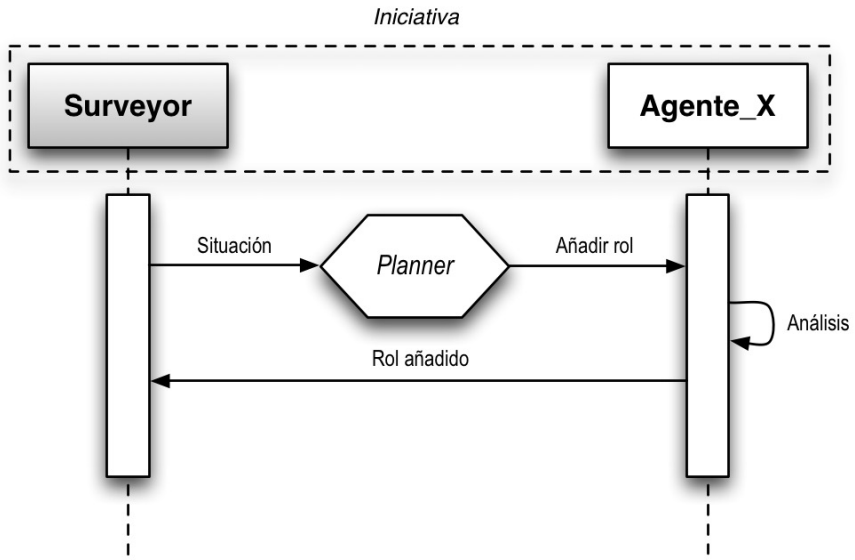

Figura 5.29: Ejemplo de comportamiento del patrón *Transformer*.

Modelo de interacción en Cálculo-π.

Tanto la especificación en Cálculo-π como su explicación pueden verse en 5.5.14.

5.5.16. Patrón *Monitor*.

- Nombre del Patrón: **Monitor**.
- Clasificación:
  - *Clasificación 1*: Monitor (M), Estructural (S)
  - *Clasificación 2*: OB-I (Observación Interna)
- Intención: este patrón facilita la monitorización de las interacciones relevantes dentro de la *iniciativa*.
- Contexto: el patrón Monitor puede ser utilizado cuando:
  - la *iniciativa* continúa en su crecimiento

- el número de integrantes se hace cada vez mayor
- agentes muy heterogéneos conforman el grupo preliminar
- se precisan *logs* para posterior análisis

- Motivación: el grupo preliminar aún no se establecido como una organización plena. Las interacciones dentro de la *iniciativa* pueden ser capturadas en *logs* por este monitor. Puede posibilitar el descubrimiento de mayores capacidades en la organización emergente. Es factible que existan varios monitores con diferentes tipos de filtrado.
- Esquema: en el gráfico de la Figura 5.30 dos agentes interactúan y Monitor recibe un *log* de la interacción. Esta información podría ser analizada posteriormente para tomar las decisiones correspondientes.

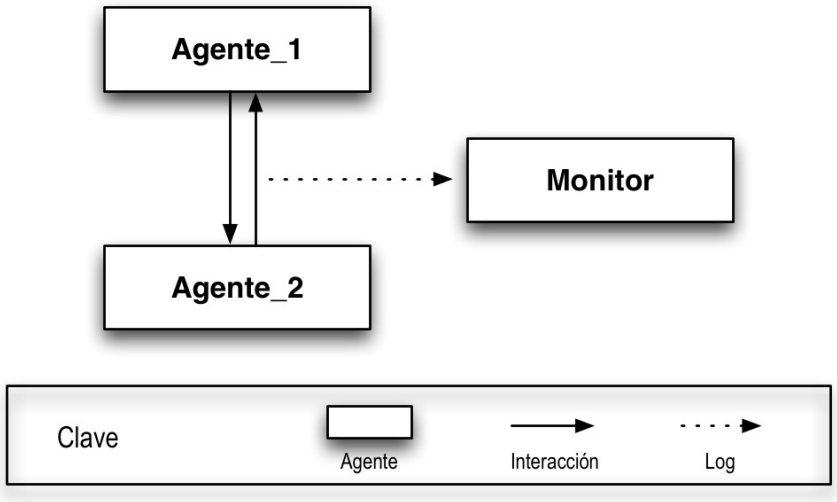

Figura 5.30: Esquema del patrón *Monitor*.

- Participantes: como mínimo participan dos agentes (uno de ellos podría ser el surveyor) y un monitor formando la *iniciativa*.
- Comportamiento: en la Figura 5.31 puede verse el comportamiento del agente monitor que captura y analiza las interacciones entre Agente_1 y Agente_2. Luego del análisis de estos datos se podría informar al surveyor sobre la posibilidad de cambiar

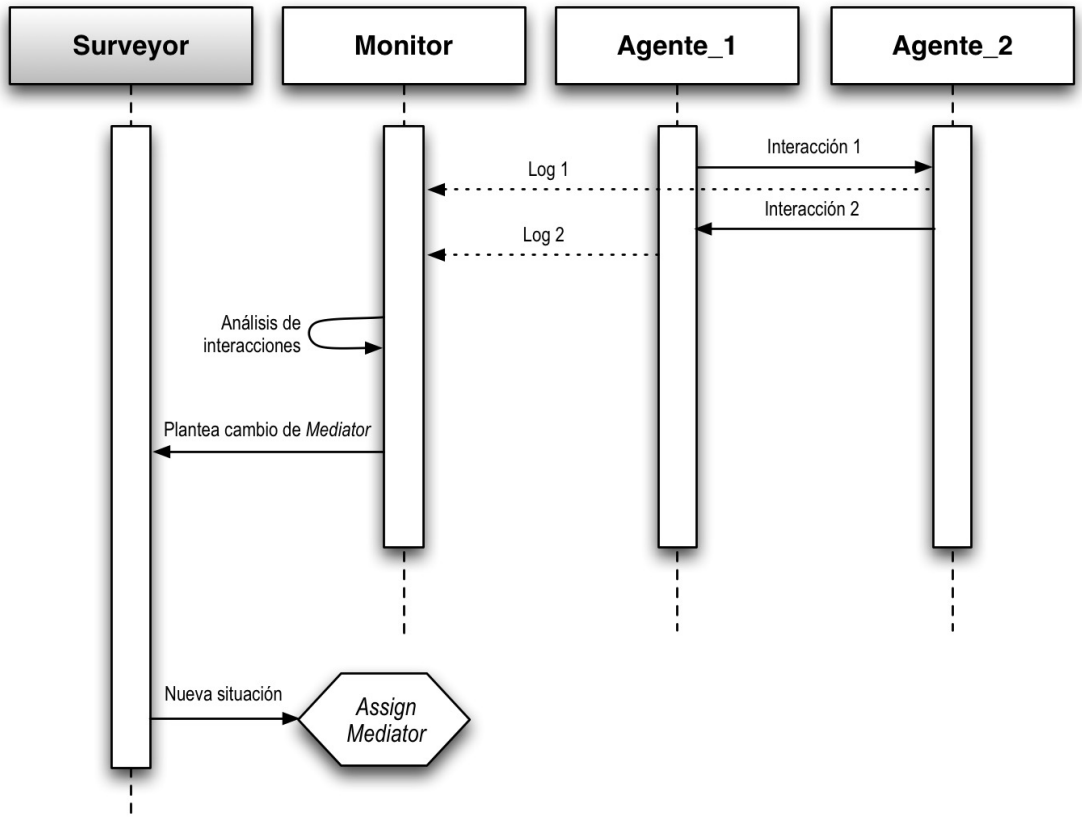

Figura 5.31: Ejemplo de comportamiento del patrón *Monitor*.

al mediador, posiblemente porque uno de ellos presenta mayores habilidades. En este caso el agente surveyor podría lanzar el patrón Assign Mediator.

- Consecuencias:
  - las interacciones de los agentes de la *iniciativa* son monitorizadas
  - los datos de los suscriptores al monitor son actualizados
  - podría facilitar la decisión de disparo de otros patrones para inducir cambios en agentes específicos
  - el agente surveyor podrá recibir actualizaciones sobre las habilidades de otros agentes

- Restricciones: para usar este patrón al menos un agente debe tener un canal de monitorización. El agente monitor no dispara otros patrones, solo facilitaría la decisión por parte del agente surveyor.

- Patrones relacionados: el patrón Surveyor funciona en paralelo. Monitor se relaciona también con todos los patrones que generen información relevante y que podría ser utilizada por Surveyor para realizar cambios, por ejemplo, Mediator, Façade, Change Surveyor, Assign Mediator, New Mediator, Update Façade. Con el patrón Planner y en conjunto con Surveyor y Transformer podría definirse un ciclo MAPE-K.

Modelo de interacción en Cálculo-π.

Boot_Sys $() \triangleq$
$(\nu\, Log)$(Creator_Agent $\langle \ldots, Log \rangle$
| Space $\langle \ldots, Log \rangle$
Creator_Monitor $\langle \ldots, Log \rangle$)

Creator_Monitor $(Log) \triangleq$
$(\nu\, \textbf{store}\,)$MyMonitor $\langle Log, \textbf{store}\, \rangle$

MyMonitor $(Log, \textbf{store}\,) \triangleq$
$*\,[Log?(m).\textbf{store}\,!\,\langle m \rangle\,]$
MyMonitor $\langle Log, \textbf{store}\, \rangle$

Para que el Patrón Monitor pudiera controlar todo el funcionamiento de la *iniciativa* debería crearse un canal de monitorización en cada proceso. Es decir, generar un *Log* en cada proceso, aunque lo más práctico sería pasarle a todos los procesos importantes un canal de *Log* en los momentos que se producen cambios importantes. Por ejemplo en un cambio de mediador, como se vió en la especificación de Thru Mediator (ver 5.5.10). Como característica, Monitor no depende del Surveyor y tampoco es conocido por los demás agentes. Mientras el sistema está en marcha Monitor estará en funcionamiento y en realidad el *Log* es único.

Básicamente, en el inico (*Boot_Sys*) de nuestro sistema de agentes se tiene una serie de procesos iniciales, Creator_Agent para cada agente que se crea, Space para crear un espacio compartido, y Create_Monitor, que aunque forma parte del patrón Monitor se lanza aquí porque debería monitorizar desde el principio. En cada uno de estos habrá un *Log*.

Creator_Agent pasará el *Log* a AgentX, Space pasará a Venue, Venue a Surveyor (a través de Commission de Elect Surveyor), Surveyor se lo pasa a Façade y Mediator, como ejemplos. En lugares precisos de cada uno para indicar cuestiones importantes.

El proceso Creator_Monitor recibe Log, crea una variable para almacenar (store) e invoca MyMonitor.

El proceso MyMonitor recibe Log y store, con una replicación en log se espera el mensaje (m) y seguidamente el canal store envía m. Store es una abstracción de almacén secuencial (un array, una base de datos). A continuación puede lanzarse el bucle.

5.5.17. Patrón *Terminator*.

- Nombre del Patrón: **Terminator**.
- Clasificación:
  - *Clasificación 1*: Decisor (DM), Estructural (S)
  - *Clasificación 2*: DE (Destruir Organización)
- Intención: el patrón describe la finalización de la actual *iniciativa*, la que no ha llegado a establecerse como organización completa. Es lanzado una vez que quede un solo miembro en el grupo preliminar.
- Contexto: el patrón Terminator puede ser utilizado cuando:
  - la *iniciativa* no ha conseguido reunir las características de una organización plena
  - el grupo preliminar se ha reducido a un solo miembro
- Motivación: el grupo preliminar ha estado en continuo crecimiento, pero no ha podido reunir las características de la organización que se pretendía para conseguir llegar

al objetivo principal. No se ha podido contar con los recursos necesarios para resolver el problema.

- Esquema: la Figura 5.32 intenta reflejar la situación en que Surveyor requiere la salida de un agente, posteriormente al quedar un solo integrante (Surveyor) la *iniciativa* se da por finalizada.

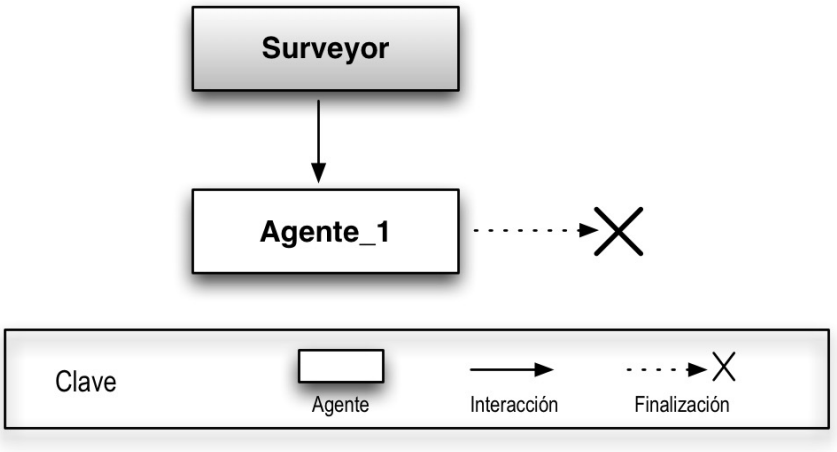

Figura 5.32: Esquema del patrón *Terminator*.

- Participantes: un agente surveyor será el encargado de liberar a los agentes, si los hubiera, antes de destruir la *iniciativa* debido a que no se ha conseguido estabilizar como organización.

- Comportamiento: la Figura 5.33 representa el comportamiento cuando la *iniciativa* no puede consolidarse como organización plena, por lo que libera a los agentes y se destruye el grupo preliminar.

- Consecuencias:

  - se liberan todos los agentes intervinientes en la *iniciativa*
  - la *iniciativa* es finalizada
  - no se ha llegado al *acuerdo* para establecer una organización de plena

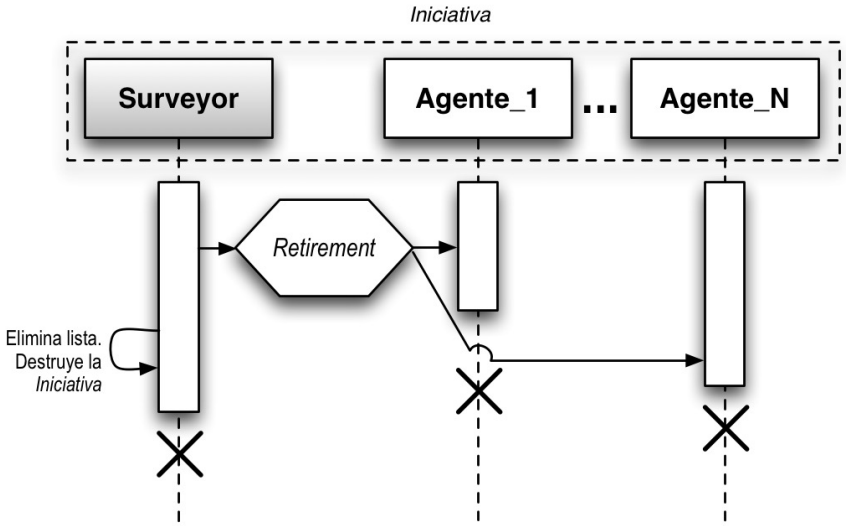

Figura 5.33: Ejemplo de comportamiento del patrón *Terminator*.

- Restricciones: no se podrá utilizar este patrón si la *iniciativa* ha podido consolidarse como una organización completa. En caso que hubiera más de un agente en el grupo preliminar que debe ser destruido, primero deberán ser liberados los agentes hasta que el último (el surveyor) habilite su uso.

- Patrones relacionados: el patrón Surveyor está funcionando en paralelo. El agente surveyor es el responsable final de la destrucción de la *iniciativa*. El patrón Retirement puede ser utilizado para liberar los agentes antes de extinguir el grupo preliminar. Consolider puede disparar este patrón en base a determinadas condiciones con respecto a la consolidación de la organización.

Modelo de interacción en Cálculo-π.

$$\begin{aligned}&\text{Space}\ (\mathit{sys\_where}, \mathit{sys\_end}, \mathit{sys\_terminate}) \triangleq \\ &\quad (\nu\, \widehat{dead})(\text{Dead\_List}\ \langle \widehat{dead}, \_ \rangle\ \mid \\ &\quad\ \text{Venue}\ \langle \mathit{sys\_place}, \mathit{sys\_end}, \mathit{sys\_terminate}, \widehat{dead} \rangle\ )\end{aligned}$$

AgentX $(\ldots, \textit{sys\_where}, \textit{sys\_end}, \ldots) \triangleq$

$(\nu\, die)(\ldots$

$+ (\nu\, hi)(\textit{sys\_where}!\, \langle hi \rangle .\textit{sys\_where}?(where).\textit{sys\_end}!\, \langle die \rangle .\tau .where?(event).$

$(\nu\, me, ret) * [event!\, \langle me, ret \rangle .ret?(you).\tau .(me!\, \langle data \rangle \mid you?(info))];$

AgentX $\langle \textit{sys\_where}, \textit{sys\_end} \rangle$)

$+ die?().\emptyset)$

Venue $(\textit{sys\_place}, \textit{sys\_end}, \textit{sys\_terminate}, dead) \triangleq$

$\ldots$

$+ \textit{sys\_end}?(die).\textbf{\textit{add\_list}}\, \langle \widehat{dead}, die \rangle .$

Venue $\langle \textit{sys\_place}, \textit{sys\_end}, \textit{sys\_terminate}, \widehat{dead} \rangle$

$+ \textit{sys\_terminate}?().$

$(\nu\, r)\textit{dead}_o!\, \langle r \rangle .r?(\widetilde{x})$.Forloop $\langle \widetilde{x} \rangle$;

Venue $\langle \textit{sys\_place}, \textit{sys\_end}, \textit{sys\_terminate}, \widehat{dead} \rangle$

Forloop $(\widetilde{x}) \triangleq$

$(\nu\, z)z!\, \langle \widetilde{x} \rangle\, (z?(i, \widetilde{y}).i!\, \langle\rangle .$

Forloop $\langle \widetilde{y} \rangle$

$+ z?(i, \_).i!\, \langle\rangle$)

Dead_List $(\widehat{dead}, \widetilde{c}) \triangleq$

$\textit{dead}_i!\, \langle \widetilde{x} \rangle$ .Dead_List $\langle \widehat{dead}, \widetilde{y} \rangle$

$+ \textit{dead}_o?(r).r!\, \langle \widetilde{c} \rangle$ .Dead_List $\langle \widehat{dead}, \widetilde{c} \rangle$

Como ya se ha explicado, este patrón se utiliza para la eliminación de la *iniciativa*. Por tanto para eliminarla habrá que eliminar los agentes que la forman, es así que el proceso *AgentX* es consciente de su existencia y utiliza el canal de sistema *sys_end* para enviar su canal de eliminación (die). Además se tiene un punto de finalización de la recursividad, el *die* al final de su definición.

El proceso Space, considerado como un espacio de cálculo antes que uno físico, recibe también parámetros relacionados con el final, por ejemplo con sys_terminate se enviará la orden para "morir". Se crea la lista de "muerte" que va completándose con la función Dead_List. En paralelo se invoca Venue.

El proceso Venue por un lado con la función **add_list** agrega los canales de finalización (die) a la lista $\widehat{dead}$; y por otro lado con sys_terminate se envía la orden de "muerte", con Forloop se van eliminando uno a uno los integrantes de la *iniciativa*.

5.5.18. Patrón *Environment*.

- Nombre del Patrón: **Environment**.
- Clasificación:
  - *Clasificación 1*: Monitor (M)
  - *Clasificación 2*: OB-E (Observación Exterior)
- Intención: este patrón posibilita la observación hacia el exterior de la *iniciativa* y la percepción de los cambios del entorno. Tiene una función más proactiva.
- Contexto: el patrón Environment puede ser utilizado cuando:
  - la *iniciativa* continúa con su crecimiento
  - existen elementos externos con la posibilidad de interactuar con el grupo preliminar
  - existen elementos externos con la intención de formar parte de la *iniciativa*
  - el entorno tiene características cambiantes
- Motivación: el grupo preliminar todavía no se ha consolidado como organización. El contexto del problema presenta características cambiantes. Se precisa un elemento con capacidad de percibir los cambios del exterior que pueden influir en el comportamiento de la *iniciativa*.

- Esquema: en la Figura 5.34 environment puede percibir el exterior y recibir información de él. También puede tener la posibilidad de acceder al surveyor y solicitarle, por ejemplo, su lista de métodos y posteriormente invocarle determinadas funciones.

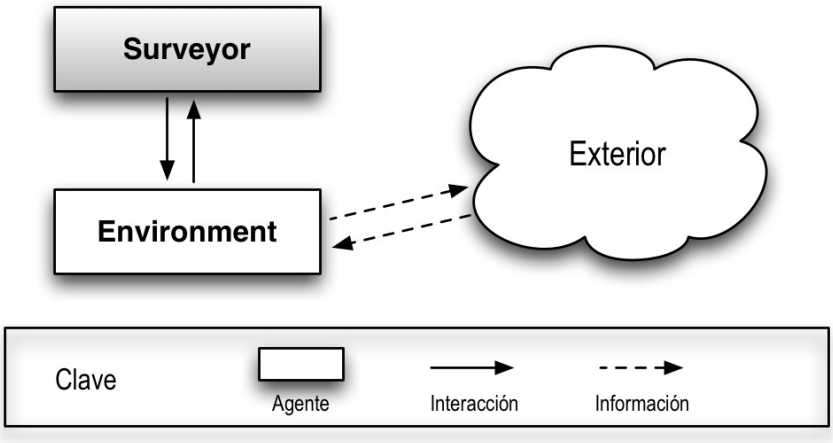

Figura 5.34: Esquema del patrón *Environment*.

- Participantes: en este patrón participan el agente surveyor y un agente tipo environment *(context awareness)* para percibir el exterior.

- Comportamiento: la Figura 5.35 representa un posible comportamiento en el cual podría participar un agente environment, que con actitud proactiva propone a un agente externo integrar la *iniciativa*. Al aceptar la propuesta el environment informa a surveyor, este lanza el patrón Thru Mediator para incluirlo en la lista de integrantes del grupo. Posteriormente el agente externo es informado de su aceptación y recibe la lista actualizada para interactuar con los integrantes de la *iniciativa*.

- Consecuencias:

  - es posible percibir los cambios en el entorno
  - facilita la posible interacción con elementos externos
  - facilita el posible ingreso de agentes externos como integrantes de la *iniciativa*

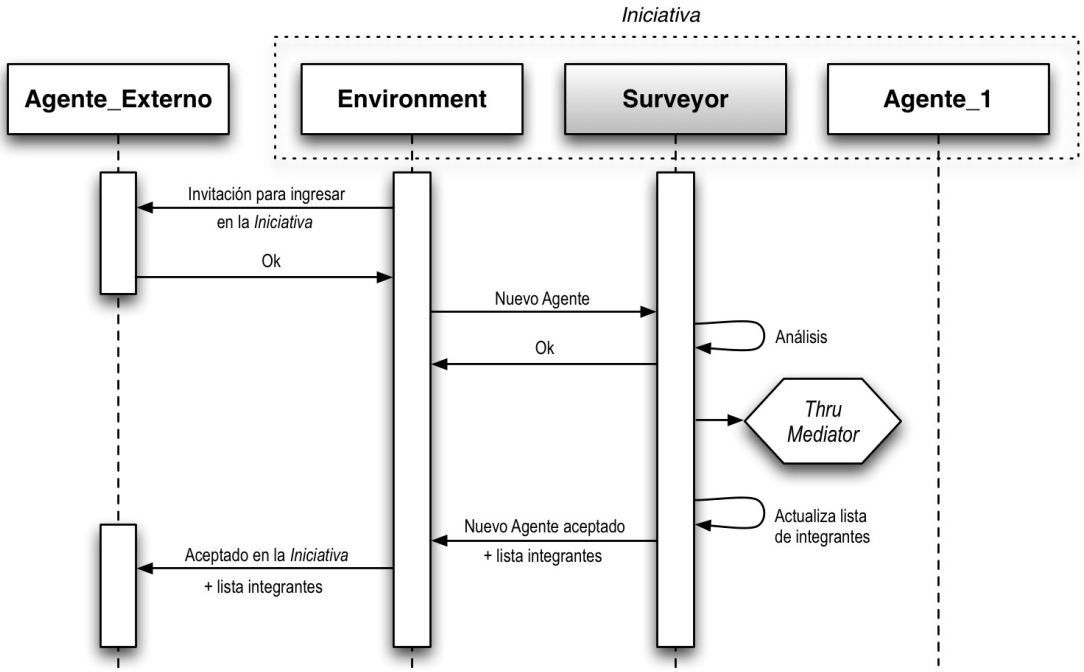

Figura 5.35: Ejemplo de comportamiento del patrón *Environment*.

- Restricciones: no se podrá utilizar este patrón si no es intención del grupo preliminar la interacción con el exterior. Tampoco se utilizará si no se admite el ingreso de elementos externos en determinado momento de la emergencia de la organización.

- Patrones relacionados: el patrón Surveyor está funcionando en paralelo. También se relaciona con Thru Mediator y Full Mesh para determinar los posibles agentes externos que precisen tanto interactuar con los integrantes de la *iniciativa* como pasar a formar parte de ella.

Modelo de interacción en Cálculo-π.

Environment $(ambient, \textit{sys\_office}) \triangleq$

$(\nu\ surveyor)$

MyEnvironment $\langle ambient, \textit{sys\_office}, surveyor \rangle$

MyEnvironment $(ambient, \textit{sys\_office}, surveyor) \triangleq$

$ambient?(info).\tau\,(info).$

DECISION_SYSTEM$(info, decision, plan).$

if $cond(decision)$ then Reaction_Domain$(info, decision)$

else Reaction_Iniciativa$(info, plan).$

MyEnvironment $\langle ambient, \textit{sys\_office}, surveyor\rangle$

$+\ \textit{sys\_office}?(boss).$

MyEnvironment $\langle ambient, \textit{sys\_office}, boss\rangle$

Reaction_Domain $(info, decision) \triangleq$

case $decision$ in

$1:\ \langle \textit{metodo\_dominio}_1\rangle$

$+\,2:\ \langle \textit{metodo\_dominio}_2\rangle$

$+\,3:\ \langle \textit{metodo\_dominio}_3\rangle$

$+\ldots$

Reaction_Iniciativa $(info, plan) \triangleq$

$\tau\,(plan, secuencia).$

UNFOLD$(secuencia);$

Unfolded1 $\langle surveyor, externo\rangle$

Unfolded1 $(surveyor, externo) \triangleq$

if $tipo(externo) =$ AgentX then $\langle$FULL MESH$\rangle\,(desarrollo)$

else $(\nu\, r)surveyor!\,\langle r\rangle\,.r?(metodos).$

$(\nu\, s)metodos!\,\langle s\rangle\,.s?(\widetilde{m}).$

$m_1!\,\langle externo\rangle\,.\emptyset$

El patrón Environment tiene la capacidad de percibir los cambios del exterior, puede tener comunicación con el *middleware* por medio de un canal privilegiado, y a partir de ahí responde a un esquema de percepción-acción o también puede tomar decisiones. Con

información de dominio tendría que reaccionar según esquemas concretos de cada dominio, y a su vez secuencias concretas llevan a tomar decisiones. Es decir que el patrón da esquemas generales de comportamiento que debería refinarse en cada contexto. Por ejemplo, si se decidiera trabajar en un entorno de *ambient intelligence* los estudios al respecto deberían formar parte de las características de Environment.

El proceso Environment se lanza también desde *Boot_Sys* y recibe ambient (canal privilegiado) y sys_office. Este último le permite conocer al Surveyor una vez que este ha sido proclamado. Seguidamente invocará a MyEnvironment con esos parámetros y una tercera variable (surveyor).

MyEnvironment recibe en su canal privilegiado la información con la que deberá tomar decisiones, utilizando cierto sistema de decisión. Con la información recibida y según las condiciones se marcarán las reacciones del proceso. Si se cumpla cierta condición sobre la decisión se lanza la reacción sobre el dominio, que como no se puede preveer ahora se estructura sobre un conjunto de métodos que lanzarían protocolos concretos para cada caso. Por el contrario, si la reacción es sobre la *iniciativa*, con la información y el plan se genera una secuencia de pasos que invoca a UNFOLD.

UNFOLD recibe esa secuencia de pasos que se traduce (Unfolded1) en un código concreto en Cálculo-π (incluso puede estar en una biblioteca). Por ejemplo, cuando se tenga acceso a un agente interesante para la *iniciativa*, con un código preelaborado puede informar al Surveyor el tipo de agente y este desplegar FULL MESH. Al invocar Surveyor devuelva la lista completa de métodos, y con **m1** agregará al agente externo como integrante del grupo.

Luego de la descripción realizada de todos y cada uno de los *patrones de adaptación*, en el apartado siguiente procedemos a explicar, a grandes rasgos, la dinámica del funcionamiento de dichos patrones.

5.6. Dinámica del Funcionamiento de los Patrones

En las secciones previas de este capítulo, así como en capítulos anteriores, se han explicado el concepto de patrón de software y sus posibles clasificaciones. También que ellos pueden ser vistos como formando un lenguaje, el que tiene un *vocabulario* (los propios patrones), una *sintaxis* (sus relaciones), y una *gramática* (o cómo actúan en la resolución de los problemas, de manera individual o en conjunto). Sin embargo, también es menester aclarar que un "lenguaje de patrones" no es un lenguaje para definir patrones.

Asimismo se ha detallado el catálogo de los patrones que componen nuestra propuesta de *patrones de adaptación*, con su descripción, su desempeño, funcionalidad, etc.

Para completar esta visión es importante también establecer la dinámica de los patrones de adaptación en cuanto a su funcionamiento.

En general puede determinarse que el funcionamiento de un patrón (o un conjunto de ellos) tiene básicamente tres estadios, los que responden a las siguientes preguntas:

1. ¿ cómo se dispara el patrón?
2. ¿ cómo se ejecuta el patrón?, y
3. ¿ cómo se detiene el patrón?

Para responder a estas preguntas es necesario comprender que nuestra propuesta incorpora un mecanismos de adaptación completo y que es llevado adelante por los patrones. Este mecanismo realiza el proceso de adaptación por sobre el sistema que ya está en funcionamiento y sin la necesidad de realizar modificaciones en los agentes del sistema. El SMA puede tener su propia estructura e integrantes (los que podrían ser, por ejemplo, agentes THOMAS [13] o agentes JADE [24]) y estar operando de manera convencional persiguiendo el objetivo para el que fue desarrollado.

Es decir, que nuestra propuesta se basa en que los agentes del SMA realicen sus funciones normalmente y, llegado el punto en que sea necesaria una adaptación, nuestros patrones puedan desplegarse sin tener que "interrumpir" la labor de los agentes y sin modificarles sus metas o capacidades.

Estos *patrones de adaptación* definen un comportamiento genérico y en base a las condiciones de disparo, tanto por cambios en el entorno como los propios del sistema, será el despliegue de uno u otro patrón. Al hacerlo, los procesos definidos en cada uno de ellos (por ejemplo el *AgentX*) tendrán que ser ejecutados en paralelo y sin interferir en el funcionamiento básico del SMA. Puede interpretarse como si se tratara de un "segundo" agente que se acopla a cada uno de los integrantes del SMA cuya función es participar en el proceso adaptativo que se ha generado, compartiendo espacio con los anteriores. Estos agentes acoplados, según sean necesarios, pueden tener la forma tanto de agente *changent* como de agente *shifter* (ver 4.3.2). De esta manera, y con estos procesos en paralelo se logra que los agentes, aparte de su función normal, puedan moverse, coordinarse, comunicarse, y en definitiva adaptarse. En pocas palabras, estamos logrando que los agentes del SMA puedan comunicarse con otros sin saberlo realmente porque esa comunicación solo está relacionada con el mecanismo de adaptación utilizado.

El patrón puede detectar la condición de disparo y esto le permite desplegarse y generar las estructuras necesarias para comportarse a partir de un *template* predefinido. La definición abstracta del patrón viene dada por una serie de código genérico cuyo lanzamiento permite su concreción específica. Es así que los procesos necesarios se acoplan a los elementos correspondientes y se ejecutan siguiendo el esquema previsto en el patrón. Esto nos proporciona un comportamiento distribuido en la medida que el o los patrones son desplegados, dando en conjunto el comportamiento global que definimos.

Como típicamente estos procesos están definidos de manera recursiva (puede notarse esto en las especificaciones en Cálculo-π de la Sección anterior), mientras la condición de disparo se mantenga el patrón volverá a ejecutarse. En caso de no conservarse esta condición el patrón debe detener su operación.

Sin embargo, pueden existir casos en que aún con la condición de disparo presente el patrón deba dejar de ejecutarse. En estas circunstancias es necesaria una segunda forma de parar su ejecución por lo que aquí juegan un papel importante los llamados *inhibidores*. La función de estos es abortar un nuevo disparo del patrón, como se explicado anteriormente.

Tomemos como ejemplo el caso del patrón Gathering (5.5.1). Cuando el SMA está en

funcionamiento y un agente se incorpora a dicho sistema se le acopla un "segundo" agente (el proceso *AgentX*), que se corresponde con un agente *shifter* como ya se explicó previamente. Si no existe condición de disparo estos procesos no interfieren en el normal desarrollo del SMA. Si por ejemplo estamos en el dominio de la emergencias, al ocurrir una de ellas el patrón puede dispararse y con él se crea el proceso *Venue*, un *changent*, para facilitar la reunión de los agentes que se encuentran en las cercanías. Cada agente que se aproxime al lugar representado por *Venue* tiene asociado su *AgentX*, por lo que estaría en condiciones de formar parte de la *iniciativa*. Esta se va creando con los agentes que han pasado por el Gathering y han quedado vinculados, pudiendo comunicarse entre si.

Otro ejemplo interesante se puede ver con el patrón Elect Surveyor, en el que todos los agentes tienen asociados un *shifter*, ya que el patrón Gathering les ha incorporado la capacidad de conversar entre ellos, es decir, que "han cambiado". En determinado momento se provoca un mecanismo de elección y uno de ellos es elegido líder, sin que el patrón original se modifique. Un agente *changent* dispara el patrón Elect Surveyor, el que por lógica debería dejar de ejecutarse una vez que se haya elegido al Surveyor. Sin embargo la condición original de que se han convocado elecciones no ha desaparecido, por lo que se tiene que lanzar el *inhibidor* correspondiente que determine que las elecciones se han acabado.

Con el patrón Terminator se pueden eliminar todos los procesos *AgentX* utilizando el canal correspondiente (5.5.17). De esta manera esos agentes quedarán desvinculados del sistema de la *iniciativa* al disolverse esta, pero sin modificar el funcionamiento del SMA original, como ya se ha dicho.

Para finalizar queremos hacer énfasis en que la definición de nuestros *patrones de adaptación* es teórica, ya que independientemente del mecanismo que se implemente esto proporciona a cualquier sistema de agentes distribuidos la capacidad de convertirse en un sistema adaptativo, y en particular de sistemas con las características de las Tecnologías del Acuerdo. Por supuesto que la implementación concreta va a depender de esas características, lo que de ninguna manera es exhaustivo ya que es posible llevarlo adelante con otros enfoques, como por ejemplo el *reflexivo*.

Con la intención de proporcionar una visión más clara de la propuesta, en la sección

siguiente se explicará el ciclo de vida utilizando nuestros *patrones de adaptación*.

5.7. El Ciclo de Vida de las Estructuras Auto-organizadas visto con los Patrones de Adaptación

Es importante comprender que nuestro *lenguaje de patrones* gira alrededor de la estructura básica que hemos llamado *iniciativa*. Como ya hemos explicado, una *iniciativa* es una estructura emergente que evoluciona con la dinámica del entorno. Es un grupo preliminar de individuos (en nuestro enfoque son los agentes software) que se ensamblan en una estructura generada por un conjunto de *controles* y *protocolos* [182], como hemos visto en 4.2.1.

En los párrafos siguientes vamos a explicar la relación entre los patrones que se han descripto en secciones anteriores y su uso en el ciclo de vida de nuestras estructuras auto-organizadas. Dicho ciclo se corresponde con una serie de pasos que los agentes deberán dar y durante los cuales podrán tomar decisiones, acatar ciertas normas, utilizar determinados protocolos, etc., hasta que la organización plena llegue a formarse, en el caso más favorable.

La figura 5.36 es una representación gráfica del *lenguaje de patrones* propuesto y nos permitirá describir con más detalle el ciclo de vida de nuestras estructuras auto-organizadas.

De acuerdo con lo explicado como *ciclo de vida* en la sección 4.3.1, el proceso puede comenzar con un solo agente. Con una razón valedera, como por ejemplo una situación de crisis o de urgencia sanitaria como veremos en el caso de estudio del Capítulo 6, varios agentes son llamados a enfrentar la situación y su resolución favorable será el principal objetivo de la futura organización. A estos efectos el patrón a utilizar debe ser el ***Gathering***. Este patrón incluye agentes *changent*, o “agentes del cambio” [181], que inducen el cambio en el resto de los elementos del patrón.

Una vez que todos los agentes se conocen y ya están en condiciones de participar en

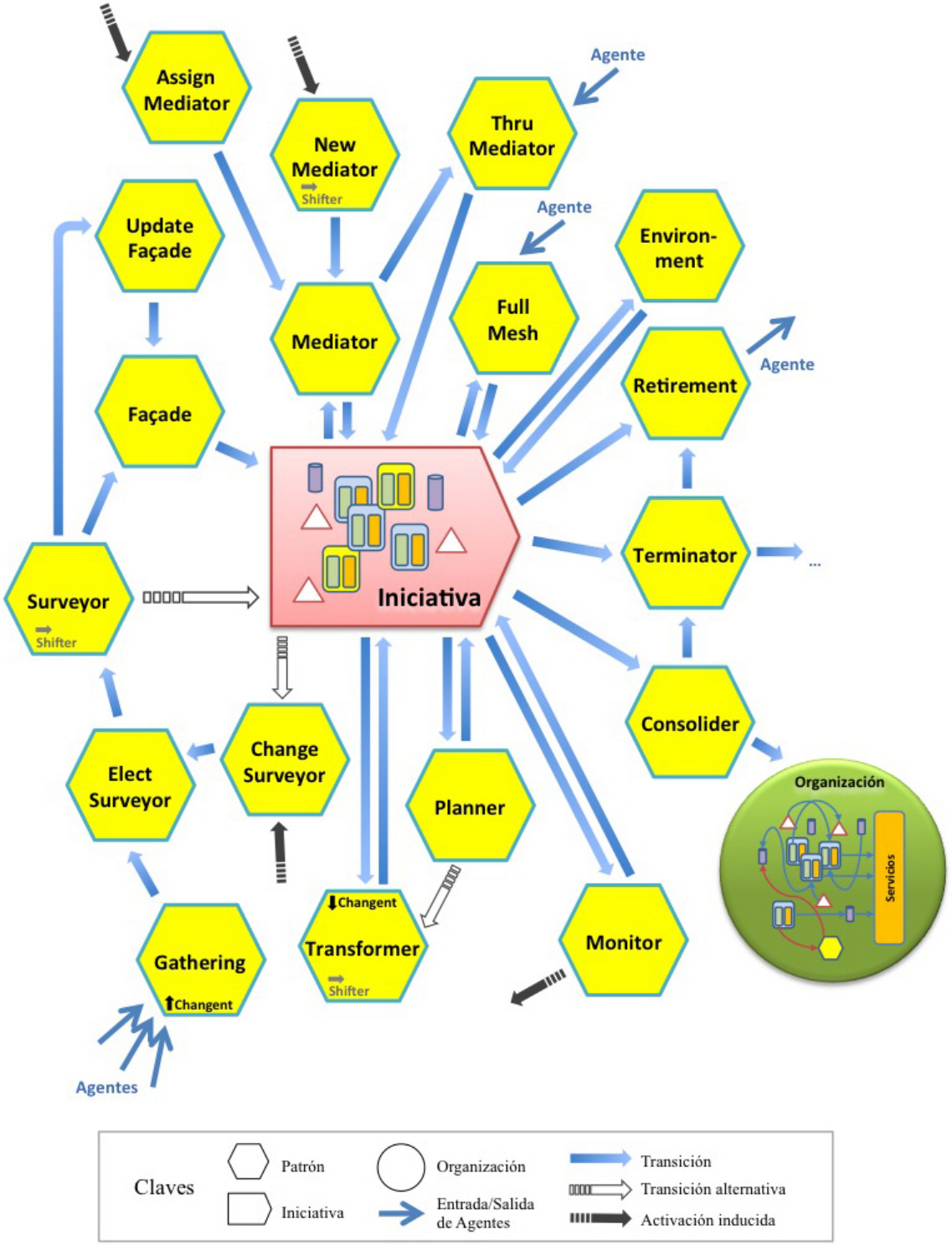

Figura 5.36: Lenguaje de Patrones de la *Iniciativa*

conversaciones puede ser lanzado el patrón ***Elect Surveyor***. Como su nombre lo indica, este patrón se utiliza para realizar la elección de un líder organizacional. Este patrón puede lanzar protocolos de elección conocidos, como por ejemplo el algoritmo de anillo o cualquier otro que se adecúe a la situación actual [147].

Por su parte, el patrón ***Change Surveyor*** puede ser utilizado cuando el actual agente surveyor debe ser reemplazado, degradándolo y trasladando su conocimiento relativo al liderazgo de la *iniciativa* al que es elegido como nuevo surveyor. Las causas pueden ir desde el deseo del actual de abandonar su puesto hasta el hecho de que su liderazgo sea cuestionado por los demás miembros y le soliciten la renuncia, por ejemplo.

Una vez elegido el líder ya es posible utilizar el patrón ***Surveyor***. Este incluye un agente *shifter*, o "agente que cambia" [181], porque un agente efectivamente ha cambiado al convertirse en el líder del grupo. Durante el proceso de surgimiento de la futura organización el patrón debe controlar el crecimiento de la *iniciativa* misma, tanto para decidir en qué momento deben agregarse nuevos elementos (agentes), como también cuándo el grupo preliminar ha alcanzado cierta estabilización. El crecimiento de la organización emergente puede ser detenido utilizando el análisis del objetivo principal y en base a los requerimientos iniciales por los que se han reunido los agentes.

Este análisis llevado a cabo por el agente surveyor actúa también como elemento *inhibidor*, ya que una vez que el objetivo principal puede ser conseguido, el surveyor estará en condiciones de detener el crecimiento del grupo preliminar. Es decir, que la *iniciativa* llega a un estado "balanceado" o seguro y puede consolidarse como organización.

El agente surveyor tiene asimismo la posibilidad de acceder a la biblioteca de patrones, entonces, y en base a las condiciones que en ese momento puede tener la *iniciativa*, estaría habilitado para disparar el patrón ***Consolider***. Este último permite que la estructura preliminar acabe como una *organización plena*. Esta estructura estable se alcanza cuando todos los participantes llegan al *acuerdo* necesario para resolver el problema o lograr el objetivo principal que había sido la causa de la reunión inicial. La organización resultante posteriormente se registra como una unidad y es conceptualmente similar a las organizaciones de numerosos enfoques de SMA, algunos de los cuales se vieron en el Capítulo 2.

Como el líder (el agente surveyor) se encuentra trabajando dentro de la *iniciativa* podría decidir que el grupo preliminar necesita un representante y de esa manera ser visto como unidad desde el exterior. A estos efectos puede lanzar el patrón ***Façade***, el que no solo representaría al grupo en términos de interacciones sino también puede redirigir cualquier comunicación entrante. Según cambios en el escenario, si el actual agente fachada debe ser reemplazado, por ejemplo si tiene abandonar el grupo porque es más necesario en otro caso de urgencia médica, se puede utilizar el patrón ***Update Façade*** para cambiarlo y trasladar sus responsabilidades al nuevo representante.

Durante el proceso de surgimiento de la estructura el grupo aún no se ha establecido, y muy posiblemente los servicios de datos no estén funcionando en plenitud. En este caso se puede usar el patrón ***Mediator*** y un agente hará las veces de mediador. Además de esta función el agente mediador estará capacitado para realizar las traducciones que fueran necesarias para facilitar la interacción entre agentes que de otra forma no podrían hacerlo, por ejemplo del formato de comunicación de la *iniciativa* al formato de un agente interno o de un nuevo agente que intenta acoplarse.

Para la sustitución del agente mediador se tienen los patrones ***Assign Mediator*** o ***New Mediator***. En el primer caso el agente mediador existente es reemplazado por otro de los integrantes de la *iniciativa* con mayores habilidades, conocimientos, etc. En el segundo caso, se lanza la elección de un nuevo agente mediador ante el hecho de que el actual debe renunciar a su rol o ha dejado de funcionar como tal, por ejemplo por la necesidad de abandonar la *iniciativa*. Por otro lado, el patrón ***Thru Mediator*** permite la interacción de nuevos individuos con el grupo emergente, o incluso su ingreso como integrante.

El patrón ***Full Mesh*** permite conexiones de tipo “todos-con-todos” entre un agente del exterior con intenciones de formar parte de la *iniciativa* y que tiene la capacidad de mantener conversaciones con todos sus integrantes. Es utilizado como una estrategia *“default”* para los casos en que un nuevo miembro tiene que ingresar en el grupo y no necesite traducciones.

En la circunstancia que uno o más agentes deban abandonar la *iniciativa*, por decisión propia o por otra razón, se lanza el patrón ***Retirement*** y así los agentes son liberados y

pueden dejar la *iniciativa*.

El patrón ***Monitor*** puede ser utilizado conjuntamente con algunos patrones que estén generando la *iniciativa*. Su función es controlar el proceso de emergencia y puede capturar y filtrar las interacciones de acuerdo a las suscripciones en este monitor. Observa las condiciones del grupo que está cambiando y favorece las correspondientes reacciones. Esto podría ser el primer paso para un ciclo MAPE [135] e inducir el lanzamiento, por ejemplo, de los patrones Change Surveyor, Assign Mediator o New Mediator, según corresponda.

Ante cierta situación del ciclo de vida puede ser necesaria una planificación de acciones y/o cambios. Como en el párrafo anterior, el clásico ciclo MAPE puede ser de utilidad. El patrón ***Planner*** será el encargado de aplicarlo e incluso inducir transformaciones dentro de la *iniciativa* y disparar el patrón ***Transformer***. Este último incluye agentes *changents* y *shifters* para lograr los cambios necesarios en los individuos dentro del patrón, o incluso podría sugerir también un cambio de patrón. La utilización del patrón Planner en conjunto con Surveyor y Transformer, como ya se ha dicho en subsecciones anteriores, define claramente un ciclo MAPE-K.

El patrón ***Environment*** será utilizado cuando se necesite que un elemento tenga la capacidad de percibir los cambios del exterior que puedan afectar el normal desarrollo del grupo preliminar emergente. Puede estar muy relacionado con los patrones Thru Mediator y Full Mesh, por ejemplo.

En el caso de que no se logre *acuerdo* alguno entre los agentes integrantes, y por lo tanto la *iniciativa* no pueda consolidarse como una organización plena, el patrón ***Terminator*** debe ser utilizado para extinguir al grupo preliminar. Este patrón libera a los agentes y elimina los canales de comunicación que se habían creado. También es usado cuando solo queda un individuo en el lugar, por lo que la *iniciativa* ha dejado de existir.

Como es usual cuando se trabaja con patrones de software, y como también surge de los párrafos anteriores, se tienen varios patrones que pueden ser utilizados de manera conjunta tanto secuencial como concurrentemente. Para saber cuáles usar juntos y en qué circunstancia puede consultarse el apartado *Patrones Relacionados* en las descripciones de cada

patrón en las subsecciones anteriores.

Por ejemplo, de manera secuencial se pueden utilizar *Gathering* y *Elect Surveyor*, en ese orden, ya que una vez que los agentes reunidos en un lugar específico puedan entablar conversaciones, el paso siguiente sería la elección de un agente líder para la *iniciativa*. Como funcionamiento concurrente se puede notar que *Surveyor*, *Monitor* y *Mediator* pueden estar trabajando en conjunto, ya que un agente surveyor puede estar suscripto a las actualizaciones de interacciones por parte de Monitor, lo que podría resultar en un necesario cambio del agente mediador debido a que, por ejemplo, acaba de ingresar a la *iniciativa* un agente con mejor rendimiento en las traducciones semánticas.

5.8. Conclusiones

En este capítulo hemos visto en detalle todas las características de la propuesta completa del ciclo de vida. Asimismo se han descrito todos y cada uno de los patrones de adaptación que forman el lenguaje de patrones y se ha explicado la relación que existe entre los patrones y su uso en el ciclo de vida de las estructuras auto-organizadas.

Es importante resaltar que todas las descripciones presentadas en Cálculo-π y que corresponden a las especificaciones de los protocolos de los patrones son teóricas, ya que su propósito principal es indicar qué tipo de protocolo se ha desarrollado. Por supuesto que se podrían implementar en distintos lenguajes de programación y con distintos esquemas.

Con estas especificaciones se ha podido probar que la propuesta de los patrones con sus respectivos protocolos es teóricamente consistente. El objetivo a continuación es probar que también tiene la posibilidad de funcionar en la práctica. En el próximo capítulo vamos a ver un ejemplo práctico de cómo funciona en el mundo real.

Capítulo 6

Caso de Estudio: Aplicación a un Sistema de Emergencias

En este Capítulo presentamos una prueba de concepto para la aplicación de varios patrones de adaptación. El caso de estudio se desarrolla en un escenario del dominio de las emergencias médicas. En primer lugar se presenta el caso de estudio y posteriormente se describe el proceso de validación llevado a cabo, con la descripción del simulador utilizado y los experimentos realizados. Finalmente se presentan las conclusiones.

6.1. Descripción del Caso de Estudio

En esta sección presentamos el caso de estudio para la prueba de concepto con el objetivo de ilustrar la situación en que una arquitectura adaptativa es una solución más que apropiada para el desarrollo de sistemas software que deban resolver problemas muy complejos. Básicamente se describirá el uso de los *patrones de adaptación* que han sido detallados en el Capítulo anterior.

El escenario que se utilizará proviene del dominio de las emergencias médicas. El ejemplo que se presenta es hipotético aunque está basado en situaciones reales y se encuentra relacionado con el software demostrador denominado *mHealth (mobile-Health)*. Este demostrador es un prototipo desarrollado en el Proyecto Tecnologías del Acuerdo (Agreement

Technologies - AT) [19] y con la cooperación de SUMMA112 [217]. Esta última entidad gestiona las emergencias médicas en la Región Autónoma de Madrid, España.

La organización SUMMA112 tiene como misión proveer asistencia sanitaria en las urgencias, emergencias, catástrofes y otras situaciones especiales. El escenario principal para la aplicación del demostrador es la gestión de los servicios de emergencias médicas, los que se ocupan de las tareas que proveen asistencia fuera de los hospitales a personas con repentinos problemas de salud.

Entre los servicios que pueden ser identificados como potenciales escenarios de aplicaciones para el soporte de las emergencias médicas se ha seleccionado el *transporte de urgencias* para el desarrollo del citado demostrador, implementándose el protocolo de SUMMA112. En todos los casos es un centro de emergencias el encargado de coordinar los recursos necesarios (ambulancias, hospitales, médicos, enfermeros, etc.) siguiendo su protocolo.

El prototipo está basado en una aplicación distribuida que provee soporte en la operación de los servicios médicos de emergencia, junto con mecanismos para coordinar ambulancias, reasignar las que se encuentren disponibles y asignar pacientes dentro de una determinada región.

La aplicación automatiza la comunicación entre las entidades, los procedimientos básicos de toma de decisiones y el intercambio de información médica, la que puede ser obtenida de fuentes remotas. En cuanto a la calidad general de la aplicación hay interés en analizar, realizar pruebas y evaluar diferentes mecanismos de organización y coordinación que podrían ayudar a mejorarla.

A estos efectos, se ha implementado una herramienta de simulación para realizar los experimentos con la aplicación del transporte de urgencias en entornos semi-realistas durante peródos de tiempo. Posteriormente se hará una breve descripción de la herramienta desarrollada como parte del Proyecto AT y que ha sido utilizada para este caso de estudio. En la Figura 6.1 puede apreciarse una interfaz representativa de la configuración del simulador *mHealth*. Posteriormente se darán más detalles del mismo.

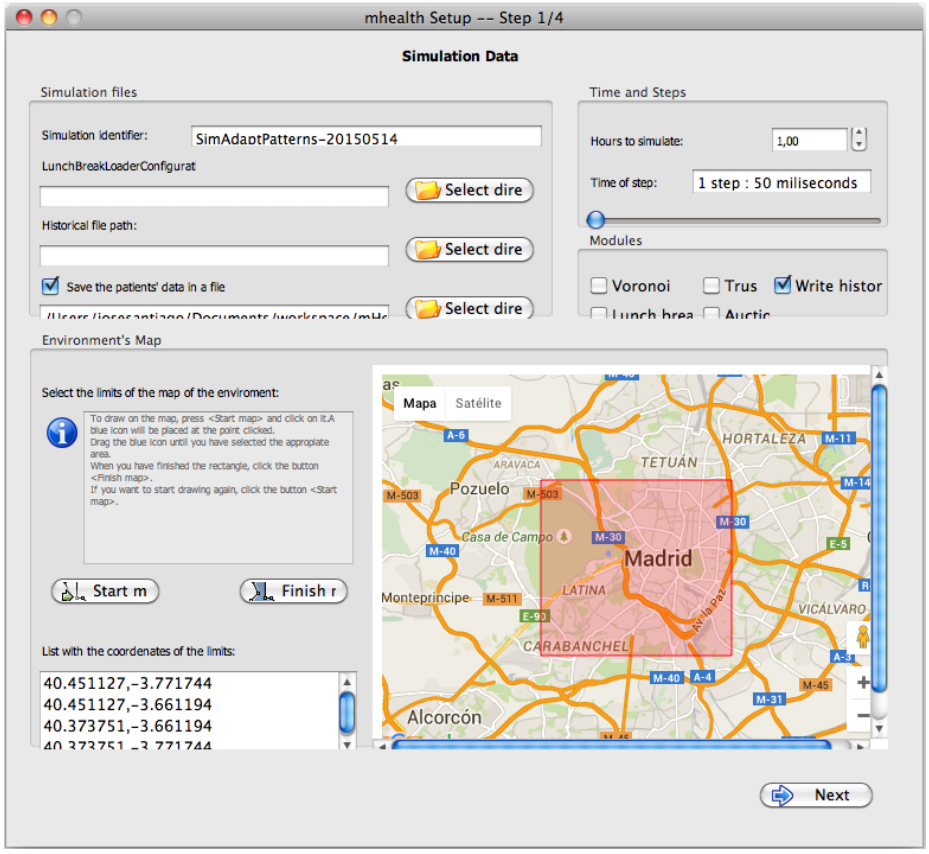

Figura 6.1: Interfaz de configuración del simulador del demostrador *mHealth*

El próximo paso es intentar conseguir la adaptabilidad real a los cambios del entorno durante los servicios de emergencia. Para ilustrar nuestra propuesta usaremos la herramienta de simulación que permite ciertas interacciones con el usuario.

A continuación describimos el escenario correspondiente a la prueba de concepto:

Se produce una crisis, un incendio de considerables proporciones, en un parque urbano situado al oeste de la ciudad de Madrid y aproximadamente veinte personas resultan heridas. La gestión de la emergencia está precisamente bajo la supervisión de SUMMA112 que ha enviado varias ambulancias al lugar y al menos tres ya han llegado al parque.

En esta situación el *patrón de adaptación* que corresponde activarse es el ***Gathering*** y, una vez activado a través de la herramienta, los agentes representados por las ambulancias están en condiciones de mantener conversaciones estructuradas componiendo un grupo in-

formal.

Para dar una estructura al grupo de trabajo se necesitaría un líder. En este caso el patrón ***Elect Surveyor*** puede ser disparado, y posteriormente ese líder activa el patrón ***Surveyor***. Utilizando el patrón ***Façade*** se elige una ambulancia para interactuar con el sistema *mHealth*. Cada vez que una ambulancia llega al lugar el sistema dispara el patrón ***Full Mesh*** para poder incorporarla al grupo, ya que se asume que las ambulancias no necesitan "traducción" para comunicarse. La *iniciativa* ya está formada pero no será necesario consolidarla porque la organización no está destinada a durar más de lo que dure la situación de crisis.

En el momento de producirse un cambio externo, como por ejemplo que el SUMMA112 requiera una o más de las ambulancias que puedan estar disponibles para hacer frente a otra situación, es de esperarse que el sistema reaccione con un comportamiento adaptativo.

Para representar ese cambio proponemos la siguiente situación: un accidente automovilístico se produce en un túnel de carretera de las inmediaciones. El lugar está en las cercanías de la primera crisis y al menos siete coches están involucrados en la tragedia. Desafortunadamente en ese momento el sistema no cuenta con ambulancias disponibles para actuar en el accidente. Recordemos que muchas están trabajando en la primera crisis mientras que las otras se encuentran realizando su labor habitual, las que no pueden anularse.

Teniendo presente esta situación, SUMMA112 solo cuenta con la alternativa de solicitar ambulancias a la *iniciativa* y lo hace a través de ***Façade***. El ***Surveyor*** acepta esta solicitud y en base a los datos actuales de la crisis define el número de ambulancias que pueden abandonar la *iniciativa*. Para tomar esa decisión puede tener en cuenta, por ejemplo, las ambulancias que ya están acabando su labor en ese lugar o que lo harán en breve porque tienen las menores cargas de trabajo. Para liberarlas se activa el patrón ***Retirement*** con el proceso que les permite abandonar la zona parquizada y trasladarse al lugar del accidente.

Cuando llegan a la nueva zona crítica el patrón ***Gathering*** es activado nuevamente, el ciclo comienza una vez más con el objetivo de formar otra *iniciativa*.

Es más que probable que los heridos necesiten tratamientos específicos, por ejemplo por las quemaduras, y deban ser trasladados a hospitales. En ese caso las ambulancias deben transportar a los pacientes y aquí también es activado el patrón ***Retirement*** como parte de la funcionalidad de la *iniciativa*.

Como el escenario propuesto es un ejemplo de urgencias médicas no resulta necesario consolidar la *iniciativa* ya que la crisis debe ser resuelta con la mayor rapidez posible. Las posibles organizaciones que podría formarse como resultado no van a perdurar en el tiempo después de que ambas crisis hayan sido solucionadas. Ninguna de las *iniciativas* habrá acabado como una *organización* "estable", por lo que el patrón ***Consolider*** no se ha tenido que utilizar.

Eventualmente una *iniciativa* puede ser extinguida con la activación del patrón ***Terminator*** cuando solo una ambulancia permanece en el lugar. En ese caso se lanza ese patrón y se termina con la *iniciativa*.

Para una mayor claridad, en la Figura 6.2 se presenta el diagrama de flujo de las pruebas realizadas con el simulador del demostrador *mHealth*. Este diagrama podría dividirse en tres partes, aunque solo para una mayor comprensión de su funcionamiento. Con la herramienta del *mHealth* es factible realizar las simulaciones clásicas del demostrador pero también se le puede añadir la posibilidad de lanzar simulaciones adaptativas.

La parte (I) se inicia con la configuración de las simulaciones, que en el caso adaptativo tendrá características específicas como se verá posteriormente en el detalle de las pruebas. A continuación se realiza el lanzamiento propiamente dicho de la simulación.

Ya en la parte (II) del diagrama, en cada paso de simulación se comprueba si se cumplen ciertas condiciones necesarias para el disparo del patrón correspondiente. En caso afirmativo se dispara el patrón, el que continuará con su funcionamiento previsto. En caso negativo la simulación debe avanzar un paso más.

Continuando por la rama afirmativa, se verifican nuevamente algunas condiciones, esta vez consultando si se debería producir la inhibición del patrón. En caso afirmativo el patrón se inhibe, se continúa con otro paso de simulación y con el bucle respectivo. En caso ne-

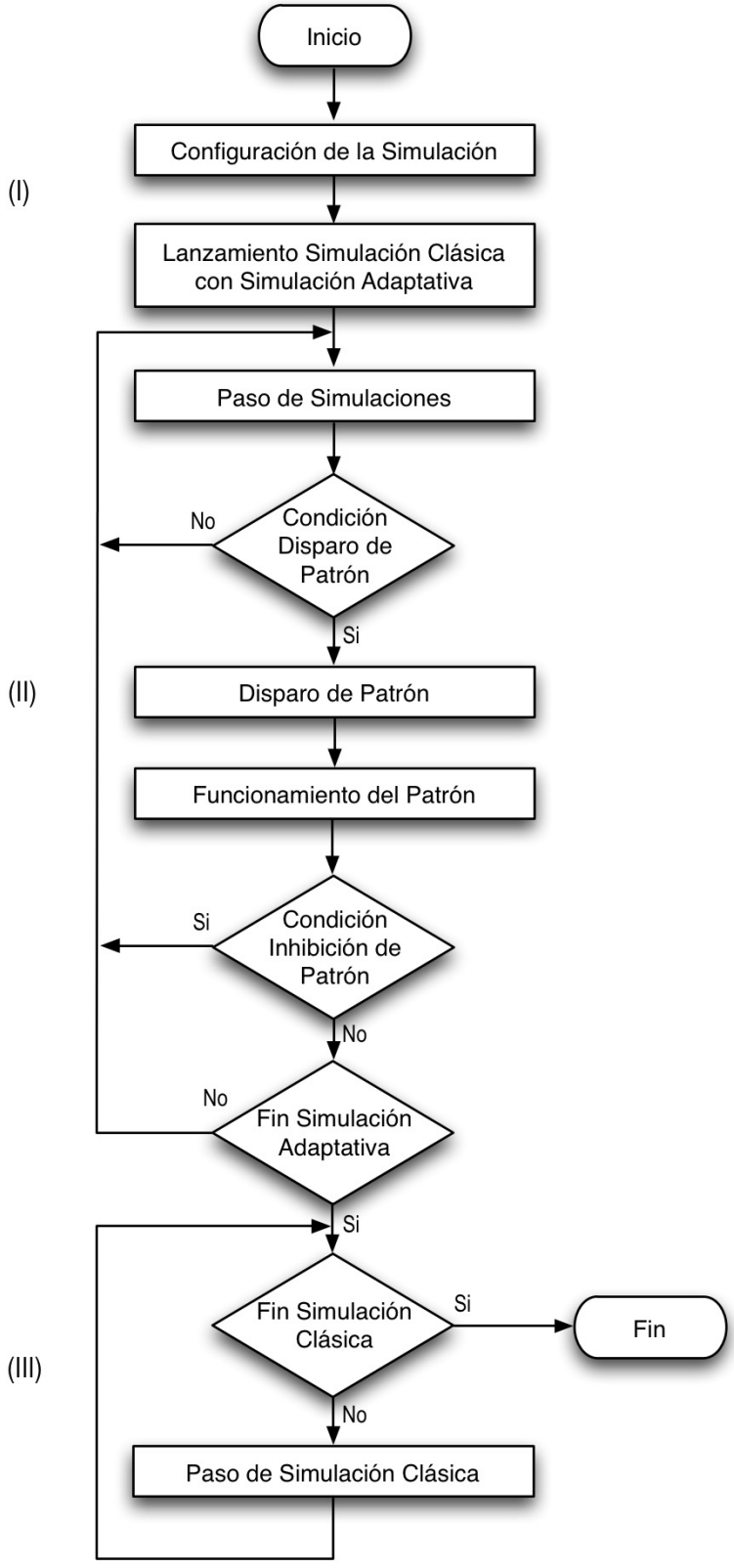

Figura 6.2: Diagrama de flujo de las pruebas de concepto realizadas con el simulador del demostrador *mHealth*.

gativo se precisa verificar si se produce condición de fin de la simulación adaptativa, por ejemplo que la *iniciativa* se consolide en una organización plena o deba eliminarse. En caso negativo la simulación continuará con un paso más.

En caso afirmativo, y ya en la parte (III) del diagrama, una vez que la simulación adaptativa ha finalizado y se han realizado sus últimos pasos, debe verificarse si existe condición de fin de la simulación clásica del *mHealth*, que normalmente es por tiempo. En caso negativo el simulador avanzará un paso más y en caso afirmativo se producirá el fin de la simulación.

Es importante notar que la herramienta de simulación no es totalmente adaptativa porque todos los patrones no se activan automáticamente. Como se ha dicho en 5.2.1, una posible implementación para el disparo de los patrones es utilizar tuplas en un espacio de tuplas, sin embargo en el contexto de las pruebas se han utilizado las condiciones específicas que esperábamos que generaran su disparo. Es decir, que parte del comportamiento está codificado en la herramienta ya que el dominio no es totalmente desconocido y se tiene la información suficiente para lanzar los patrones. En la sección siguiente se darán más de detalles de la herramienta de simulación y de los experimentos realizados para probar la utilidad del enfoque en la búsqueda de comportamiento adaptativo en las arquitecturas con la activación y desactivación de los patrones.

6.2. Proceso de validación

En este trabajo el proceso de validación ha sido realizado en base a la descripción del caso de estudio de la sección anterior. Básicamente el proceso ha contado con una serie de pasos, comenzando con la adecuación del simulador del demostrador *mHealth*.

Se ha tenido que modificar parte del código para que el comportamiento de la herramienta pudiera simular la situación de crisis propuesta.

También se ha tenido que adaptar la generación de pacientes para que estos se presenten en un area específica, con muy poca diferencia de tiempo entre ellos, y no en toda la región

de manera distribuida y en tiempos muy diferentes. El nivel de enfermedad es también relevante para los experimentos. Estos y otros detalles se explicarán posteriormente.

Cabe aclarar que la activación de los patrones de adaptación no ha sido totalmente automática, es decir que parte de su comportamiento también ha sido adaptado al contexto de las emergencias, ya que el dominio es conocido, como se dijo anteriormente. La dinámica de su funcionamiento ha sido explicada con más detalle en 5.6.

Como era de esperarse, los patrones de adaptación se han desplegado adecuadamente y los ejemplos realizados ayudaron a comprobar cómo un subconjunto de patrones pudieron ser combinados para obtener el comportamiento adaptativo del grupo preliminar: la *iniciativa*.

Es de notar que medidas de calidad o performance no se han tenido en cuenta en las pruebas, a diferencia del funcionamiento normal de las simulaciones del demostrador *mHealth*. Se ha tomado la decisión de no otorgarle relevancia en este trabajo porque el mayor énfasis está en verificar que los patrones son adecuados para lograr la adaptación del sistema ante cambios del entorno.

Seguidamente se hará una breve descripción de la herramienta de simulación utilizada y después se explicará una de las pruebas realizadas como ejemplo. Para mayores detalles del demostrador *mHealth* puede consultarse [20].

6.2.1. La herramienta de simulación

Las pruebas se han realizado utilizando como herramienta de simulación el demostrador *mHealth* [20], desarrollado en el marco del Proyecto Tecnologías del Acuerdo - AT [19].

Se ha utilizado como modelo de referencia el SUMMA112 [217], que es el Servicio de Urgencia Médica de la Comunidad de Madrid (España). Básicamente el demostrador simula la coordinación de las ambulancias que participan en las emergencias.

Las descripciones siguientes constituyen una síntesis de las características que pueden encontrarse en los documentos propios del Proyecto AT: el *deliverable* que trata sobre el análisis y diseño del demostrador (*D8.3.1 mHealth Requirements Analysis and Design*), y el *deliverable* que hace una revisión del prototipo (*D8.3.3 mHealth Medical Emergency*

Transportation Prototype Review). En primer lugar se describirá la aplicación y posteriormente el simulador.

La Aplicación se ha desarrollado como un SMA basado en organizaciones, por lo que los agentes interactúan entre sí para resolver de manera eficiente la gestión de las emergencias médicas.

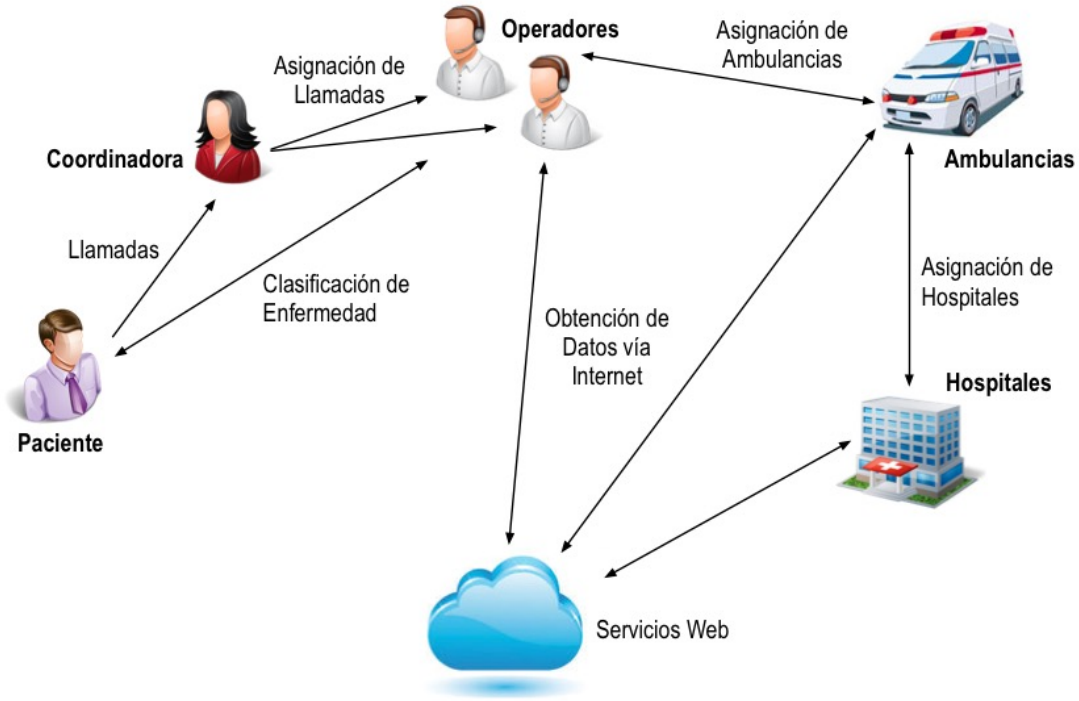

Figura 6.3: Visión general de la arquitectura de la aplicación

Los agentes intervinientes son racionales, perciben el entorno y actúan en consecuencia, teniendo en cuenta su estado interno y las acciones que les son propias. Tienen roles definidos dentro el sistema, que como puede verse en la Figura 6.3 son cinco: el *agente Paciente*, enfermo que se comunica con el centro coordinador; el *agente Centro Coordinador*, que es contactado por los pacientes y distribuye las llamadas de urgencias a los correspondientes operadores; el *agente Operador*, atiende las llamadas que le deriva el centro, realiza un primer diagnóstico del paciente y, de ser necesario, le asigna una ambulancia; otro rol es

el del *agente Ambulancia*, que es el único que tiene movimiento y al que se le asignan las misiones de asistencia al paciente, también recibe información del operador, acude al lugar donde se encuentra el enfermo, pudiendo asistirle allí mismo o transportarle a un hospital; por último el *agente hospital*, que informa al sistema de sus especialidades y de la admisión del paciente correspondiente.

Como es de esperarse, estos agentes (los actores del dominio) tienen la capacidad de comunicarse entre ellos y en la aplicación se producen determinadas secuencias de pasos e intercambios de mensajes realizados por los diferentes participantes del sistema. Aparte de estas características, también se utilizan servicios externos para obtener la información médica de los pacientes a través de Internet.

Para la implementación se utilizaron el lenguaje de programación Java, el *framework* JADE (Java Agent DEvelopment) para el SMA, SWT (Standard Widget Toolkit), Java Script, Restlet y Json. Más detalles al respecto, además de descripciones relacionadas con tareas, procesos de decisión, interacciones entre los agentes, etc. pueden encontrarse en los documentos ya citados.

El Simulador se utiliza para hacer pruebas, analizar y evaluar los diferentes mecanismos de coordinación y organización propuestos para mejorar la gestión del transporte en las emergencias médicas. Esta herramienta de simulación permite realizar experimentos configurando la aplicación con los parámetros semi-reales necesarios.

Su arquitectura puede verse en la figura 6.4. En ella se destacan claramente dos niveles, en el superior, la *capa de aplicación*, con la aplicación propiamente dicha, los agentes que participan en el escenario del transporte de emergencias, las estructuras que dan forma a las interacciones entre agentes y otros mecanismos utilizados para la coordinación.

El segundo nivel es la *capa de simulación*. Tiene cuatro componentes: el *simulador del entorno*, el *módulo de configuración de experimentos*, el *módulo de evaluación*, y la *interfaz gráfica de usuario*. El primero es el componente central y se encarga de simular un entorno que interactúa con la aplicación aportando realismo para los experimentos. Además provee los parámetros y valores de percepción para los agentes, ejecuta las acciones de estos y aplica los cambios al entorno que se generan. El segundo módulo permite configurar los experimentos con los parámetros correspondientes y el tercero recibe los datos obtenidos

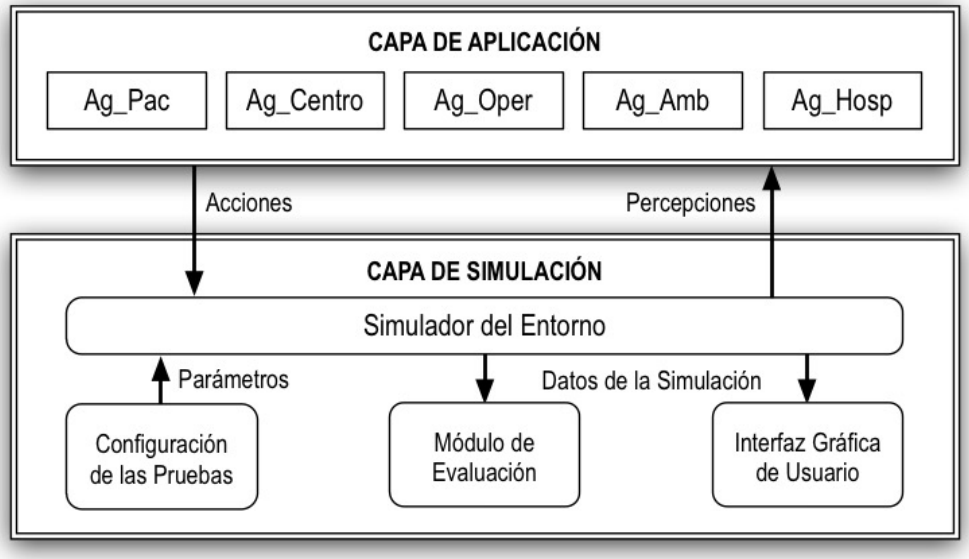

Figura 6.4: Arquitectura del simulador de la aplicación

y guardados durante la simulación, como la secuencia de estados del entorno, para realizar las evaluaciones correspondientes. Esos datos de la simulación también son recibidos por el cuarto módulo, y con ellos, gracias a la interfaz gráfica, el usuario puede observar lo que está ocurriendo durante las pruebas.

Las simulaciones se ejecutan de manera escalonada durante cierto período y cada paso de la simulación (*time step*) representa ciertos segundos de ese tiempo real. El simulador del entorno se encarga de generar los "estados del entorno" que contienen los parámetros importantes para la aplicación. Con el estado del entorno el simulador guarda un registro de todos los agentes y de las actividades de asistencia y transporte que ocurren en el sistema.

En el comienzo de la simulación este componente carga los parámetros, inicializa el sistema y crea los agentes. El proceso básico en cada paso de tiempo consiste en:

- Recibir las acciones que los agentes tienen que realizar en el sistema. En cada paso los agentes realizan una única actividad (sincronizada). Por ejemplo una llamada de un

paciente al centro coordinador, o que un agente envía un mensaje a otro como interacción.

- Simular la ejecución de las acciones en el entorno, cambiando los valores de los parámetros afectados. Las acciones se ejecutan secuencialmente, sin orden preciso y verificando la posibilidad o no de su ejecución en base al estado actual del entorno.

- Simular influencias externas en el entorno, ciertos cambios que no son resultado de las acciones de los agentes, sino por ejemplo de otros actores externos a la aplicación.

- Enviar el nuevo estado del entorno a los módulos de evaluación y de interfaz gráfica del usuario.

- Determinar las percepciones para los agentes como resultado del nuevo estado del entorno. Estas son posteriormente enviadas a cada uno de ellos en base a su rol en la aplicación.

Este ciclo se repite hasta que se acabe el tiempo de la simulación. De manera similar los agentes reciben sus percepciones, las procesan, actualizan su estado interno, determinan las próximas acciones a realizar y las envían al simulador del entorno para su ejecución. Este componente es utilizado como vía de ejecución de las acciones de los agentes y para paso de mensajes entre ellos.

Para su implementación se usaron, entre otros, el lenguaje de programación Java, Javadoc para la documentación, SWT para las interfaces gráficas y para la integración con Google Maps, necesaria para simular del movimiento de las ambulancias.

Con respecto a la *configuración* de las simulaciones, se puede elegir crear una nueva configuración, editar una existente, abrir un archivo de evaluación, etc. como se ve en la Figura 6.5.

Si creamos una nueva configuración las interfaces nos permitirán ingresar: los *datos de la simulación* (entre ellos su identificador, archivos históricos y de configuración, tiempo de duración, tiempo del paso y mapa o area de operación); los *parámetros del centro coordinador y de los pacientes* (número de operadores, de pacientes, niveles de enfermedad, distribución de pacientes, etc.); los *parámetros de las ambulancias* (identificador, capacidad para tratar distintos niveles de enfermedad, ubicación en el mapa, entre otros); y finalmente los *parámetros de los hospitales* (identificador, cantidad de profesionales y de camas en urgencias, servicios que provee, ubicación en el mapa, etc.).

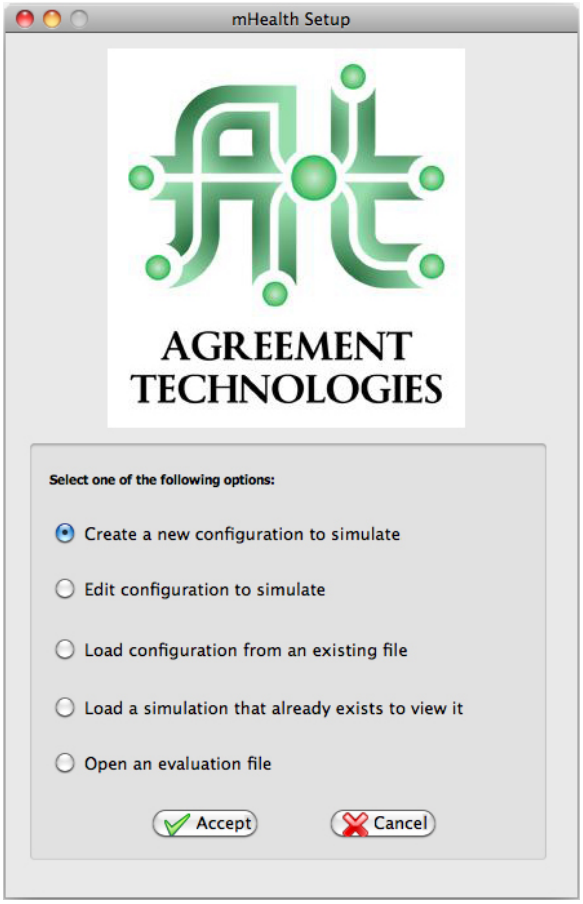

Figura 6.5: Interfaz para selección de opciones

El usuario puede elegir ver gráficamente lo que sucede durante el experimento, por ejemplo la aparición de los pacientes, el movimiento de las ambulancias, etc. En la Subsección siguiente (6.2.2), en la descripción de las pruebas, podrá verse cómo se van configurado estas pantallas para realizar nuestros experimentos.

El *módulo de evaluación* monitoriza la secuencia de estados del entorno de una simulación y genera tanto los datos relativos al experimento, como los específicos de cada paciente, ambulancia y hospital. Asimismo calcula valores de diferentes medidas de calidad y registra los resultados en una planilla de cálculo para evaluarlos a posteriori.

Es de destacar que para poder utilizar el simulador, y con el objetivo de llevar adelante nuestras pruebas, fue preciso realizar ciertas modificaciones a su código, por ejemplo para que al iniciar la simulación puedan ser lanzados los procesos necesarios para que tenga el comportamiento adaptativo buscado. Asimismo, a los agentes ambulancia se les ha agregado el código correspondiente para que cuando sean creados, estos cuenten con los procesos (concurrentes) que van a ser necesarios para el despliegue de los patrones. Por otro lado, también fue necesario que los agentes ambulancia puedan interactuar entre sí. Para ello se han debido adaptar sus interacciones al crearles canales de comunicación, ya que en base a su control los patrones de adaptación podrían ser disparados o cancelados.

Seguidamente se describe un ejemplo de los experimentos realizados, así como también las consideraciones que se han tenido en cuenta para que el simulador se adecuara a nuestro caso de estudio y permitiera realizar la prueba de concepto.

6.2.2. Realización de las pruebas

Para la prueba de concepto, como ya se ha adelantado, se utilizó el simulador del demostrador *mHealth*. El servicio de transporte de urgencias es el escenario que ha sido seleccionado para tal fin.

Buscando cierta correlación con las especificaciones en Cálculo-π presentadas en los patrones del capítulo anterior, en determinadas partes de la descripción de la prueba se citarán los métodos y procesos que se consideren apropiados para dotar de mayor claridad a la explicación.

Con la herramienta de simulación se han realizado varios ejemplos de uso de los patrones de adaptación con sus correspondientes configuraciones. A continuación se describirá uno de estos experimentos, que involucran los patrones Gathering, Surveyor, Retirement y Terminator.

En la configuración de la simulación se utiliza un solo centro coordinador, y aunque esta decisión implica cierta centralización en las decisiones iniciales, esta no influye en los resultados obtenidos porque los elementos principales para nuestras pruebas son las ambulancias que llegan a la zona en cuestión. Estas ambulancias (agentes) son los que van

a participar en el despliegue de los patrones de adaptacion, siguiendo las pautas establecidas en los mismos.

Con esta simulación puede verse cómo estos patrones son especialmente adecuados para lograr la coordinación y la auto-organización de un grupo de ambulancias.

El objetivo de este grupo está muy claro desde el principio, todos sus integrantes tienen la misma meta: resolver la situación de crisis en el menor tiempo posible.

Normalmente el simulador genera los pacientes en distintas posiciones del área de estudio, con diferentes niveles de enfermedad y en diferentes momentos del tiempo de simulación. Según el caso esto ocasionaría el envío de una ambulancia por cada paciente que se haya comunicado con el centro de coordinación. Sin embargo esto no es lo que precisamos para nuestra prueba de concepto, ya que son necesarias varias ambulancias, practicamente, en el mismo lugar, y no necesariamente una por cada llamada de pacientes.

En la vida real, un paciente involucrado o un testigo daría el alerta de la crisis sin esperar a estar herido o necesitar asistencia médica. Esta situación no formaba parte de los requerimientos para el desarrollo del simulador del mHealth, en el cual cada llamada representa un caso de emergencia independiente. En vista de esto se ha tenido que hechar mano de ciertos artificios para representar la crisis y conseguir algunas simplificaciones de esa situación.

El simulador tiene cierta flexibilidad para configurar los parámetros para cada simulación. En la Figura 6.1 puede apreciarse la primera de las cuatro interfaces para crear (o editar) la configuración de una simulación, las que se han explicado en la Subsección anterior (6.2.1). En esta interfaz se determina, entre otros datos, el tiempo de simulación, el que será de sesenta minutos para este experimento. También es posible definir el área de operación, el paso de simulacińn (*time step*) y seleccionar algunos módulos de coordinación, que para nuestras pruebas no ha sido necesario.

En la segunda interfaz se configuran los (agentes) centros de coordinación y los (agentes) pacientes. Para la generación de pacientes también pueden usarse datos desde un archivo (en formato XML). Esta facilidad nos permitió utilizar un archivo al que dotamos con las características necesarias para generar nuestra crisis. Es decir, había que modificar la forma en que se distribuían los pacientes en la zona de estudio y forzar a que lo hagan de

una manera más precisa. Las principales características eran:

- Las llamadas de los pacientes al centro coordinador están muy cercanas en el tiempo, tanto en la primera crisis como en la segunda, cierto tiempo después de aquella.

- Los lugares de aparición de los pacientes están muy cerca unos de otros, los valores de latitud y longitud de un paciente tienen poca variación con los de otro paciente. Al igual que con las llamadas al centro coordinador, se tendrá un grupo de valores similares para la primera crisis y otro grupo de latitudes y longitudes para la segunda.

- Se tuvo en cuenta pacientes en estado crítico, con niveles de enfermedad 0 y 1. Valores mayores indican menor gravedad. Con estos niveles de severidad los pacientes requerirán UVIs (Unidades de Vigilancia Intensiva), las que son ambulancias equipadas para cuidados intensivos.

Con los parámetros determinados de esta manera el objetivo es obtener una representación de crisis para que el simulador se adecúe a la situación buscada.

Como se ha explicado en la Sección 6.1, la crisis en primera instancia afecta a veinte personas, las que están involucradas directamente con el incendio declarado. Por lo tanto, veinte es el número de pacientes que deberán ser simulados para las pruebas.

Continuando con la tercera interfaz de configuración del simulador hemos especificado que participarán quince ambulancias, aunque no todas intervendrán en la resolución de la crisis, por lo que la *iniciativa* tendrá un número menor de integrantes. Se ha decidido que el sistema agregue de forma aleatoria ese número de agentes ambulancia.

La cuarta interfaz permite configurar los hospitales que son necesarios para la simulación clásica del demostrador, aunque para nuestro enfoque es suficiente con agregar aleatoriamente un cierto número de ellos.

Al lanzar la simulación, y mediante nuestro archivo XML de pacientes, estos comienzan a comunicarse con el centro coordinador y su aparición se produce tal como esperamos: muy cerca físicamente unos de otros, con poco tiempo de separación entre ellos y con los dos niveles de enfermedad previstos, el 0 y el 1.

Como una correspondencia con los formalismos definidos, el proceso Space (ver 5.5.3) será lanzado al inicio del experimento para que los procesos que dependen de él puedan ser

disparados a su vez.

A medida que avanza la simulación se van asignando ambulancias para atender a los pacientes. Cada vez que los agentes ambulancia son creados por el sistema se lanza un segundo proceso agente (al que podemos llamar "meta-agente") que se acopla al primero: se corresponde con el proceso *AgentX* de las especificaciones formales del Capítulo anterior.

Al cabo de cierto tiempo, la mayoría de los pacientes ya se han generado y se encuentran localizados en la zona de la crisis. Ya con el paciente *pat1*, el centro coordinador cumpliendo con su función programada comienza a enviar las ambulancias para hacer frente a la situación, como se puede apreciar en el tiempo de progreso de la simulación *9,67* de la Figura 6.6.

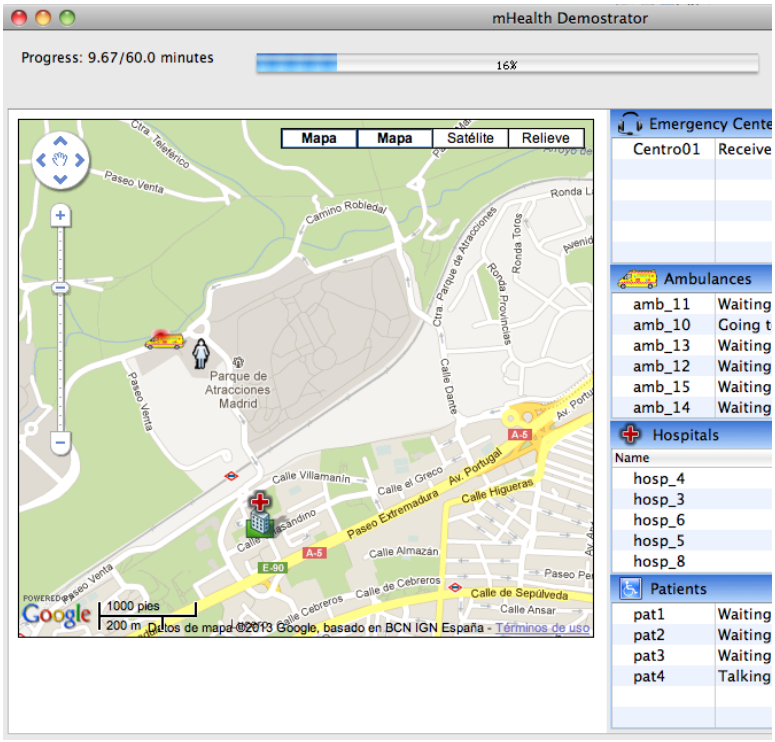

Figura 6.6: El centro coordinador envía la primera ambulancia

Con los parámetros provistos el centro coordinador identifica la situación de crisis y envía tres UVIs cercanas y disponibles. Esta condición detectada permite que con este número de agentes ambulancia ya sea posible utilizar el patrón de adaptación Gathering.

Además, con el control de las interacciones entre el agente centro coordinador y los agentes ambulancia se concreta dicho patrón.

Como se está trabajando con el simulador no se han implementado los patrones de una manera genérica sino que se han adecuado para utilizarse directamente en el caso de las ambulancias. Además, y como ya se ha dicho, el dominio no es absolutamente desconocido y se tienen los datos necesarios para el disparo de los patrones.

Teniendo en cuenta esto, al iniciar la simulación a cada agente ambulancia se le acopla un proceso (*AgentX*) que no interfiere en su funcionamiento sino que es utilizado para poder generar el comportamiento buscado con el patrón de adaptación. Asimismo se disparan los procesos *Venue* y *Connector* (ver 5.5.1) para que estas tres ambulancias puedan trabajar en conjunto, generar conversaciones y así formar parte de la *iniciativa*.

La zona de crisis está representada por el agente (proceso) *Venue*, y las ambulancias que se acercan a ese "espacio" (que representa más un espacio de cálculo que uno físico) tienen la posibilidad de formar parte del grupo preliminar. La Figura 6.7 representa el momento en que el patrón Gathering se concreta con tres ambulancias y en base a las condiciones del experimento (tiempo de progreso *15,42*).

Si seguimos el ciclo de vida propuesto en el Capítulo anterior (ver 5.7) el paso siguiente sería utilizar el patrón Elect Surveyor. Para una simplificación, teniendo en cuenta el dominio y considerando que no habrá pérdida de generalidad, en esta prueba el "líder" de esta *iniciativa* será la primera ambulancia que llega a la crisis, de esta forma no es necesario disparar el patrón Elect Surveyor.

Por definición del patrón Gathering, una vez que este haya cumplido su función los agentes ya están en condiciones de participar en conversaciones, se conocen, pueden interactuar y trabajar en conjunto. A continuación se puede disparar el patrón Surveyor porque el "líder" es conocido: la *amb_10* en la Figura 6.6 del experimento. Además este recibe, como otro resultado del Gathering, la lista de integrantes del grupo.

Entre las funciones que tiene el Surveyor (ver 5.5.3) este puede decidir cuál de los integrantes de la *iniciativa* puede actuar como una fachada para la interacción con el exterior

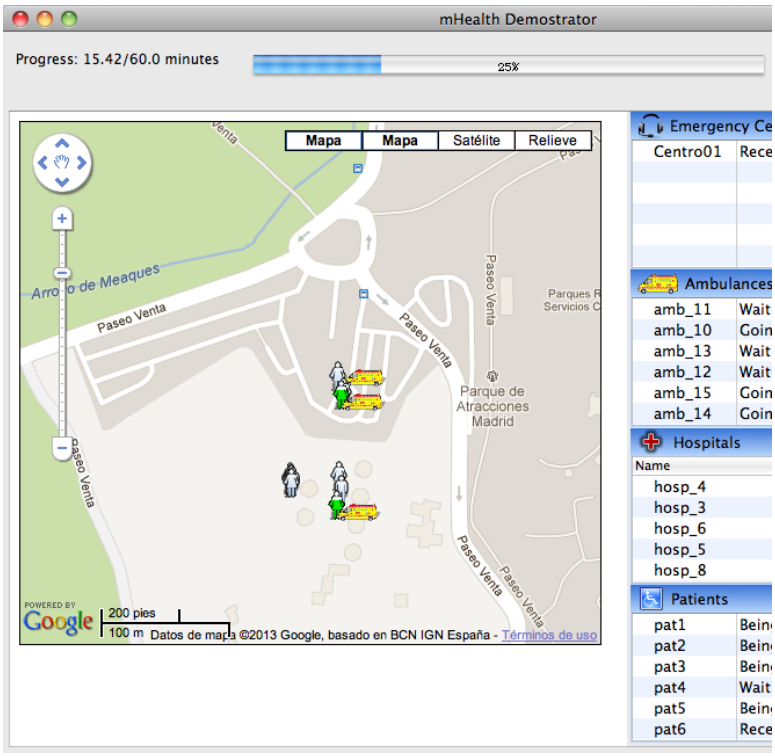

Figura 6.7: Patrón de adaptación *Gathering* activado

de la misma. Es así que *amb_10* decide que la ambulancia *amb_5* funcionará como tal, entonces pone en funcionamiento el patrón Façade (ver 5.5.5) y llama a su método *create_facade*. Asimismo le facilita la lista de capacidades de los integrantes.

La aplicación *mHealth*, a través de la fachada, se comunica con la *iniciativa* informando que se ha producido una segunda crisis. Esta vez se trata de un accidente de tráfico en un túnel cercano al parque de la primera *iniciativa* y el centro coordinador solicita alguna ambulancia que se encuentre disponible.

El Surveyor recibe de *amb_5* esta información y decide que la tercera ambulancia será la que acuda al accidente. Pone en funcionamiento el patrón Retirement (ver 5.5.12) que consiste básicamente en remover un miembro de la lista de integrantes del grupo con la llamada al método *remove_member* del patrón Surveyor. Este agente abandona el grupo, se dirige a la otra ubicación (*Venue*) donde una nueva *iniciativa* se estará formando y el ciclo vuelve a

comenzar.

Continuando con la simulación, a medida que transcurre el tiempo algunos pacientes son curados y el simulador los elimina de su lista de agentes pacientes activos. Como alternativa puede ocurrir que al requerir de cuidados más específicos se simule su traslado a un hospital. Es lo que sucede en nuestro experimento con el paciente *pat3* y su traslado lo realizará la ambulancia *amb_5*. Para ello el Surveyor disparará nuevamente el patrón Retirement y además, como ese agente también funciona como fachada de la *iniciativa*, la eliminará como tal llamando a su método *kill_facade*. Puede apreciarse esta situación en la Figura 6.8 y con el tiempo de progreso *35,92* de la simulación.

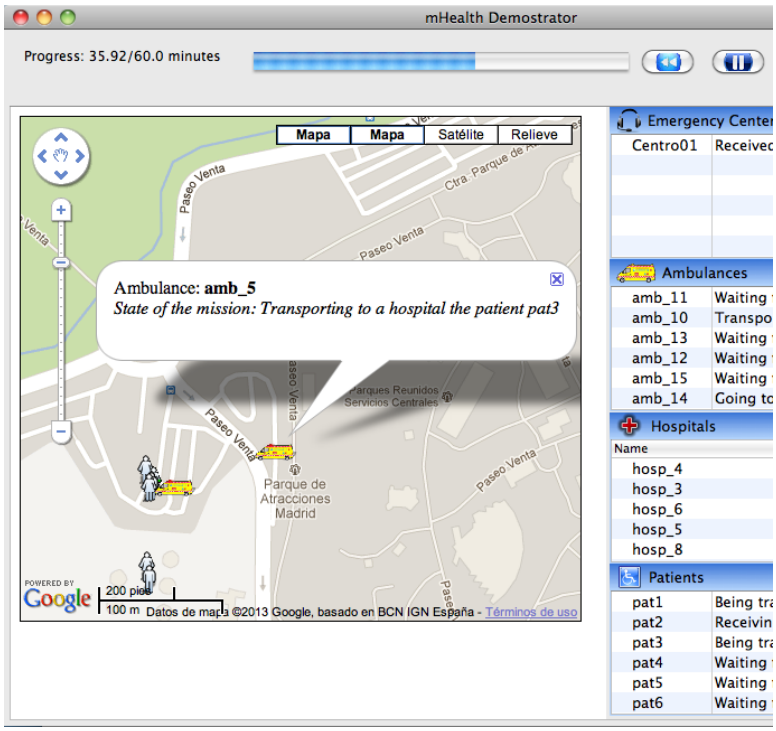

Figura 6.8: Patrón de adaptación *Retirement* activado

A partir de este momento el agente *amb_10* que, recordemos, representa al Surveyor es el único que ha quedado en la zona de crisis y continuará en funcionamiento en tanto dure esta. Sin embargo, y en base a uno de los prerequisitos del patrón Terminator (ver 5.5.17),

al quedar solo un agente en el grupo ese patrón debe ser disparado para dar por finalizada la *iniciativa*, ya que esta necesita al menos dos agentes involucrados para existir.

Si nos atenemos al formalismo del Calculo-π, el patrón Terminator se encarga de finalizar todos los procesos *AgentX* presentes, en este caso el proceso correspondiente al Surveyor.

Con la terminación de los procesos involucrados con la *iniciativa* lo que se consigue es que la construcción adaptativa deje de existir, pero sin que afecte a los agentes "reales" del SMA ya que estos continúan con su funcionamiento original. De esta manera se logra incorporar un mecanismo de adaptación completo, una capa adaptativa por encima del SMA. Al incorporarse procesos concurrentes e independientes a los agentes del sistema, estos continúan con su labor sin sufrir interferencias porque esos procesos pertenecen a la *iniciativa* y a los patrones de adaptación involucrados.

Como se propuesto en 6.1, al ser el escenario un ejemplo de urgencias médicas, en esta prueba no se ha consolidado la *iniciativa*. Incluso como se ha visto, esta ha dejado de existir al no tener más que un integrante. En este punto de nuestro ejemplo es importante destacar que sin la *iniciativa* el enfoque adaptativo puede considerarse cerrado, sin embargo el SMA sobre el que se han estado aplicando nuestros conceptos de adaptación prosigue con su labor "original". Es decir que la simulación del *mHealth* de nuestro experimento continúa hasta completar los sesenta minutos que han sido configurados.

Con la descripción del ejemplo se ha tratado de probar lo que fue en primera instancia la intención: en lugar de simular una organización AT clásica se han propuesto mecanismos para que se trabaje con una organización AT adaptativa, con un acuerdo adaptativo entre sus integrantes. A diferencia de una simulación clásica con el *mHealth*, en nuestro caso no es necesario que las emergencias simuladas tengan el cierre previsto durante el enfoque adaptativo. Lo realmente importante es destacar el disparo de los patrones, su ejecución, su despliegue y su posterior detención.

Se ha descrito el comportamiento del patrón como un fragmento de código que se ha

incorporado y que se ejecuta dentro del sistema. A continuación se ha visto que el comportamiento era el esperado, pero llegado a este punto fue necesaria nuestra intervención para que los patrones sean detenidos, ya que la condición de finalización de cada patrón por medio de *inhibidores* no ha sido desarrollado en profundidad. En el contexto de nuestro trabajo podríamos no considerarlos un tema de suma relevancia porque lo importante es el disparo y el despliegue de los patrones. En su realización práctica es un tema muy importante por lo que su estudio y desarrollo bien pueden constituir una línea de trabajo futuro.

Independientemente de los datos que provee el simulador, con nuestros experimentos se ha pretendido comprobar si en base a los pasos en la simulación los patrones de adaptación se van disparando, además de proveer y utilizar sus propios resultados. Básicamente, si pueden ser utilizados para conseguir el comportamiento emergente del grupo de agentes, en base a los objetivos de cada patrón.

Teniendo en cuenta los resultados, se ha visto que el sistema se comporta como se esperaba. Aún con las limitaciones que han sido encontradas, las pruebas, que incluso pueden considerarse no deterministas o concluyentes, sí que permiten llegar a la conclusión que el enfoque es viable y es posible conseguir comportamiento adaptativo en arquitecturas que tienen a las Tecnologías del Acuerdo como elemento principal de su construcción.

6.3. Conclusiones

En este capítulo se han presentado los resultados de la prueba de concepto de la propuesta de patrones de adaptación que hemos desarrollado.

El escenario del caso de estudio proviene del dominio de las emergencias, y aunque el ejemplo presentado es hipotético el mismo está basado en situaciones reales y se encuentra relacionado con el software demostrador *mHealth*. Este es un prototipo desarrollado en el marco del Proyecto Tecnologías del Acuerdo.

Una vez realizados los experimentos se ha podido comprobar cómo los patrones de adaptación se desplegaron de la manera esperada y pudo verse como se combinaron ade-

cuadamente para obtener el comportamiento adaptativo de la organización preliminar, la *iniciativa*.

Para las pruebas se han tenido en cuenta a las ambulancias como elementos principales de las simulaciones, las que obviamente se podrían extender con más parámetros y características, pero en realidad los experimentos se han realizado hasta el punto que consideramos óptimo para demostrar el comportamiento buscado. Es decir, que nuestro objeto en este caso no es el sistema en sí sino el cambio que se produce en el sistema.

Con este ejemplo no se pretende realizar una total demostración pero se ha probado que este enfoque puede funcionar. Se ve claramente en este ejemplo lo práctico y útil que sería un sistema de este tipo, y varias de las cuestiones que hay que decidir en el sistema se podrían definir fácilmente, por ejemplo con un conjunto de reglas.

En este capítulo hemos visto que nuestra propuesta puede funcionar, y bien, en sistemas reales, lo que nos permite destacar que los resultados actuales son prometedores y que la arquitectura adaptativa es posible.

Queda claro que hay muchos detalles que pueden mejorarse o también incluirse para obtener mejores resultados, pero el objetivo de esta tesis no era desarrollar esos pequeños detalles. Por el contrario la meta era probar que el esquema de patrones generales que hemos desarrollado funciona en este sistema y acompañar el marco teórico propuesto.

Con todo esto se ha obtenido una forma clara de integrar adaptación en sistemas de agentes, orientados a servicios y centrados en organizaciones. Claramente esto puede proveer y extender las capacidades adaptativas en el marco de las Tecnologías del Acuerdo.

Por lo tanto, se han alcanzado los objetivos que se pretendían en este capítulo, y en el próximo se presentarán las conclusiones finales de este trabajo de tesis y las futuras líneas de trabajo identificadas.

Capítulo 7

Conclusiones y Trabajos Futuros

A lo largo de este trabajo de tesis hemos estudiado la manera de integrar esquemas de auto-adaptación en sistemas de agentes de software con estructuras de organizaciones. Asimismo hemos confrontado este enfoque con el actual estado del arte de la materia, hemos realizado un planteamiento original basado en el concepto de *iniciativa* y una estructura desplegada como una serie de patrones de adaptación que desarrollan un ciclo de vida. Dicho ciclo ha sido representado por un lenguaje de patrones, el que además ha sido probado con experimentos en entornos semi-realistas.

A continuación vamos a resumir las aportaciones de esta tesis, las conclusiones y las posibles líneas de trabajo futuro.

Aportaciones del trabajo

En este trabajo se han explorado conceptos estructurales como base de un enfoque arquitectónico para proporcionar auto-adaptación a sistemas software. La idea clave es crear un contexto, también arquitectónico, en el que los agentes son coordinados y reorganizados por su inclusión en estructuras preliminares y luego compuestos en organizaciones más estables.

Los principales contribuciones que aporta este trabajo son:

- La definición conceptual de una *iniciativa*. Como se vio en el Capítulo 4 (4.2.2), a partir de una combinación de controles y protocolos utilizados por los agentes es posible generar dinámicamente un grupo preliminar. Este grupo emergente tiene una estructura provisoria y puede cambiar con la dinámica del entorno. De igual manera, como el enfoque propuesto es también una arquitectura orientada a servicios, los propios servicios pueden ser los puntos de partida para la definición funcional de estas estructuras preliminares o "protoorganizaciones". El concepto de *iniciativa* es considerado punto de partida para proveer mecanismos que permiten el cambio en la composición y de los tipos de elementos dentro de los sistemas auto-adaptativos. El dinamismo necesario puede ser soportado por un *acuerdo emergente* visto como una estructura arquitectónica evolutiva, basada en la combinación predefinida de controles y protocolos gestionados en el contexto del marco orientado a servicios, basado en agentes y centrado en organizaciones.

- Para llevar adelante la adaptación y obtener el comportamiento buscado se ha propuesto la utilización de lo que damos en llamar "elementos del cambio".

  Definimos a uno de ellos como *agente shifter*, que es aquel agente que tiene la capacidad de adaptar su comportamiento a los cambios de requerimientos, tanto del sistema como del entorno. Originalmente es un agente estandar pero que puede cambiar. Inclusive como resultado de las interacciones con sus pares estará capacitado para desempeñar funcionalidades concretas una vez que se le ha instruido para ello.

  Asimismo se ha definido otro elemento del cambio, el *agente changent*. En este caso, tanto si su origen es interno como externo al sistema, son los "agentes del cambio", ya que inducen el cambio en los agentes que sean necesarios. Para ello, por ejemplo, pueden habilitarles funcionalidades especiales del *middleware*, o también utilizar mecanismos para incentivarlos (o sancionarlos) en cuanto al uso de funcionalidades o cuando tienen ciertos roles, siempre tendiendo a modificar el comportamiento del sistema.

  Ejemplos de estos tipos de agentes pueden encontrarse en el Capítulo 5, cuando se presentaron las especificaciones en Cálculo-π de los protocolos de los patrones de

adaptación.

- La aportación más relevante de este trabajo consiste en el *catálogo de patrones de adaptación*. Como hemos analizado en el Capítulo 5, la *iniciativa* es el punto de partida para la generación de las *organizaciones basadas en acuerdos* que nos permiten obtener el *comportamiento emergente* en nuestras arquitecturas adaptativas. La *iniciativa* se genera a partir de estos patrones y a medida que evoluciona, en base al objetivo final, puede ir modificando su estructura e incluso su composición.

  Como es sabido un patrón no representa una solución real (final) sino que proporciona un modelo abstracto de solución reutilizable. La importancia del uso de nuestros patrones viene dada porque son utilizados desde el comienzo mismo del desarrollo de la solución. Recordemos por ejemplo que los agentes *changent* tienen la capacidad para desplegar un patrón completo, o varios de ellos, según que la *iniciativa* necesite llevar a cabo alguna modificación.

  Con la aportación de estos patrones de adaptación, y en base a las pruebas realizadas con el simulador *mHealth* presentado en 6.2.1, se ha conseguido que las estructuras estáticas presentes en el simulador puedan ser "transformadas" en dinámicas. Con la utilización de la información adicional proporcionada por los patrones se logra extender su potencial, comportamiento y expresividad.

Estos aportes se entienden como elementos necesarios para conseguir el comportamiento emergente de nuestras arquitecturas adaptativas, haciendo un amplio uso del concepto de las Tecnologías del Acuerdo. Los patrones propuestos favorecen la creación de las estructuras necesarias para lograr una auto-adaptación real. Los protocolos que definen los patrones, al ser representados formalmente con las especificaciones en Cálculo-π, permiten demostrar el prometedor enfoque y que el potencial de crecimiento es significativo.

Conclusiones

En el Capítulo 1 nos habíamos planteado una serie de objetivos para encaminar el trabajo de la tesis. En esta sección podremos verificar cómo cada uno de esos objetivos se han

podido cumplir.

En primera instancia se proponía realizar un estudio general relacionado con los temas del trabajo, el que ha sido desarrollado en el Capítulo 2. De igual manera con los sistemas adaptativos, auto-adaptativos, los que también se estudian y analizan en este Capítulo.

También se planteaba proporcionar un enfoque conceptual y tecnológico para definir esquemas de comportamiento general, y de esta forma precisar un ciclo de vida. Esto se presenta en el Capítulo 3, se desarrolla con mayor detalle y profundidad en el Capítulo 4, y se establece de manera definitiva en el Capítulo 5, y específicamente en la Sección 5.7.

Además de lo ya citado, otro objetivo era desarrollar un conjunto de patrones de adaptación que expresasen comportamientos concretos. Hemos definido conceptualmente a los patrones de adaptación en el Capítulo 4, y en el Capítulo 5 se han representado y desarrollado por completo todos los patrones.

Asimismo se proponía también probar que este enfoque podría funcionar en un esquema real, lo que fue llevado adelante con las pruebas realizadas en el Capítulo 6.

Por todo lo expuesto creemos haber resuelto satisfactoriamente los objetivos que se han planteado al inicio de este trabajo.

Líneas futuras

Como es habitual una tesis doctoral nunca puede considerarse finalizada. Siempre existirán temas y cuestiones que habrían sido interesante estudiar, por lo que a partir de este trabajo se pueden identificar futuras líneas de investigación. Seguidamente presentamos el listado de dichas líneas.

- Uno de los objetivos de esta tesis ha sido demostrar que los patrones de adaptación propuestos pueden ser vistos como “simiente” de las estructuras que disparan un

comportamiento, el que es requerido para inducir la real auto-adaptación, demostrando su utilidad y poder expresivo. Una vez dicho esto queda claro que una futura línea de trabajo es la implementación de los patrones y de las variantes de este enfoque en el marco de la plataforma de agentes THOMAS [13], la que ha sido seleccionada para dar soporte a esta tesis, también con el aporte del proyecto OVAMAH [200] y siempre en el contexto de las Tecnologías del Acuerdo.

- La auténtica arquitectura adaptativa que se pretende puede ser viable ya que la infraestructura tiene la capacidad de crecer añadiendo nuevos patrones de adaptación. Esto puede lograrse ampliando el número de patrones, identificando y clasificándolos en base a los nuevos requerimientos que surjan a medida que la investigación avance o también cuando se realice el refinamiento de los actuales patrones de adaptación.

- La utilización de los patrones de adaptación en conjunto con el simulador *mHealth* ha sido de manera prácticamente manual, es decir que se ha tenido una evolución dirigida. Esto justifica otra línea de trabajo futuro como es la continuación de la investigación hacia una reconfiguración y adaptación automáticos.

- Como otra línea de trabajo, también se pretende profundizar en el estudio y desarrollo de los mecanismos de disparo e inhibición de los patrones de adaptación para lograr mayor automatización en el despliegue y funcionamiento de dichos patrones.

- En este trabajo se ha asumido que los agentes que participan en los sistemas propuestos para el estudio tienen características benevolentes, son cooperativos, buscan la solución de los problemas sin generar conflictos, etc. Una línea de trabajo que también puede enriquecer el enfoque de adaptación propuesto sería la utilización de agentes que no sean colaborativos, con comportamientos egoístas o interesados.

- En el Capítulo 5 se han representado formalmente los protocolos que definen cada uno de los patrones de adaptación utilizando el Cálculo-π para su especificación. Describir formalmente la arquitectura de un sistema software es proveer la abstracción conceptual para modelarlo, por lo que la elección del ADL es un factor importante a tener en cuenta. Investigar la utilidad y el posible uso posterior del lenguaje

π-ADL [167] para realizar la descripción de nuestra arquitectura es una tarea que representa un interesante desafío. Este lenguaje ha sido diseñado para describir arquitecturas teniendo en cuenta tanto el punto de vista estructural como el de comportamiento. Con él es factible generar especificaciones de arquitecturas estáticas, dinámicas y también móviles, según los autores.

Por supuesto que estas no son las únicas lineas de investigacion, habría muchas más, ya que ciertos aspectos puntuales de esta tesis podrían ser detallados como líneas de trabajo, pero con las presentadas creemos haber desarrollado las principales.

Apéndice A

Conclusions

Throughout this researh for this PhD thesis we have studied the way to integrate self-adaptive schemes with organization oriented multi-agent systems. We also have confronted this approach with the state-of-the-art of the subject. We have made an original proposition based on the concept of the *initiative* and we also have developed certain structures as a set of adaptation patterns which develop a life cycle. This life cycle has been depicted by a pattern language, which has been tested in experiments on semi-realistic environments.

Next, we present the contributions of this thesis, some conclusions and certain future lines of research.

Contributions

We have explored structural concepts as the basis of an architectural approach to provide self-adaptivity to complex software systems. The key idea is to create an architectural context in which agents are coordinated and organized in preliminary structures, and then they are composed in more stable organizations.

Main contributions of this work are summarized as follows:

- The concept of the *initiative*. In Chapter 4 (4.2.2) we could see that, depending on concrete goals, software agents can be arranged dynamically into a preliminary group by using two different kind of mechanisms: controls and protocols. This emergent

group has a primary structure and it grows with the environmental dynamics; it is not yet fully established and evolves. The required dynamism can be supported by an evolving architectural structure in the context of the service-oriented, agent-based and organization-centric framework defined by Agreements Technologies. Even high-level services can be proposed as the starting point for the functional definitions of these preliminary structures, or "proto-organizations". The *initiative* can provide necessary mechanisms to allow the changes of composition and the type of elements into self-adaptive systems. The evolving architectural structure can be supported by an emergent agreement, based on combining predefined controls and protocols in order to obtain adaptation patterns.

- In order to obtain the desired adaptation and behaviour, in this work we have proposed what we called "elements of change".

  First, we have defined the *shifter agent*. A *shifter agent* is able to adapt its behaviour according to requirement changes in the environment or in the system itself. In the beginning it can be a standard agent but it is able to change. Even as a result of interactions between pairs it could perform a concrete functionality once it has been instructed to do so.

  We also have defined the *changent agent*, which is another element of change. Whatever its origin, internal or external to the system, they are "agents of change" because they induce changes to another agents. They can allow, for example, special *middleware* functionalities to some agents, and (or) use certain mechanisms to incentivize (or punish) them, when they use some functions or play certain roles.

  Some examples of *shifter agents* and *changent agents* can be found in Chapter 5, where we described the protocols for adaptation patterns in π-Calculus specification.

- The main contribution of this thesis is the *adaptation patterns catalog*. As we analized in Chapter 5, the *initiative* is the starting point for our *agreement-based organizations* in order to obtein *emergent behaviour* of our adaptive architectures. The *initiative* can be generated from these patterns, and during its evolution is able to modify its structure, even its composition.

It is already known that a pattern does not represent an actual (final) solution, but provides an abstract model of a reusable solution. The use of our *adaptation patterns* from the beginning of the solution development shows how important can be. Remember, for example, that a *changent agent* is able to deploy a complete pattern, or various of them, in case the *initiative* needs some kind of modification.

With the contribution of our *adaptation patterns*, and based on tests with the *mHealth* simulator (6.2.1), we have achieved that static structures from the simulator can be "transformed" into dynamic structures. With additional information provided by the patterns we can extend their potencial, behaviour and expressiveness.

We believe that these contributions are neccesary elements in order to achieve the desired emergent behaviour of our adaptive architectures, making an extensive use of concepts from Agreement Technologies.

The patterns we have proposed encourage the creation of necessary structures to achieve actual self-adaptation. The protocols defining the patterns, formally specified by π-Calculus, demonstrate that this approach is very promising and its potential is very significative.

Conclusions

In Chapter 1 a set of objectives has been proposed for this thesis research. We can now verify in this section how well this objectives have been accomplished.

We proposed, in the first place, make a general study related with this thesis subject. The study of the state-of-the-art has been developed in Chapter 2. Also adaptive systems and self-adaptive systems have been studied in this chapter.

We also proposed to provide a conceptual and technological approach in order to define general behaviours schemes, and also define a life cycle with them. These are presented in Chapter 3, they have a detailed developed in Chapter 4 and a deep study in Chapter 5, specifically in Section 5.7.

Another objective was to develop a set of adaptation patterns to express concrete behaviours. We have defined them conceptually in Chapter 4, and we have depicted and developed the whole set of all the patterns in Chapter 5.

We also proposed to make some experiments to prove the approach in certain real situations. The proof of concept has been achieved in Chapter 6.

For all these reasons we have successfully resolved the objectives that have been proposed at the beginning of this work.

Future work

Some research lines for future work have been identified from this thesis.

- One of the main goals of this thesis has been to demonstrate that the proposed adaptation patterns can be seen as "seed" of the structures which trigger the (desired) behaviour, required to induce actual self-adaptivity. They have already proven its utility and expressive power. In fact, it is clear that the implementation of the patterns and variants of this approach in the THOMAS [13] framework have to be a future work, sopported also by the OVAMAH project [200] into the context of AT.
- The desired adaptive architecture is indeed feasible because the infrastructure is able to grow just adding new adaptive patterns. This could be achieved, as future work also, extending the number of patterns as a result of new requirements from research advances or refining existing adaptation patterns.
- Use adaptation patterns along with *mHealth* simulator was basically a manual task, obtaining a directed evolution. So another future work line is to continue the research towards an automatic reconfiguration and adaptation.
- As another future line, we also pretend to deepen the study and development of trigger mechanisms and inhibition of adaptation patterns, for greater automation in the deployment and operation of these patterns.

- It could be interesting to study our adaptation approach using another kind of agents, not only collaborative or cooperative. With greedy agents, for example, the application domain could be very different and powerful.

- We formally specified in π-Calculus the protocols defining patterns in Chapter 5. A formal description of software system architecture provides a conceptual abstraction for its model, so selecting an appropriate ADL is an important issue. A feasibility study and later use of the π-*ADL* language [167] to formally describe our architecture could be an interesting challenge. This ADL has been designed to formally describe architectures from the structural and behavioural viewpoints. This language can address specification of static, dynamic and mobile architectures, according with the authors.

These are not the only future research lines, of course. There will be much more interesting details to study, but we think that the presented ones are the most importants.

Apéndice B

Publicaciones

Como parte del trabajo de investigación de esta Tesis Doctoral se ha contribuido con numerosas publicaciones en foros tanto nacionales como internacionales, las que se presentan a continuación.

- Perez-Sotelo, J. S., Cuesta C. E., y Ossowski S.: *The Agreement as an Adaptive Architecture for Open Multi-Agent Systems*, II Taller de Sistemas Autónomos y Adaptativos (WASELF2009), San Sebastian, España. Actas II WASELF, SISTEDES, vol. 3, no. 4, pp. 62-76. ISSN 19883455, 2009.

- Perez-Sotelo, J. S., Cuesta C. E., y Ossowski S.: *El Acuerdo como Arquitectura Adaptativa para Sistemas Multiagente Abiertos*, XV Congreso Argentino de Ciencias de la Computación (CACIC) - VI Workshop Ingeniería de Software (WIS), San Salvador de Jujuy, Argentina. Actas XV CACIC, ISBN 9788972406841, 2009.

- Perez-Sotelo, J. S., Cuesta C. E., y Ossowski S.: *Agreement Technologies for Adaptive, Service-Oriented Multi-Agent Systems*, II Workshop on Agreement Technologies (WAT2009), Sevilla, Spain. CEUR Workshop Proceedings, vol. 635. ISSN 16130073, 2009.

- Perez-Sotelo, J. S., Cuesta C. E., y Ossowski S.: *El Papel de las Tecnologías del Acuerdo en la Definición de Arquitecturas de Software Adaptativas*, XI Simposio

Argentino de Ingeniería de Software (ASSE2010) - 39ª Jornadas Argentinas de Informática (JAIIO2010), Buenos Aires, Argentina. Anales 39ª JAIIO, ISSN 18502776, 2010.

- Perez-Sotelo, J. S., Cuesta C. E., y Ossowski S.: *Auto-Adaptación en Organizaciones de Agentes: de la Iniciativa al Acuerdo Emergente*, III Taller de Sistemas Autónomos y Adaptativos (WASELF2010) - XV Jornadas de Ingeniería del Software y Bases de Datos (JISBD2010), Valencia, España. Actas III WASELF, SISTEDES. ISSN 19883455, 2010.

- Perez-Sotelo, J. S., Cuesta C. E., y Ossowski S.: *Towards Adaptive Service Ecosystems with Agreement Technologies*, I Workshop Adaptation in serVice EcosYsTems y ArchiTectures (AVYTAT2010) - OnTheMove Federated Conferences 2010, Creta, Grecia. LNCS vol. 6428, pp. 77-87. ISBN 9783642169601, Springer, 2010.

- Cuesta, C. E., Perez-Sotelo, J. S., y Ossowski S.: *Self-Organising Adaptive Structures: the Shifter Experience*, Journal ERCIM-News, European Research Consortium for Informatics y Mathematics, no. 85, pp. 35-36. ISSN 09264981, 2011.

- Perez-Sotelo, J. S., Cuesta C. E., y Ossowski S.: *The Role of Agreement Technologies in the Definition of Adaptive Software Architectures*, SADIO Electronic Journal of Informatics y Operations Research, vol. 10. ISSN 15146774, 2011.

- Billhardt, H., Centeno R., Cuesta C. E., Fernandez A., Hermoso R., Ortiz R., Ossowski S., Perez-Sotelo J. S., y Vasirani M.: *Organisational Structures in Next-Generation Distributed Systems: Towards a Technology of Agreement*, Journal Multiagent and Grid Systems, vol. 7, no. 2-3, pp. 109-125. ISSN 15741702 (Print)/18759076 (Online), IOS Press, 2011.

- Perez-Sotelo, J. S., Cuesta C. E., y Ossowski S.: *Adaptation Patterns in Multi-Agent Architectures: the Gathering Pattern*, II Workshop on Variability, Adaptation y Dynamism in software systEms y seRvices (VADER2011) - OnTheMove Federated Conferences 2011, Creta, Grecia. LNCS vol. 7046, pp. 657-661. ISBN 9783642251252, Springer, 2011.

- Perez-Sotelo, J. S., Cuesta C. E., y Ossowski S.: *Adaptation Patterns for Organisation-Based, Multi-Agent Architectures in Agreement Technologies*, Proceedings 9th European Workshop on Multi-agent Systems (EUMAS2011), Maastricht, Holanda, 2011.

- Perez-Sotelo, J. S., Cuesta C. E., y Ossowski S.: *Patrones de adaptación para arquitecturas de software basadas en tecnologías del acuerdo*, REICIS Revista Española de Innovación, Calidad e Ingeniería del Software, vol. 7, no. 3, pp. 6-25. ISSN 18854486, ATI-España, 2011.

- Perez-Sotelo, J. S., Cuesta C. E., y Ossowski S.: *Agent Adaptive Organisations with Agreements Technologies*, Proceedings 1st International Conference on Agreement Technologies (AT2012), Dubrovnik, Croacia. CEUR Workshop Proceedings, vol. 918. ISSN 16130073, 2012

- Perez-Sotelo, J. S. (co-autor): Capítulo *Adaptive Agent Organisations*. En S. Ossowski (Ed.), Agreement Technologies. Law, Governance and Technology Series, Vol. 8, Springer Verlag, pp. 328-332, ISBN 9789400755826, 2013.

- Perez-Sotelo, J. S., Cuesta C. E., Billhardt, H. y Ossowski S.: *Lifecycle of Adaptive Agreements: a Pattern Language*, Proceedings 2nd International Conference on Agreement Technologies (AT2013), Beijing, China. LNCS - LNAI 8068, ISBN 9783642398599, Springer, 2013.

Referencias

[1] Agents@RMIT. Prometheus Design Tool. http://www.cs.rmit.edu.au/agents/pdt/, febrero 2013.

[2] UMBC AgentWeb. Kqml - knowledge query and manipulation language. http://www.csee.umbc.edu/csee/research/kqml/, febrero 2013.

[3] O. Zohreh Akbari. A survey of agent-oriented software engineering paradigm: Towards its industrial acceptance. *Journal of Computer Engineering Research*, 2:14–28, abril 2010.

[4] L. D. Alahakoon, S. K. Halgamuge, and B. Sirinivasan. Dynamic self-organizing maps with controlled growth for knowledge discovery. In *IEEE Transactions on Neural Networks, Special Issue on Knowledge Discovery and Data Mining*, volume 11, pages 601–614. IEEE Computer Society, 2000.

[5] Juan M. Alberola, Jul, and Ana Garcia-Fornes. Cost-aware reorganization service for multiagent systems. In *Proceedings of ITMAS 2011*, pages 1–16, 2011.

[6] H. Aldewereld, L. Penserini, Frank Dignum, and Virginia Dignum. Regulating organizations: The ALIVE approach. In *Proceedings of the International Workshop on Regulations Modelling and Deployment (ReMoD 2008/CAiSE 2008)*, pages 37–48, Montpellier, Francia, junio 2008.

[7] Christopher Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[8] Christopher Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[9] Fernando Alonso, José L. Fuertes, Loïc Martínez, and Héctor Soza. Measuring the social ability of software agents. In Walter Dosch, Roger Y. Lee, Petr Tuma, and Thierry Coupaye, editors, *SERA*, pages 3–10. IEEE Computer Society, 2008.

[10] Jesper Andersson, Rogério de Lemos, Sam Malek, and Danny Weyns. Modeling dimensions of self-adaptive software systems. In Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*. Springer-Verlag Berlin Heidelberg, 2009.

[11] AOS. Jack intelligent agents development environment manual. http://www.aosgrp.com/documentation/jack/JDE_Manual_WEB/index.html, febrero 2013.

[12] Estefanía Argente. *GORMAS: Guidelines for ORganization-based Multiagent Systems*. PhD thesis, Universidad Politécnica de Valencia, June 2008.

[13] Estefania Argente, Vicente Botti, Carlos Carrascosa, Adriana Giret, Vicente Julian, and Miguel Rebollo. An abstract architecture for virtual organizations: The THOMAS project. Technical report dsic-ii/13/08, DSIC, Universidad Politécnica de Valencia, Valencia, Spain, 2008.

[14] Estefania Argente, Vicente Botti, Carlos Carrascosa, Adriana Giret, Vicente Julian, and Miguel Rebollo. An abstract architecture for virtual organizations: The THOMAS approach. *Knowl. Inf. Syst.*, 29(2):379–403, November 2011.

[15] Estefanía Argente, Vicente Botti, and Vicente Julián. GORMAS: An organizational-oriented methodological guideline for open MAS. In Marie-Pierre Gleizes and Jorge Gómez-Sanz, editors, *Agent-Oriented Software Engineering X*, volume 6038 of *LNCS*, pages 16–23. Springer Berlin/Heidelberg, 2011.

[16] Estefania Argente, Adriana Giret, S. Valero, Vicente Julián, and Vicente Botti. Survey of mas methods and platforms focusing on organizational concepts. *Frontiers in Artificial Intelligence and Applications*, pages 309–316, 2004.

[17] Estefania Argente, Vicente Julian, and Vicente Botti. Multi-agent system development based on organizations. *Electronic Notes in Theoretical Computer Science*, 150(3):55–71, May 2006.

[18] Alexander Artikis, Dimosthenis Kaponis, and Jeremy Pitt. *Dynamic Specifications of Norm-Governed Systems*, chapter 19, pages 460–479. IGI Global, marzo 2009.

[19] AT. Agreements Technologies Project. http://www.agreement-technologies.org, septiembre 2015.

[20] AT. Demonstrator mHealth - Agreements Technologies Project. http://www.agreement-technologies.org/demos, septiembre 2015.

[21] Manuel Atencia and Marco Schorlemmer. I-ssa: Interaction-situated semantic alignment. In *Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems:*, OTM '08, pages 445–455, Berlin, Heidelberg, 2008. Springer-Verlag.

[22] Daniel Austin, Abbie Barbir, Christopher Ferris, and Sharad Garg. *Web Services Architecture Requirements*. W3C WSA Working Group, W3 Consortium, 2004.

[23] Luciano Baresi, Sam Guinea, and Giordano Tamburelli. Towards decentralized self-adaptive component-based systems. In *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS'08)*, pages 57–64, New York, NY, USA, 2008. ACM.

[24] F.L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*, volume 7 of *Wiley Series in Agent Technology*. Wiley, 2007.

[25] Jenifer Pérez Benedí. *PRISMA: Aspect-Oriented Software Architectures*. PhD thesis, Universidad Politécnica de Valencia, Valencia, España, diciembre 2006.

[26] Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, editors. *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Kluwer Academic Publishers, 2004.

[27] C. Bernon, M.P. Gleizes, S. Peyruqueou, and G. Picard. ADELFE: A methodology for adaptive multi-agent systems engineering. In P. Petta, R. Tolksdorf, and F. Zambonelli, editors, *Proceedings of the 3rd International Workshop on Engineering Societies in the Agents World (ESAW'02)*, pages 156–169, 2002.

[28] Holger Billhardt, Roberto Centeno, Carlos E. Cuesta, Alberto Fernández, Hermoso Ramón, Rubén Ortiz, Sascha Ossowski, José S. Pérez-Sotelo, and Matteo Vasirani. Organisational structures in next-generation distributed systems: Towards a technology of agreement. *Multiagent and Grid Systems*, 7(2-3):109–125, 2011.

[29] Holger Billhardt, Roberto Centeno, Alberto Fernández, Hermoso Ramón, Rubén Ortiz, Sascha Ossowski, and Matteo Vasirani. On the relevance of organizational structures for a technology of agreement. In *OTM 2008 Workshops*, number 5333 in LNCS, pages 138–149, Monterrey, Mexico, noviembre 2008. Springer-Verlag Berlin Heidelberg.

[30] Karun N. Biyani and Sandeep S. Kulkarni. Mixed-mode adaptation in distributed systems: A case study. In *Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'07)*, page 14, Washington, DC, USA, mayo 2007. IEEE Computer Society.

[31] Guido Boella, Pablo Noriega, Gabriella Pigozzi, and Harko Verhagen, editors. *Normative Multi-Agent Systems (Dagstuhl Seminar Proceedings 09121)*, number 09121 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[32] Guido Boella, Gabriella Pigozzi, and Leendert van der Torre. Normative systems in computer science - ten guidelines for normative multiagent systems. In Guido Boella, Pablo Noriega, Gabriella Pigozzi, and Harko Verhagen, editors, *Normative Multi-Agent Systems*, number 09121 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[33] Olivier Boissier, Jomi F. Hübner, and Jaime S. Sichman. Organization oriented programming: From closed to open organizations. In *Engineering Societies in the Agents World VII*, volume 4457 of *LNCS*, pages 86–105. Springer-Verlag, 2007.

[34] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. *Web Services Architecture*. W3C WSA Working Group, W3 Consortium, February 2004.

[35] Rodney A. Brooks. Intelligence without reason. In Ray Myopoulos, John; Reiter, editor, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 569–595, Sydney, Australia, August 1991. Morgan Kaufmann.

[36] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. *Engineering Self-Adaptive Systems through Feedback Loops*, volume 5525 of *LNCS*, pages 48–70. Springer-Verlag Berlin Heidelberg, marzo 2009.

[37] Arne Brutschy, Giovanni Pini, Carlo Pinciroli, Mauro Birattari, and Marco Dorigo. Self-organized task allocation to sequentially interdependent tasks in swarm robotics. *Autonomous Agents and Multi-Agent Systems*, pages 1–25, 2012.

[38] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley Publishing, Chichester, UK, 1996.

[39] Lawrence Cabac and Daniel Moldt. Formal semantics for AUML agent interaction protocol diagrams. In *Proceedings of AOSE 2004*, pages 47–61. Springer, 2004.

[40] Carlos Canal, Juan Manuel Murillo, and Pascal Poizat. Software adaptation. In *Proceedings of the 1st International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'04)*, pages 9–31, Oslo, Norway, junio 2004. Springer.

[41] D. Capera, J. P. Georgé, M. P. Gleizes, and P. Glize. Emergence of Organisations, Emergence of Functions. In *AISB'03 Symposium on Adaptive Agents and Multi-Agent Systems, University of Wales, Aberystwyth*. Society for the Study of Artificial Intelligence and the Simulation of Behaviour, 2003.

[42] D. Capera, Georgé J.P., M.P. Gleizes, and P. Gize. The AMAS theory for complex problem solving based on self-organizing cooperative agents. In *Proceedings of the 1st International Workshop on Theory and Practice of Open Computational Systems (TAPOCS'03)*, pages 383–388, 2003.

[43] Matteo Casadei and Mirko Viroli. Applying self-organizing coordination to emergent tuple organization in distributed networks. In Sven Brueckner, Paul Roberson, and Umesh Bellur, editors, *2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'08)*, pages 213–222, Venice, Italy, October 2008. IEEE Computer Society.

[44] Walter Cazzola, Andrea Savigni, Andrea Sosio, and Francesco Tisato. A fresh look at programming-in-the-large. In *The Twenty-Second Annual International Computer Software and Application Conference (COMPSAC '98)*, Viena, Austria, agosto 1998.

[45] Roberto Centeno Sánchez. *Mecanismos incentivos para la regulación de sistemas multiagente abiertos basados en organizaciones*. PhD thesis, Universidad Rey Juan Carlos, Julio 2012.

[46] Betty H. C. Cheng, Rogerio de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee. *Software Engineering for Self-Adaptive Systems: A Research Roadmap*, volume 5525 of *LNCS*, pages 1–26. Springer-Verlag Berlin Heidelberg, marzo 2009.

[47] Shang-Wen Cheng. *Rainbow: Cost-Effective Software Architecture-Based Self-Adaptation*. PhD thesis, School of Computer Science - Carnegie Mellon University, Pittsburgh, PA 15213, mayo 2008.

[48] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. Making self-adaptation an engineering reality. In O. et al. Babaoglu, editor, *SELF-STAR 2004*, number 3460 in LNCS, pages 158–173. Springer-Verlag Berlin Heidelberg, 2004.

[49] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. *Web Services Description Language (WSDL) 1.1*. W3C Consortium, March 2001. W3C Note.

[50] Paolo Ciancarini. Coordination models and languages as software integrators. *ACM Computing Surveys*, 28(2):300–302, June 1996.

[51] Hewlett-Packard Development Company. Adaptive Enterprise. http://www.hp.com/hpinfo/newsroom/press/2003/030506a.html, mayo 2003.

[52] Microsoft Corporation. Dynamic Systems Initiative. http://www.microsoft.com/en-us/news/press/2003/mar03/03-18dynamicsystemspr.aspx, marzo 2003.

[53] Cristóbal Costa, Jennifer Pérez, and José Angel Carsí. Hacia la reconfiguración dinámica de arquitecturas software orientadas a aspectos. In J. Araújo, J. Hernández, E. Navarro, and M. Pinto, editors, *Desarrollo de Software Orientado a Aspectos DSOA 2006*, Sitges (Barcelona), España, octubre 2006.

[54] Luciano Coutinho, Jaime S. Sichman, and Olivier Boissier. Modeling organization in MAS: A comparison of models. In *1st Workshop on Software Engineering for Agent-oriented Systems*, pages 1–10, 2005.

[55] Luciano Coutinho, Jaime S. Sichman, and Olivier Boissier. Organizational modeling dimensions in multiagent systems. In Virginia Dignum, Frank Dignum, Bruce Edmonds, and Eric Matson, editors, *Proceedings of Workshop Agent Organizations: Models and Simulations (AOMS@IJCAI'07)*, Hyderabad, India, enero 2007.

[56] Carlos Cuesta. *Arquitectura de software dinámica basada en reflexión*. PhD thesis, Departamento de Informática - Universidad de Valladolid, Valladolid, España, julio 2002.

[57] Carlos Cuesta, J. S. Perez-Sotelo, and S. Ossowski. Self-organising adaptive structures: the shifter experience. *European Research Consortium for Informatics and Mathematics - ERCIM News*, 2011(85):35–36, abril 2011.

[58] Carlos E. Cuesta and M. Pilar Romay. Elements of self-adaptive systems - a decentralized architectural perspective. In Danny Weyns, Sam Malek, Rogério de Lemos, and Jesper Andersson, editors, *Proceedings of the 1st International Conference on Self-Organizing Architectures (SOAR'09)*, volume 6090 of *LNCS*, pages 1–20, Cambridge, UK, septiembre 2010. Springer-Verlag Berlin Heidelberg.

[59] Carlos E. Cuesta, M. Pilar Romay, Pablo de la Fuente, and Manuel Barrio-Solórzano. Architectural aspects of architectural aspects. In Morrison R. and F. Oquendo, editors, *EWSA 2005*, volume 3527 of *LNCS*, pages 247–262. Springer-Verlag Berlin Heidelberg, 2005.

[60] Prithviraj Dasgupta and Matthew Hoeing. Task selection in multi-agent swarms using adaptive bid auctions. In *Proceedings of the 1st IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'07)*, Boston, MA, USA, julio 2007. IEEE Computer Society.

[61] Mehdi Dastani, Joris Hulstijn, Frank Dignum, and John-Jules Ch. Meyer. Issues in multiagent system development. In *Proceedings of the Third International Joint*

Conference on Autonomous Agents and Multiagent Systems - Volume 2, AAMAS '04, pages 922–929, Washington, DC, USA, 2004. IEEE Computer Society.

[62] Institut de Recherche en Informatique de Toulouse. ADELFE - agent-oriented methodology. http://www.irit.fr/ADELFE/, agosto 2003.

[63] Tom De Wolf and Tom Holvoet. Towards autonomic computing: agent-based modelling, dynamical systems analysis, and descentralised control. In *Proceedings of the 1st International Workshop on Autonomic Computing Principles and Architectures*, pages 10–20, Banff, Canada, 2003.

[64] Scott A. Deloach. The MaSE Methodology. *Methodologies and Software Engineering for Agent Systems-The Agent-Oriented Software Engineering Handbook Series: Multiagent Systems, Artificial Societies, and Simulated Organizations*, 11:107–125, 2004.

[65] Scott A. DeLoach. Moving multi-agent systems from research to practice. *International Journal of Agent-oriented Software Engineering*, 3(4):378–382, May 2009.

[66] Scott A. DeLoach and Juan C. García-Ojeda. O-mase: a customisable approach to designing and building complex, adaptive multi-agent systems. *IJAOSE*, 4(3):244–280, 2010.

[67] Scott A. DeLoach, Walamitien H. Oyenan, and Eric T. Matson. A capabilities-based model for adaptive organizations. *Autonomous Agents and Multi-Agent Systems*, 16(1):13–56, 2008.

[68] Giovanni Denaro, Mauro Pezzè, and Davide Tosi. Adaptive integration of third-party web services. In *Proceedings of the 2005 Workshop on Design and evolution of autonomic application software (DEAS'05)*, St. Louis, Missouri, USA, mayo 2005. ACM.

[69] Virginia Dignum. *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, Universiteit Utrecht, 2004.

[70] Virginia Dignum, editor. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. IGI Global, Hershey, PA, USA, marzo 2009.

[71] Virginia Dignum and Frank Dignum. A landscape of agent systems in the real world. Technical report uu-cs-2006-061, Department of Information and Computing Sciences, Utrecht University, Utrecht, The Neterlands, 2006.

[72] Virginia Dignum, Javier Vázquez-Salceda, and Frank Dignum. Omni: Introducing social structure, norms and ontologies into agent organizations. In *Programming Multi-Agent Systems*, volume 3346 of *LNCS*, pages 181–198. Springer Berlin/Heidelberg, 2005.

[73] Mark d'Inverno and Michael Luck. *Understanding agent systems*. Springer series on agent technology. Springer, New York, 2004.

[74] N. Dulay, N. Damianou, E. Lupu, and M. Sloman. *A Policy Language for the Management of Distributed Agents*, pages 84–100. Springer-Verlag, 2001.

[75] Edmund H. Durfee and Victor R. Lesser. Distributed artificial intelligence (vol. 2). In M. Huhns, editor, *Distributed Artificial Intelligence (Vol. 2)*, chapter Negotiating task decomposition and allocation using partial global planning, pages 229–243. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989.

[76] A. E. Eiben. Evolutionary computing and autonomic computing: Shared problems, shared solutions? In O. Babaoglu et al., editor, *SELF-STAR 2004*, volume 3460 of *LNCS*, pages 36–48. Springer-Verlag Berlin Heidelberg, 2004.

[77] Ahmed Elkhodary and Jon Whittle. A survey of approaches to adaptive application security. In *Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'07)*, Washington, DC, USA, mayo 2007. ACM.

[78] Lee D. Erman and Victor R. Lesser. The hearsay-ii speech understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12:213–253, 1980.

[79] S. Esparcia, E. Argente, V. Julian, and V. Botti. GORMAS: A methodological guideline for Organizational-Oriented Open MAS. In *Handbook on Agent-Oriented Design Processes*, pages 173–218. Springer, 2014.

[80] Jeff A. Esteban, Ken Laskey, Francis G. McCabe, and Danny Thornton. *Reference Architecture for Service Oriented Architecture 1.0*. Organization for the Advancement of Structured Information Standards (OASIS), 2008.

[81] Marc Esteva, David de la Cruz, and Carles Sierra. Islander: an electronic institutions editor. In M. Gini, T. Ishida, C. Castelfranchi, and W. L. Johnson, editors, *First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, pages 1045–1052, Bologna, Italy, julio 2002. ACM.

[82] Marc Esteva, Juan-Antonio Rodriguez-Aguilar, Carles Sierra, Pere Garcia, Josep Arcos, and Frank Dignum. On the formal specification of electronic institutions. *Agent mediated electronic commerce*, 1991(12):126–147, 2001.

[83] Marc Esteva, Bruno Rosell, Juan A. Rodriguez-Aguilar, and Josep Ll. Arcos. Ameli: An agent-based middleware for electronic institutions. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, volume 1, pages 236–243. IEEE Computer Society, 2004.

[84] R. Mandiau et al., editor. *La méthode VOYELLES, dans Systèmes Multi-Agents: Des Théories Organisationnelles aux Applications Industrielles.*, Oslo, Norway, 2001. Hèrmes.

[85] J. Ferber and O. Gutknecht. AALAADIN: a meta-model for the analysis and design of organizations in multi-agent systems. In *3rd International Conference on Multi-Agent Systems (ICMAS-98)*, pages 128–135. IEEE Computer Society, 1998.

[86] J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: an organizational view of multi-agent systems. *Agent-Oriented Software Engineering IV*, pages 443–459, 2004.

[87] Jacques Ferber. *Multi-Agent Systems - an Introduction to Distributed Artificial Intelligence*. Addison-Wesley-Longman, 1999.

[88] Innes A. Ferguson. Touring machines: Autonomous agents with attitudes. *IEEE Computer*, 25(5):51–55, 1992.

[89] Alberto Fernández Gil. *Descripción, Descubrimiento y Composición de Servicios en Entornos Multiagente Abiertos. Un Enfoque Organizacional*. PhD thesis, Universidad Rey Juan Carlos, Móstoles, Madrid - España, noviembre 2007.

[90] José Luis Fernández-Marquez, Giovanna Di Marzo Serugendo, Sara Montagna, Mirko Viroli, and Josep Lluis Arcos. Description and composition of bio-inspired design patterns: a complete overview. *Nat Comput*, mayo 2012.

[91] JoseLuis Fernandez-Marquez, Giovanna Di MarzoSerugendo, PaulL Snyder, Giuseppe Valetto, and Franco Zambonelli. *A Pattern-Based Architectural Style for Self-Organizing Software Systems*, pages 149–172. Auerbach Publications, 2015/05/27 2014.

[92] José Luiz Fiadeiro. Designing for software's social complexity. *Computer*, pages 34–39, 2007.

[93] FIPA-Architecture-TC. FIPA Abstract Architecture Specification. Standard SC00001L, FIPA - Foundation for Intelligent Physical Agents, Geneva, Switzerland, diciembre 2002.

[94] FIPA-TC-C. FIPA ACL Message Structure Specification. Standard SC00061G, FIPA - Foundation for Intelligent Physical Agents, Geneva, Switzerland, diciembre 2002.

[95] Michael J. Fischer, Nancy Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[96] Caire G., Coulier W., Garijo F. J., Gómez J., Pavón J., Leal F., Chainho P., Kearney P. E., Stark J., Evans R., and Massonet P. Agent oriented analysis using message/uml. In *2nd International Workshop on Agent-Oriented Software Engineering (AOSE'01)*, pages 119–135, London, UK, 2002. Springer-Verlag.

[97] Alexander R. Galloway. *Protocol: How Control Exists after Decentralization*. The MIT Press, 1st edition, 2004.

[98] Erich Gamma, Richard Helm, Ralp Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Professional, October 1994.

[99] Emilia Garcia, Adriana Giret, and Vicente Botti. Developing Regulated Open Multi-agent Systems. In Sascha Ossowski, Francesca Toni, and George Vouros, editors, *1st International Conference on Agreement Technologies (AT-2012)*, volume 918, pages 12–26, Dubrovnik, Croacia, octubre 2012. CEUR - Workshop Proceedings.

[100] Luca Gardelli, Mirko Viroli, Matteo Casadei, and Andrea Omicini. Designing self-organising environments with agents and artefacts: a simulation-driven approach. *International Journal of Agent-Oriented Software Engineering*, 2(2):171, 2008.

[101] Luca Gardelli, Mirko Viroli, and Andrea Omicini. Design patterns for self-organising systems. In Hans-Dieter Burkhard, Rineke Verbrugge, and László Zsolt Varga, editors, *Multi-Agent Systems and Applications V*, volume 4696 of *Lecture Notes in Computer Science*, pages 123–132. Springer Berlin Heidelberg, September 2007. 5th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'07), Leipzig, Germany, 25–27 September 2007, Proceedings.

[102] Luca Gardelli, Mirko Viroli, and Andrea Omicini. Design patterns for self-organizing multiagent systems. In Tom De Wolf, Fabrice Saffre, and Richard Anthony, editors, *2nd International Workshop on Engineering Emergence in Decentralised Autonomic System (EEDAS) 2007*, pages 62–71, ICAC 2007, Jacksonville, Florida, USA, June 2007. CMS Press, University of Greenwich, London, UK.

[103] Francisco Garijo, Juan Tous, José M. Matias, Stephen Corley, and Marius Tesselaar. Development of a multi-agent system for cooperative work with network negotiation capabilities. In *Proceedings of the 2nd International Workshop on Intelligent Agents for Telecommunication Applications*, IATA'98, pages 204–219, London, UK, 1998. Springer-Verlag.

[104] Francisco J. Garijo. Tecnología de agentes - experiencias y perspectivas para el desarrollo de nuevos servicios y aplicaciones. *Bole.tic*, 24:1–9, 2002.

[105] David Garlan and Bradley Schmerl. The RADAR architecture for personal cognitive assistance. *International Journal of Software Engineering and Knowledge Engineering*, 2007.

[106] David Garlan, Daniel P. Siewiorek, Asim Smailagic, and Peter Steenkiste. Project aura: Toward distraction-free pervasing computing. *Pervasive Computing - IEEE*, pages 22–31, 2002.

[107] Benjamin Gâteau, Olivier Boissier, Djamel Khadraoui, and Eric Dubois. MoiseInst: An organizational model for specifying rights and duties of autonomous agents. In *3rd European Workshop on Multi-Agent Systems - EUMAS 2005*, pages 484–485, Brussels, Belgium, diciembre 2005.

[108] John C. Georgas and Richard N. Taylor. Policy-based self-adaptive architectures: A feasibility study in the robotics domain. In *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS'08)*, pages 105–112, Leipzig, Germany, mayo 2008. ACM.

[109] Ioannis Georgiadis, Jeff Magee, and Jeff Kramer. Self-organizing software architectures for distributed systems. In Garlan et al., editor, *Proceedings of the 1st ACM SIGSOFT Workshop on Self-Healing Systems (WOSS'02)*, pages 18–19, New York, NY, USA, noviembre 2002. ACM Press.

[110] Adriana Giret. Anemona: a multi-agent methodology for holonic manufacturing systems. In *Advanced Manufacturing*, Advanced Manufacturing. Springer, 2008.

[111] F. Giunchiglia, J. Mylopoulos, and A. Perini. The Tropos software development methodology: processes, models and diagrams. In *Workshop on Agent Oriented Software Engineering (AOSE-2002)*, 2002.

[112] Marie-Pierre Gleizes, Thierry Millan, and Gauthier Picard. Adelfe: Using spem notation to unify agent engineering processes and methodology. Technical Report IRIT/2003-10-R, Institut de Recherche en Informatique de Toulouse, Toulouse, Francia, junio 2003.

[113] Marie-Pierre Gleizes, Camps V., and Glize P. A theory of emergent computation based on cooperative self-organization for adaptive artificial systems. In *4th European Congress of Systems Science*, Valencia, España, 1999.

[114] Hassan Gomaa and Mohamed Hussein. Model-based software design and adaptation. In *Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'07)*, page 7, Washington, DC, USA, mayo 2007. IEEE Computer Society.

[115] S. Hassas, G.D. Marzo-Serugendo, Anthony Karageorgos, and C. Castelfranchi. Self-organising mechanisms from social and business/economics approaches. *Informatica*, 30(1):63–71, 2006.

[116] B. Henderson-Sellers and P. Giorgini. *Agent-Oriented Methodologies*. ITPro collection. Idea Group Pub., 2005.

[117] Ramón Hermoso. *El concepto de confianza en sistemas organizativos. Un enfoque de coordinación para agentes software*. PhD thesis, Universidad Rey Juan Carlos, 2011.

[118] Klaus Herrmann. Self-organizing replica placement - a case study on emergence. In *Proceedings of the 1st IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'07)*, Boston, MA, USA, julio 2007. IEEE Computer Society.

[119] J. R. Hilera and V. J. Martínez. *Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones.* Alfaomega, 2000.

[120] Bryan Horling and Victor Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(4):281–316, mayo 2005.

[121] Nick Howden, Ralph Ronnquist, Andrew Hodgson, and Andrew Lucas. Intelligent agents - summary of an agent infrastructure. In *In 5th International Conference on Autonomous Agents*, 2001.

[122] Enda Howley, Peter Vrancx, and Matt Knudson, editors. *Proceedings of the Adaptive and Learning Agents Workshop (ALA'12) @ AAMAS 2012*, Valencia, España, junio 2012.

[123] Jomi F. Hübner, Jaime S. Sichman, and Olivier Boissier. S-Moise+: A middleware for developing organised multi-agent systems. In *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems - COIN2006*, volume 3913 of *LNCS (LNAI)*. Springer, Heidelberg, 2006.

[124] Jomi F. Hübner, Jaime S. Sichman, and Olivier Boissier. Developing organised multi-agent systems using the MOISE+ model: Programming issues at the system and agent levels. *International Journal of Agent-oriented Software Engineering*, 2007.

[125] IBM. An architectural blueprint for autonomic computing. Technical report, International Business Machines Corporation, enero 2006.

[126] IBM. Autonomic computing. http://www-03.ibm.com/systems/z/os/zos/features/sysmgmt/autonomic/index.html, mayo 2013.

[127] Carlos A. Iglesias, Mercedes Garijo, and José Centeno-González. A survey of agent-oriented methodologies. In *Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages*, ATAL '98, pages 317–330, London, UK, UK, 1999. Springer-Verlag.

[128] Carlos A. Iglesias, Mercedes Garijo, José Centeno-González, and Juan R. Velasco. Analysis and design of multiagent systems using mas-common kads. In *Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages*, ATAL '97, pages 313–327, London, UK, UK, 1998. Springer-Verlag.

[129] Pavón J. and Gómez J. Agent oriented software engineering with ingenias. In *Multi-Agent Systems and Applications III*, volume 2691 of *LNCS*, pages 394–403. Springer-Verlag, 2003.

[130] Nicholas R. Jennings, Katia P. Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[131] Nick Jennings, P. Faratin, A. Lomuscio, S. Parsons, Carles Sierra, and Michael Wooldridge. Automated negotiation: prospects, methods and challenges. *International Journal of Group Decision and Negotiation*, 10(2):199–215, 2001.

[132] D. Jordan, J. Evdemon, A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guizar, N. Kartha, C. Kevin Liu, R. Khalaf, D. Koening, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluiri, and A. Yiu. Web Services Business Process Execution Language Version 2.0. Technical report, Organization for the Advancement of Structured Information Standards (OASIS), 2007.

[133] V. Julian, M. Rebollo, E. Argente, V. Botti, C. Carrascosa, and A. Giret. Using THOMAS for Service Oriented Open MAS. In *Post-Proceedings of SOCASE 2009*, pages 56–70. Springer, 2009.

[134] Leslie Pack Kaelbling. A situated-automata approach to the design of embedded agents. *SIGART Bull.*, 2(4):85–88, July 1991.

[135] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, enero 2003.

[136] Dongsun Kim and Sooyong Park. Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. In *Proceedings of the 2009 International Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'09)*, Vancouver, Canada, mayo 2009.

[137] David Kinny and Michael P. Georgeff. Commitment and effectiveness of situated agents. In *IJCAI*, pages 82–88, 1991.

[138] David Kinny and Michael P. Georgeff. Modelling and design of multi-agent systems. In *ATAL*, pages 1–20, 1997.

[139] Manuel Kolp, Paolo Giorgini, and John Mylopoulos. Multi-agent architectures as organizational structures. *Autonomous Agents and Multi-Agent Systems*, 13(1):3–25, julio 2006.

[140] Fabio Kon, Fabio Costa, Gordon Blair, and Roy H. Campbell. The case for reflective middleware. *Communications of the ACM*, 45(6):33–38, junio 2002.

[141] Jeff Kramer and Jeff Magee. Self-managed systems: an architectural challenge. In *2007 Future of Software Engineering*, FOSE '07, pages 259–268, Washington, DC, USA, 2007. IEEE Computer Society.

[142] P. Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, 2000.

[143] Robert Laddaga. Active software. In Paul Robertson, Howie Shrobe, and Robert Laddaga, editors, *Self-Adaptive Software*, volume 1936 of *Lecture Notes in Computer Science*, pages 11–26. Springer Berlin Heidelberg, 2001.

[144] Anthony LaMarca, Waylon Brunette, David Koizumi, Matthew Lease, Stefan B. Sigurdsson, Kevin Sikorski, Dieter Fox, and Gaetano Borriello. PlantCare: An investigation in practical ubiquitous systems. In Gaetano Borriello and Lars Erik Holmquist, editors, *Proceedings of the 4th International Conference of Ubiquitous Computing (UbiComp 2002)*, volume 2498 of *LNCS*, pages 316–332, Göteborg, Sweden, septiembre-octubre 2002. Intel Research Laboratory at Seattle, Springer Berlin Heidelberg.

[145] J. Lind. Patterns in agent-oriented software engineering. In *Agent-Oriented Software Engineering Workshop - AAMAS'02*, Bologna, Italy, 2002.

[146] Jürgen Lind. *Iterative software engineering for multiagent systems: the MASSIVE method*. Springer-Verlag, Berlin, Heidelberg, 2001.

[147] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[148] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F. Brown, and Rebekah Metz. *Reference Model for Service Oriented Architecture 1.0*. Organization for the Advancement of Structured Information Standards (OASIS), October 2006. Official OASIS Standard (Normative).

[149] Pattie Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, July 1994.

[150] Pattie Maes. Artificial life meets entertainment: Lifelike autonomous agents. *Communications of the ACM*, 38(11):108–114, 1995.

[151] Jeff Magee and Jeff Kramer. Dynamic structure in software architectures. *ACM Software Engineering Notes*, 21(6):3–14, noviembre 1996.

[152] Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Comput. Surv.*, 26(1):87–119, March 1994.

[153] X. Mao and E. Yu. Organizational and social concepts in agent oriented software engineering. In *AOSE IV*, volume 3382 of *LNAI*, pages 184–202. Springer-Verlag, 2005.

[154] P. Marrow, E. Bonsma, F. Wang, and C. Hoile. DIET - A Scalable, Robust and Adaptable Multi-Agent Platform for Information Management. *BT Technology Journal*, 21(4):130–137, 2003.

[155] F. Von Martial. *Coordinating Plans of Autonomous Agents*. LNAI 610. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992.

[156] Ana Mas. *Agentes Software y Sistemas Multi-Agente: Conceptos, Arquitecturas y Aplicaciones*. Pearson Prentice-Hall, 2005.

[157] Philip K. McKinley, Seyed M. Sadjadi, and Betty H. C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, 2004.

[158] M. Mikalsen, N. Paspallis, J. Floch, E. Stav, George A. Papadopoulos, and A. Chimaris. Distributed context management in a mobility and adaptation enabling middleware (madam). In *Proceedings of the 2006 ACM Symposium on Applied Computin*, pages 733–734. ACM, 2006.

[159] Ambra Molesini, Andrea Omicini, Enrico Denti, and Alessandro Ricci. SODA: A roadmap to artefacts. In *Engineering Societes in the Agents World VI*, volume 3693 of *LNAI*, pages 46–92, 2006.

[160] A. Mukhija and M. Glinz. Runtime adaptation of applications through dynamic recomposition of components. In *Proceedings of the International Conference on Architecture of Computing Systems*, pages 124–138, 2005.

[161] Hausi A. Müller, Liam O'Brien, Mark Klein, and Bill Wood. Autonomic Computing - Software Architecture Technology. Technical Note CMU/SEI-2006-TN-006, Carnegie Mellon Software Engineering Institute, Pittsburgh, Pennsylvania, USA, abril 2006.

[162] Jörg P. Müller. *The Design of Intelligent Agents - A Layered Approach*, volume 1177 of *Lecture Notes in Computer Science*. Springer, 1996.

[163] Karen L. Myers and Neil Yorke-Smith. Proactivity in an intentionally helpful personal assistive agent. In *AAAI Spring Symposium: Intentions in Intelligent Systems*, pages 34–37, 2007.

[164] National Institutes of Health - U. S. Department of Health & Human Services. Stem Cell Information. http://stemcells.nih.gov/info/Pages/Default.aspx, Agosto 2015.

[165] Ayodele Oluyomi. *Patterns and Protocols for Agent-Oriented Software Development*. PhD thesis, Department of Computer Science and Software Engineering - Facult of Engineering - The University of Melbourne, Melbourne, Australia, noviembre 2006.

[166] Ayodele Oluyomi, Shanika Karunasekera, and Leon Sterling. Description templates for agent-oriented patterns. *Journal of Systems and Software*, 81(1):20–36, January 2008.

[167] Flavio Oquendo. PI-ADL: An architecture description language based on the higher-order typed pi-calculus for specifying dynamic and mobile software architectures. *SIGSOFT Softw. Eng. Notes*, 29(3):1–14, May 2004.

[168] Peyman Oreizy, Michael M. Gorlik, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, mayo-junio 1999.

[169] OSGi Alliance. OSGi service platform service compendium. Technical report, OSGi Alliance, agosto 2009.

[170] S. Ossowski, V. Julian, J. Bajo, H. Billhardt, V. Botti, and J. M. Corchado. Open Issues in Open MAS: An Abstract architecture proposal. In *CAEPIA 2007*, volume II, pages 151–160. AEPIA, 2007.

[171] Sascha Ossowski. *Co-ordination in artificial agent societies: social structures and its implications for autonomous problem-solving agents*. Springer-Verlag, Berlin, Heidelberg, 1999.

[172] Sascha Ossowski. Coordination in multi-agent systems: Towards a technology of agreement. In *Proceedings of the 6th German conference on Multiagent System Technologies*, MATES '08, pages 2–12, Berlin, Heidelberg, 2008. Springer-Verlag.

[173] Sascha Ossowski, editor. *Agreement Technologies*, volume 8 of *Law, Governance ant Technology*. Springer, 2013.

[174] Lin Padgham and Michael Winikoff. Prometheus: A methodology for developing intelligent agents. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *AOSE*, volume 2585 of *Lecture Notes in Computer Science*, pages 174–185. Springer, 2002.

[175] George A. Papadopoulos and Farhad Arbab. Coordination models and languages. In *Andvances in Computers*, volume 46, pages 329–400. Academic Press, 1998.

[176] A. Paschke, C. Kiss, and S. Al-Hunaty. Npl: Negotiation pattern language - a design pattern language for decentralized (agent) coordination and negotiation protocols. In *E-Negotiation - An Introduction, ISBN 81-314-0448-X*. ICFAI University Press, 2006.

[177] Adrian Paschke, Christine Kiss, and Samer Al-Hunaty. A pattern language for decentralized coordination and negotiation protocols. In *2005 IEEE International Conference on e-Technology, e-Commerce, and e-Services (EEE 2005), 29 March - 1 April 2005, Hong Kong, China*, pages 404–407, 2005.

[178] Tharindu Patikirikorala, Alan Colman, Jun Han, and Liuping Wang. Survey on the design of self-adaptive software systems using control engineering approaches. In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'12)*, Zürich, Switzerland, junio 2012.

[179] Luis Peña, Sascha Ossowski, José-María Peña, and Simón M. Lucas. Learning and evolving combat game controllers. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 195–202, 2012.

[180] J. Santiago Perez-Sotelo, Carlos E. Cuesta, and Sascha Ossowski. Towards Adaptive Service Ecosystems with Agreement Technologies. In *Proceedings of I Workshop on Adaptation in serVice EcosYstems and ArchiTectures (AVYTAT2010) - OTM Federated Conferences*, volume 6428 of *LNCS*, pages 77–87, Crete, Greece, 2010. Springer.

[181] J. Santiago Perez-Sotelo, Carlos E. Cuesta, and Sascha Ossowski. Adaptation patterns in multi-agent architectures: the gathering pattern. In *Proceedings of II Workshop on Variability, Adaptation and Dynamism in software systEms y seRvices (VADER2011) - OTM Federated Conferences*, volume 7046 of *LNCS*, pages 657–661, Crete, Greece, 2011. Springer.

[182] J. Santiago Perez-Sotelo, Carlos E. Cuesta, and Sascha Ossowski. The role of agreement technologies in the definition of adaptive software architectures. *SADIO Electronic Journal of Informatics and Operations Research*, 10(1):53–67, 2011.

[183] Gauthier Picard, Jomi Fred Hübner, Olivier Boissier, and Marie-Pierre Gleizes. Reorganisation and Self-organisation in Multi-Agent Systems. In *International Workshop on Organizational Modeling (OrgMod'09)*, pages pages 66–80, Paris, France, 2009. http://www.emse.fr/p̃icard/publications/picard09orgmod.pdf.

[184] Jeremy Pitt, Daniel Ramirez-Cano, Moez Draief, and Alexander Artikis. Interleaving multi-agent systems and social networks for organized adaptation. *Comput Math Organ Theory*, 17:344–378, mayo 2011.

[185] Proceedings of the AAAI Workshop on Coordination, Organizations, Institutions and Norms (COIN @ AAAI08). *Categorizing Social Norms in a Simulated Resource Gathering Society*, Chicago, Illinois, USA, julio 2008.

[186] Mikhail Prokopenko, Fabio Boschetti, and Alex J. Ryan. An information-theoretic primer on complexity, self-organization, and emergence. *Complexity*, 15(1):11–28, 2009.

[187] RAE. Real Academia Española. http://www.rae.es, septiembre 2015.

[188] Sarvapali D. Ramchurn, Carles Sierra, Lluis Godo, and Nicholas R. Jennings. Negotiating using rewards. *Artificial Intelligence*, 171(10-15):805–837, 2007.

[189] Andres J. Ramirez. Design patterns for developing dynamically adaptive systems. Master's thesis, Michigan State University, 2008.

[190] Andres J. Ramirez and Betty H. C. Cheng. Design patterns for developing dynamically adaptive systems. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'10)*, SEAMS '10, pages 49–58, New York, NY, USA, 2010. ACM.

[191] Andres J. Ramirez and Betty H.C. Cheng. Applying adaptation design patterns. In *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC '09, pages 69–70, New York, NY, USA, 2009. ACM.

[192] Alois Reitbauer, Alessandro Battino, BartSaint Germain, Anthony Karageorgos, Nikolay Mehandjiev, and Paul Valckenaers. The mabe middleware. In Luis M. Camarinha-Matos, editor, *Emerging Solutions for Future Manufacturing Systems*, volume 159 of *IFIP International Federation for Information Processing*, pages 53–60. Springer US, 2005.

[193] Hitachi American Ltd. Research and Development. Harmonious Computing Project (HARC). http://www.hitachi-america.us/products/business/rd/about_us/, 2004.

[194] Alessandro Ricci, Andrea Omicini, and Enrico Denti. Activity Theory as a Framework for MAS Coordination. *Engineering Societies in the Agents World III: Third International Workshop, ESAW 2002, Madrid, Spain, September 16-17, 2002. Revised Papers*, pages 295–394, 2003.

[195] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Environment-based coordination through coordination artifacts. In Danny Weyns, H. Dyke Parunak, and Fabien Michel, editors, *Environments for Multi-Agent Systems*, volume 3374 of *Lecture Notes in Computer Science*, pages 190–214. Springer Berlin Heidelberg, 2005.

[196] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Programming MAS with artifacts. In Rafael P. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems*, volume 3862 of *Lecture Notes in Computer Science*, pages 206–221. Springer Berlin Heidelberg, March 2006. 3rd International Workshop (PROMAS 2005), AAMAS 2005, Utrecht, The Netherlands, 26 July 2005. Revised and Invited Papers.

[197] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. CArtAgO: A framework for prototyping artifact-based environments in MAS. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Environments for MultiAgent Systems III*, volume 4389 of *Lecture Notes in Computer Science*, chapter 4, pages 67–86. Springer Berlin Heidelberg, May 2007. 3rd International Workshop (E4MAS 2006), Hakodate, Japan, 8 May 2006. Selected Revised and Invited Papers.

[198] S. Rodríguez, V. Julián, J. Bajo, C. Carrascosa, V. Botti, and J. M. Corchado. Agent-based virtual organization architecture. *Engineering Applications of Artificial Intelligence*, 24(5), 2011.

[199] Sara Rodríguez, Yanira de Paz, Javier Bajo, and Juan M. Corchado. Social-based planning model for multiagent systems. *Expert Systems with Applications*, 38(10):13005–13023, septiembre 2011.

[200] Sara Rodríguez, Belén Pérez-Lancho, Juan F. De Paz, Javier Bajo, and Juan M. Corchado. Ovamah: Multiagent-based adaptive virtual organizations. In *12th International Conference on Information Fusion, 2009. FUSION '09*, pages 990–997, Seattle, WA, USA, julio 2009.

[201] Sara Rodríguez González, Fernando de la Prieta Pintado, Elena García Peña, Carolina Zato Domínguez, Juan M. Corchado Rodríguez, and Javier Bajo Pérez. Multiagent systems and self-organizative virtual organizations, a step ahead in adaptive mas. In *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, pages 36–41. IEEE Computer Society, noviembre 2011.

[202] Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter - Designing Conventions for Automated Negotiation among Computers*. MIT Press, 1994.

[203] William Ross, Alexis Morris, and Mihaela Ulieru. NEXUS: A synergistic human-service ecosystems approach. In *Self-Adaptive and Self-Organizing Systems Workshops (SASOW'12), 2012 IEEE Sixth International Conference on*, pages 175–180. IEEE Computer Society, 2012.

[204] Norman Salazar, Juan A. Rodríguez-Aguilar, and Josep Lluis Arcos. An infection-based mechanism for self-adaptation in multi-agent complex networks. In *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'08)*, SASO '08, pages 161–170, Washington, DC, USA, 2008. IEEE Computer Society.

[205] Norman Salazar, JuanA. Rodriguez-Aguilar, and JosepLl. Arcos. An infection-based mechanism in large convention spaces. In Julian Padget, Alexander Artikis, Wamberto Vasconcelos, Kostas Stathis, VivianeTorres Silva, Eric Matson, and Axel Polleres, editors, *Coordination, Organizations, Institutions and Norms in Agent Systems V*, volume 6069 of *LNCS*, pages 273–288. Springer Berlin Heidelberg, 2010.

[206] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2):1–42, May 2009.

[207] Davide Sangiorgi and David Walker. *PICalculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.

[208] Guus T. Schreiber and Hans Akkermans. *Knowledge engineering and management: the CommonKADS methodology*. MIT Press, Cambridge, MA, USA, 2000.

[209] Giovanna Di Marzo Serugendo, Marie Pierre Gleizes, and Anthony Karageorgos. Self-organisation and emergence in MAS: an overview. *Informatica (Slovenia)*, 30(1):45–54, 2006.

[210] Carles Sierra, Vicent Botti, and Sascha Ossowski. Agreement computing. *KI - Künstliche Intelligenz*, 25:57–61, 2011. 10.1007/s13218-010-0070-y.

[211] Carles Sierra and John K. Debenham. Information-based agency. In *Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI'07)*, pages 1513–1518. AAAI Press, 2007.

[212] Carles Sierra, Juan A. Rodríguez-Aguilar, Pablo Noriega, Marc Esteva, and José L. Arcos. Engineering multi-agent systems as electronic institutions. *Novatica*, 170, julio-agosto 2004.

[213] Carles Sierra, Jordi Sabater, Jaume Agusti, and Pere Garía. The SADDE Methodology. In F. Bergenti, M.P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering*

Handbook, Multiagent Systems, Artificial Societies, and Simulated Organizations, pages 195–216. Kluwer Academic Publishers, 2004.

[214] David Canfield Smith, Allen Cypher, and Jim Spohrer. Kidsim: programming agents without a programming language. *Communications of the ACM*, 37(7):54–67, July 1994.

[215] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Comput.*, 29(12):1104–1113, December 1980.

[216] João Pedro Sousa and David Garlan. Aura: An architectural framework for user mobility in ubiquitous computing environments. In *Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture - WICSA 2002*, pages 29–43. Kluwer Academic Publishers, 2002.

[217] SUMMA112. Servicio de Urgencia Médica de la Comunidad de Madrid. www.madrid.org/cs/Satellite?pagename=SUMMA112/Page/S112_home, octubre 2015.

[218] Katia P. Sycara. Multiagent systems. *AI Magazine*, 19(2):79–92, 1998.

[219] Juan T., Pearce A., and Sterling L. ROADMAP: Extending the Gaia Methodology for complex open systems. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, pages 3–10, New York, NY, USA, 2002. ACM Press.

[220] Amund Tveit. A survey of agent-oriented software engineering. In *First NTNU Computer Science Graduate Student Conference*, Trondheim, Norway, 2001. Norwegian University of Science and Technology (NTNU).

[221] H. Van Dyke Parunak and Sven A. Brueckner. Engineering swarming systems. In Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, volume 11 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 341–376. Springer US, 2004.

[222] H. Van Dyke Parunak and Sven A. Brueckner. Software engineering for self-organizing systems. *Knowledge Eng. Review*, 30(4):419–434, 2015.

[223] Peter Van Roy, Seif Haridi, Alexander Reinefeld, Jean-Bernard Stefani, Roland Yap, and Thierry Coupaye. Self-management for large-scale distributed systems: an overview of the SELFMAN project. In *Formal Methods for Components and Objects*, volume 5382 of *LNCS*, pages 153–178. Springer-Verlag, 2008.

[224] Mirko Viroli, Matteo Casadei, and Andrea Omicini. A framework for modelling and implementing self-organising coordination. In S.Y. Shin, S. Ossowski, R. Menezes, and M. Viroli, editors, *24th Annual ACM Symposium on Applied Computing (SAC'09)*, volume III, pages 1353–1360, Honolulu, Hawaii, USA, 2009. ACM Press.

[225] Gerhard Weiß, editor. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, Cambridge, MA, USA, 1999.

[226] Danny Weyns. A pattern language for multi-agent systems. In *Proc. WICSA/ECSA 2009*, pages 191–200. IEEE Computer Society, 2009.

[227] Danny Weyns and Michael Georgeff. Self-adaptation using multiagent systems. *IEEE Software*, 27(1):86–91, enero/febrero 2010.

[228] Danny Weyns, Robrecht Haesevoets, and Alexander Helleboogh. The macodo organization model for context-driven dynamic agent organzations. *ACM Transaction on Autonomous and Adaptive Systems*, V(N), noviembre 2010.

[229] Danny Weyns, Robrecht Haesevoets, Alexander Helleboogh, Tom Holvoet, and Wouter Joosen. The MACODO middleware for context-driven dynamic agent organizations. *TAAS*, 5(1), 2010.

[230] Danny Weyns, Alexander Helleboogh, and Tom Holvoet. How to get multi-agent systems accepted in industry? *International Journal of Agent-oriented Software Engineering*, 3(4):383–390, May 2009.

[231] Danny Weyns and Tom Holvoet. An architectural strategy for self-adapting systems. In *Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '07, pages 3–, Washington, DC, USA, 2007. IEEE Computer Society.

[232] FIPA DPDF WG. FIPA Design Process Documentation Template. Standard SC00097B, FIPA - Foundation for Intelligent Physical Agents, Geneva, Switzerland, enero 2012.

[233] L. K. Wickramasinghe and Alahakoon L. D. Adaptive agent architecture inspired by human behaviour. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04)*. IEEE Computer Society, 2004.

[234] Jeannette M. Wing. FAQ on π-Calculus. http://www.cs.cmu.edu/~wing/publications/Wing02a.pdf, diciembre 2002. Último acceso: 17-10-2015.

[235] Michael Wooldridge. Intelligent agents. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 27–77. MIT Press, Cambridge, MA, USA, 1999.

[236] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition, 2009.

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

[238] F. Zambonelli, M.-P. Gleizes, M. Mamei, and R. Tolksdorf. Spray computers: frontiers of self-organization for pervasive computing. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2004 (WETICE'04)*, pages 403–408. IEEE Computer Society, 2004.

[239] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12(3):317–370, July 2003.

[240] Ji Zhang and Betty H. C. Cheng. Model-based development of dynamically adaptive software. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 371–380, New York, NY, USA, 2006. ACM.

[241] Walter Zimmer. Relationships between design patterns. In *NO SE*, 2000.