Tesis Doctoral

Un Enfoque en la Protección de Sistemas de Agentes

Presentada por

Antonio Muñoz Gallego

Para optar al grado de
Doctor Ingeniero en Informatica

Dirigida por

Dr. Antonio Maña Gómez

Profesor Titular de Universidad

Departamento de Lenguaje y Ciencias de la Computación
Universidad de Málaga

Quiero dedicar esta tesis a mi madre que tengo plena confianza que lo estará disfrutando allá donde se encuentre.
Soy consciente de que te lo debía mamá.

Agradecimientos

En primer lugar agradezco a mi director de tesis y amigo Antonio todo su apoyo y guía que considero que, sin duda, han sido fundamentales para todos estos años de trabajo.

Agradezco a Marioli ese apoyo tan cómplice e incondicional para todo lo que ha implicado la evolución de todas y cada una de las etapas de esta tesis. A Domy porque siempre ha representado mi mano derecha y evidentemente durante todos estos años no me ha defraudado en ningún momento. A Nacho que me ha servido de brújula para no perder el norte. A Marian que siempre ha sabido llevarnos a todos al plano terrenal y en mi caso especialmente. Por su puesto a mi padre que gracias a él me enorgullezco de ser quien soy y quien, sin ser del todo consciente de ello, me hizo firmar el contrato de compromiso esta tesis. Agradezco a mis abuelas Ascensión y Eulogia, ya que sin su ayuda no habría podido tener la oportunidad de estar escribiendo estas líneas.

Agradezco también al resto de mis familiares que con paciencia han comprendido todo este tiempo empleado. Quiero dar las gracias a mis amigos Salva, Javi, Monte, Onieva, Sergio, Iván, Esther, Nati, Espe, Belén, Arantxa, Dani, Fran, Seba, Carlos, José María, Gustavo y a los que me dejo en el tintero. Asimismo a mis compañeros de laboratorio, en especial a Pablo, Jose, Andrés y Hristo por ese aliento continuo en el camino y los apretones de última hora.

Para finalizar quiero hacer mención a todos los que han contribuido en el desarrollo de este trabajo que de cualquier forma han contribuido a que por fin vea la luz.

Resumen

Los agentes móviles son entidades software que tienen la habilidad de migrar entre nodos dentro de una red, actuando de forma autónoma y en cooperación con otros agentes con el objetivo de realizar una serie de tareas. Actualmente existen diferentes aplicaciones basadas en agentes en numerosos entornos de computación tales como redes peer-to-peer, web crawlers, etc. Los sistemas multiagentes, MAS (Multi-Agent Systems) representan una propuesta arquitectural muy prometedora para la construcción de aplicaciones basadas en Internet y distribuidas.

A pesar de la atención prestada, por parte de la comunidad investigadora, a este campo no ha habido una aceptación evidente, de hecho, únicamente se ha aplicado en un número reducido de escenarios del mundo real. En este trabajo hemos llegado a la conclusión de que los aspectos de seguridad juegan un papel vital para el desarrollo de este paradigma. Un hecho que lo evidencia es que el principal problema a resolver antes de que la tecnología basada en agentes esté lista para ser usada de forma habitual fuera de la comunidad investigadora es la falta de mecanismos de seguridad apropiados. Para conseguir un nivel de seguridad apropiado, en el contexto de los sistemas multiagentes, no es suficiente que la plataforma de agentes ofrezca un conjunto de mecanismos de seguridad estándares tales como son los sandboxing, o mecanismos de cifrado y firma digital, sino que es necesario, diseñar la infraestructura considerando los aspectos de la seguridad desde las primeras etapas del diseño hasta la últimas de la implementación. Además, la seguridad de una plataforma de agentes necesita ser adaptada a las características específicas de estos sistemas.

En esta tesis doctoral partimos de una base de estudio de las tecnologías actuales que pueden usarse para construir sistemas de agentes seguros. Pero debemos advertir que este estudio nos ha servido para concluir que ninguno es suficientemente completo, puesto que los mecanismos existentes hasta el momento se centran o bien en la protección de la agencia o bien en la protección del agente. Esto se debe en gran medida a que la mayoría de las soluciones que se usan para la protección de los agentes no está diseñada para este modelo computacional, sino que suelen ser adaptaciones de otros mecanismos usados para la protección general del software. También queremos resaltar el hecho de que la aplicación de las técnicas actuales para la protección de los sistemas basados en agentes implica ciertos conocimientos en materia de seguridad que no resultan triviales. De hecho hemos comprobado que una de las causas de que no se apliquen técnicas de seguridad en este ámbito es la dificultad que presentan en su aplicación este tipo de prácticas.

En esta tesis presentamos dos soluciones para la protección de sistemas basados en agentes claramente diferenciadas. Aunque, como describimos en el capítulo de conclusiones, también son complementarias. La primera de las técnicas se basa en el uso de hardware criptográfico para la protección de los agentes. Hemos diseñado un protocolo de migración segura específico para un hardware determinado Trusted Platform Module (TPM). No obstante, esta técnica se puede aplicar a cualquier otro tipo de hardware con características similares, como es el caso de las tarjetas inteligentes. La segunda de las técnicas diseñadas hace uso del paradigma conocido como "computación protegida". Este se basa en la partición del código de los agentes y en realizar una repartición entre las

partes de código obtenidas. Como vemos en transcurso de esta tesis hemos desarrollado herramientas de ayuda para la aplicación de esta técnica a sistemas de agentes reales, de forma sencilla y automática. Hemos incluido un capítulo para las conclusiones y los trabajos futuros, ya que consideramos de gran interés continuar con esta línea de investigación, que creemos que tiene un futuro muy prometedor. Hemos añadido un apéndice en el que se incluye una traducción, en inglés, de la introducción de la tesis junto con un resumen, en inglés, del contenido global de la tesis. Y por último hemos añadido un apéndice con la metodología seguida para el desarrollo de esta tesis.

Índice general

Capítulo 1

Introducción

1.1. Acerca de la tesis

Este trabajo de tesis se ha desarrollado en el Grupo de Investigación de la Ingeniera del Software de la Universidad de Málaga (GISUM) del departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga. Las soluciones a las que ha contribuido el desarrollo de esta tesis se enmarcan dentro de los siguientes proyectos de investigación:

- UBISEC: Ubiquitous Networks with a Secure Provision of Services, Access, and Content Delivery (IST-FP6-506926).
- SERENITY: System Engineering for Security and Dependability (IST-PF6-027587).
- Mistico-Mechanics: Proyecto de la Junta de Castilla la Mancha en colaboración con la Universidad de Castilla La Mancha.

1.2. Motivación y objetivos

Los agentes móviles son entidades software con la habilidad de migrar entre nodos dentro de una red, actuando tanto de forma autónoma como en cooperación con otros agentes para realizar una serie de tareas. En la actualidad existen numerosas aplicaciones basadas en tecnología de agentes, utilizadas en diversos entornos de computación, como son las redes peer-to-peer, los web crawlers, etc. Los sistemas multiagentes, MAS (Multi-Agent Systems) representan una propuesta arquitectural muy prometedora para la construcción de aplicaciones para la web, así como para aplicaciones distribuidas. Este tipo de sistemas pueden aportar beneficios muy relevantes, especialmente dentro de escenarios altamente distribuidos. De hecho, los altos niveles de autonomía y auto-organización de los sistemas de agentes proporcionan un excelente apoyo para el desarrollo de sistemas, en los que la fiabilidad es esencial. En esta categoría destacamos los escenarios de computación ubicua y los ambientes inteligentes dado que estos sistemas son la base de numerosas aplicaciones en las que la fiabilidad y la seguridad son esenciales.

A pesar del interés prestado a este campo, por parte de la comunidad investigadora, su aceptación no ha tenido un impacto tan relevante como parecía esperar. Únicamente se ha aplicado en un número bastante reducido de escenarios del mundo real. Pensamos que este hecho se debe a que los aspectos de seguridad juegan un papel crucial en el desarrollo de sistemas multi-agentes y se considerada uno de los principales problemas a resolver antes de que esta tecnología esté totalmente preparada para ser usada de una forma habitual, en especial en entornos empresariales.

No obstante, para conseguir los mecanismos de seguridad apropiados para los sistemas basados en agentes, no basta únicamente con que la plataforma de agentes proporcione un conjunto de mecanismos y técnicas de seguridad estándares como los sandboxing o ciertos mecanismos de cifrado y de firma digital. En nuestro trabajo partimos de la base de que es necesario que durante todas las fases de la vida del software se tengan en consideración los aspectos de seguridad. En efecto, como muchos autores han mostrado la idea de considerar la seguridad como un elemento añadido ha quedado obsoleta. Otro aspecto a resaltar es el hecho de que la seguridad de un sistema basado en agentes necesita ser diseñada conforme a las características específicas de estos sistemas y de las tecnologías que se usan para construirlas. Especialmente relevante es el caso de la movilidad.

Los principales objetivos de este trabajo de tesis son:

- El estudio de los puntos débiles de la seguridad de los sistemas basados en agentes, tanto a nivel de agente como de la plataforma en la que este se ejecuta.
- Un análisis de los diferentes mecanismos que existen hasta el momento para proporcionar seguridad a los sistemas basados en agentes. La evaluación de los mismos tendrá en cuenta tanto aspectos de seguridad como de usabilidad, los cuales resultan vitales para la aceptación práctica de las soluciones.
- El tercer objetivo consiste en el diseño de una solución que proporcione a los desarrolladores de sistemas de agentes los medios necesarios para la construcción de sistemas seguros, así como su implementación, junto con las herramientas necesarias para facilitar, en la medida de lo posible, la tarea al usuario final.
- El último de los objetivos es la validación de nuestra solución. Esto se llevará a cabo mediante el análisis de la utilización de nuestra solución y las herramientas de ayuda asociadas aplicados en sistemas reales basados en agentes.

Hemos de resaltar el hecho de que inicialmente nuestro tercer objetivo era el diseño de una única solución, sin embargo durante el diseño de la misma se nos planteaba el dilema de decantarnos por una solución basada únicamente en elementos software o por el contrario hacer uso de algún tipo de dispositivo de seguridad. Por todo esto decidimos no elegir entre una y otra opción sino que continuamos el estudio de ambas soluciones en paralelo.

1.3. Publicaciones que avalan esta tesis

En esta sección hacemos un repaso de la contribución de esta tesis. Como hemos mencionado en los objetivos iniciales hemos desarrollado dos metodologías completamente

funcionales para la protección de sistemas basados en agentes.

La primera de estas metodologías se basa en el uso de hardware criptográfico. Hemos desarrollado un protocolo de migración seguro que es la base de esta solución, que hace uso de las funcionalidades proporcionadas por este dispositivo. Este protocolo representa un avance respecto al estado del arte en la protección de sistemas de agentes. Hemos validado nuestro protocolo mediante el uso de técnicas de "model checking", en concreto, hemos usado AVISPA, que proporciona un motor de interés para la validación de protocolos. Hemos probado que el protocolo cumple las expectativas para las que fue diseñado a través de diferentes casos de usos. Para lograr un uso fácil de nuestra solución hemos desarrollado una librería de migración segura integrada en la conocida plataforma de agentes JADE, sin necesidad de tener un conocimiento experto en materia de seguridad para utilizarla.

La segunda metodología se basa en el uso de la técnica conocida como "computación protegida". Esta metodología difiere diametralmente de la anterior puesto que se basa únicamente en protección software. El núcleo de esta metodología es la realización de una división inteligente del código y una repartición del mismo entre diferentes agentes. Hemos desarrollado herramientas de apoyo para la aplicación de esta metodología a nivel de usuario, para facilitar la ardua tarea de aplicar técnicas expertas de seguridad para la protección de agentes. Para ello hemos diseñado herramientas para automatizar la división del software conforme a las necesidades de cada caso mediante un "perfil de protección". Hemos analizado un conjunto amplio de posibles requisitos para la división del código y hemos diseñado este conjunto de herramientas para capturar los requisitos analizados.

Además se ha contrastado el valor de nuestro trabajo en la comunidad científica por medio de varias publicaciones que describimos y relacionamos con cada una de las partes desarrolladas en esta tesis. Además de la presentación y discusión de los distintos elementos que forman parte del conjunto de la investigación completa de la tesis en diferentes foros y conferencias de prestigio.

Vamos a hacer un repaso de todas los trabajos de investigación que avalan el valor científico de esta tesis y su relación con algunas de las publicaciones. Para ello hemos estructurado la sección en tres partes claramente diferenciadas. En la primera de ellas, describimos las aportaciones más relevantes centradas en la resolución del problema haciendo uso del elemento hardware de apoyo. En la segunda, mostramos las publicaciones, que hemos considerado, más relevantes en lo que respecta a la solución que toma como base la "computación protegida". Por último, en la tercera parte resaltamos una publicación centrada en aplicaciones futuras de los resultados obtenidos a lo largo de esta tesis.

1.3.1. Publicaciones relacionadas con la solución basada en el uso de TPM

En el capítulo 3 hemos descrito con riguroso nivel de detalle las características principales de esta solución, en cambio, en esta sección vamos a citar las principales publicaciones relacionadas con esta solución.

Hemos considerado importante incluir el trabajo "The Role of Trusted Computing in the Secure Agent Migration", que tiene por autores a Antonio Muñoz, Antonio Maña y Daniel Serrano. Publicada en el "International Journal of Computer Science & Applica-

tions". ISSN 0972-9038. Así como el trabajo titulado, "TPM-based Protection for Mobile Agents", que tiene como autores a Antonio Muñoz y a Antonio Maña Gomez. Aceptado para publicarse en el "Security and Communication Networks". Estas publicaciones describen los conceptos básicos de la protección de los sistemas de agentes que hacen uso de elementos hardware específicos para ello. Estas publicaiones están orientadas en mostrar un mecanismo para reforzar el proceso de migración obteniendo una migración segura. En la primera parte del capítulo 3 se describen los resultados obtenidos en este trabajo. En cambio, el segundo trabajo es una evolución del anteriormente comentado, aunque en este caso se centra en las posibles mejoras aplicables a este método de migración a distintos niveles. En primer lugar se estudian las diferentes alternativas a otros protocolos existentes, como es el caso del protocolo de atestación anónima directa que consideramos de especial interés para ciertos casos. Además se realiza un proceso de validación y se diseña un nuevo protocolo basado en el uso de una clave ligada al estado del sistema, que es la mayor aportación práctica de esta solución basada en hardware. En el capítulo 3 mostramos los resultados obtenidos en esta publicación.

Asimismo hemos introducido en esta sección dos capítulos de libros de bastante importancia. El primero de ellos se titula "Model Cheking Ambient Intelligence with AVISPA", dentro del libro "Ambient Intelligent Perspectives", los autores de este han sido Antonio Muñoz, Antonio Maña y Daniel Serrano. Ha sido publicado en el IO Press y editado por Peter Mikuleck, Tereza Liskov, Pavel Cech y Vladimir Bures en el 2009. ISSN 1875-4163. ISBN 978-1-58603-946-2. En este se describen una serie de mecanismos para modelar soluciones de seguridad específicos para ambientes inteligentes, para su posterior validación mediante unas herramientas de model-checking. Lo particularmente interesante de esta aportación es que proporciona ciertas herramientas para el modelado de soluciones de seguridad particulares de ambientes inteligentes. Al igual que el capítulo titulado "Verification of S&D solutions for workflows, network communications and devices", con Carsten Rudolph, Luca Compagna, Antonio Muñoz, y Jurgen Repp como autores. Este capítulo forma parte del libro "Security and Dependability for Ambient Intelligence", editado por Dr. Georgios Spanoudakis, Antonio Maña y Christos Kokolakis, publicado en Junio de 2009 por el Information Security Series, Springer, ISBN-978-0-387-88775-3. Son aportaciones de especial interés puesto que nos han proporcionado los medios necesarios para poder elaborar una validación formal del protocolo que usamos como base de nuestra solución a lo largo de la investigación. Permitiéndonos la identificación de las debilidades presentes en el mismo.

1.3.2. Publicaciones relacionadas con la solución basada en la computación protegida

De forma paralela a la sección anterior, en esta vamos a mostrar el conjunto de publicaciones relacionadas con la solución basada en la técnica conocida como "computación protegida" que hemos considerado de mayor impacto.

Los dos trabajos más destacados relacionados con esta parte de la tesis son el titulado "Protected Computing vs. Trusted Computing", realizado por Antonio Maña y Antonio Muñoz, presentado en el First International Conference on Communication System Software and Middleware, publicado por el IEEE. Este trabajo representa la semilla de todo

el trabajo de esta tesis, y en particular del capítulo 4. En este artículo se plantea el uso del Protected Computing como una alternativa al uso del Trusted Computing. En la presente tesis doctoral planteamos un paralelismo de dos soluciones para la protección de agentes, precisamente una basada en la primera de las propuestas planteadas en este artículo (Protected Computing) y la otra basada en la segunda (Trusted Computing). En ella se plantean las ventajas e inconvenientes de cada una de las tecnologías propuestas.

Al igual que la publicación previa esta ha sido esencial para el desarrollo de la investigación de toda la tesis y en ella se plantea la organización de la misma en los dos grandes bloques, la solución basada en hardware y la solución basada en software. A continuación y en el mismo año el trabajo con nombre "Mutual Protection for Multiagent Systems", realizado por Antonio Maña y Antonio Muñoz. Proceedings of the Third International Workshop on Safety and Security in Multiagent Systems (SASEMAS '06), como parte del AAMAS. En este artículo presentamos la primera propuesta, a alto nivel, de la aplicación de la computación protegida para la protección de agentes. Las aportaciones que obtuvimos en este congreso nos fueron de gran utilidad, ya que el AAMAS representa el congreso internacional más importante dentro del ámbito de los agentes. Además de disponer de una sesión específica para los aspectos de seguridad, sin lugar a dudas, representó un avance cualitativo para el desarrollo de este trabajo. Por último en este año se publicó "A Secure and Auto-configurable Environment for Mobile Agents in Ubiquitous Computing Scenarios", los autores fueron Javier Lopez, Antonio Maña y Antonio Muñoz, en el Third International Conference Ubiquitous Intelligence and Computing, en las páginas 977-987, LNCS 4159, Springer, Wuhan, China, Septiembre de 2006.

En el año siguiente se publicó el trabajo "Towards Secure Agent Computing for Ubiquitous Computing and Ambient Intelligence", realizado por Antonio Maña, Antonio Muñoz y Daniel Serrano. En la cuarta conferencia internacional de Ubiquitous Intelligence and Computing. Los datos específicos de esta publicación son ISSN 0302-9743. ISBN 978-3-540-73548-9. LNCS, Springer Verlag. Este congreso tuvo lugar en Hong Kong (China) en el 2007. En este artículo planteamos a un nivel bastante abstracto una solución basada en el uso de smartprot, en este trabajo hacíamos la primera distinción entre una versión académica diseñada para un número fijo de agentes dentro de un sistema y otra dinámica, con un mayor grado de complicación.

En el mismo año y en esta misma línea, pero desde otra perspectiva distinta hicimos el trabajo "Trusted Code Execution in JavaCard", Antonio Maña y Antonio Muñoz. Este trabajo se publicó en los proceedings Springer-Verlag, LNCS. Dentro de la conferencia Trustbus en el 2007, que tuvo lugar en Regensburg (Alemania). Hemos de resaltar la relevancia de este congreso, que representa uno de los más importantes internacionalmente dentro del ámbito de la seguridad. En este artículo se propone una mejora a la ejecución del código en Javacard. Esta tecnología está descrita en el capítulo , así como su aplicación como alternativa a los resultados de la aplicación de las técnicas de trusted computing se discuten en el capítulo 3.

Tras una etapa de maduración, en el 2009 se publicaron dos trabajos de gran impacto. El primero de ellos titulado "SecMiLiA: An Approach in the Agent Protection", realizado por Antonio Muñoz, Antonio Maña y Daniel Serrano. Trabajo que fue publicado en el proceedings de la cuarta conferencia internacional de Availability, Reliability and Security ARES 2009, esta conferencia tuvo lugar entre los días 16-19 de marzo en Fukuoka en el

Institute of Technology (FIT) publicada por el IEEE. En este artículo presentamos la primera versión de la librería de migración segura para agentes móviles y es este artículo encontramos una descripción de las clases más relevantes que componen la librería. Este trabajo describe las librerías desarrolladas utilizando la tecnología del *Trusted Computing* y aplicada a un estándar de facto como es JADE. Los resultados se encuentran en la sección C.1 del capítulo 3. En este artículo se defiende el uso de la tecnología de Trusted Computing como pieza clave para la consecución de una solución segura. El segundo de los artículos que destacamos de este año se llama "Agent Protection based on the use of cryptographic hardware", con los autores Antonio Muñoz, Antonio Maña, Rajesh Harjani y Marioli Montenegro. Trabajo que aparece publicado en los proceedings del 33 Annual IEEE International Computer Software and Applications Conference, COMPSAC'09 celebrado en Seattle,Washington,entre los días 20-24 de Julio y que fue publicado por el IEEE. Este trabajo se presentó en una de las conferencias internacionales más relevantes dentro del ámbito de la seguridad con el objetivo de validar las bases criptográficas desarrolladas en el capítulo 3 frente a un grupo de expertos en la materia. Se obtuvieron unas críticas bastante constructivas con la aportación de nuevas ideas que están descritas a lo largo de este trabajo de tesis. Tras las contribuciones del trabajo previamente descrito, decidimos incorporar nuevos detalles y proponer la idea en un escenario de discusión dentro del ámbito de la seguridad.

1.3.3. Publicaciones relacionadas con trabajos futuros

En esta sección únicamente vamos a citar un artículo que hemos destacado del resto en lo que respecta a los trabajos futuros. Este tiene como título "Agent Paradigm for Engineering AmI", bajo la autoría conjunta de Amed Raian, Amir Sameh, Paolo Giorgini, Antonio Maña y Antonio Muñoz. Hemos de resaltar la colaboración, en el desarrollo del mismo, con un grupo de investigación de la Universidad de Trento (Italia), se publicó en la segunda edición de la conferencia Ambient Intelligence Conference 2007, las actas de este congreso se publicaron con el ISBN 978-2-287-78543-6, por Springer-Verlag. En este artículo se realiza una defensa del paradigma del agente móvil en el ámbito de los Ambientes Inteligentes. En concreto se enfatiza la utilidad para la resolución de cierto tipo de problemas. Representa un punto de partida sobre el que continuar los resultados obtenidos en el trabajo de esta tesis.

Capítulo 2

Antecedentes y Trabajos Previos

A lo largo de este capítulo vamos a hacer un repaso sobre los antecedentes y los trabajos previos más influyentes para el desarrollo de la presente tesis. El objetivo principal de este capítulo no es otro que el de situar al lector en el problema que nos planteamos al comienzo de esta tesis. Para ello vamos a describir en profundidad lo que entendemos por agente móvil, puesto que es el elemento sobre el que hemos centrado todo nuestro trabajo, para ello vamos a describir sus características más importantes, los diferentes tipos existentes, la definición de sistemas multiagentes, etc. A continuación mostramos las tecnologías para trabajar con agentes que actualmente tienen un mayor impacto en la comunidad, como es el estándar FIPA y el framework JADE. También vamos a dar un repaso por los trabajos que hasta el momento se han hecho para la protección de los sistemas de agentes. Hemos organizado este tipo de mecanismos de protección en dos grupos bien definidos, el primero de ellos es el que engloba a las soluciones que protegen a los agentes frente a la agencia y el segundo contiene a las soluciones que tratan de proteger a las agencias de los posibles ataques por parte de los agentes. Y por último vamos a hacer una descripción de las tecnologías de base para el desarrollo del trabajo de esta tesis.

Antes de comenzar una descripción detallada de los antecedentes y los trabajos previos en la protección de los agentes móviles y las plataformas de los mismos, debemos puntualizar el hecho de que existen dos concepciones distintas de agentes móviles. Por un lado nos encontramos con la asumida en el área de la *Inteligencia Artificial* que se centra en la capacidad de decisión y de razonamiento, en la autonomía y ciertos aspectos de los agentes, y por otra parte nos encontramos la que está más relacionada con la *Ingeniería del Software*, que muestra a los agentes como un modelo para construir sistemas software. Es en torno a esta última concepción en la que centramos todo nuestro trabajo.

2.1. El paradigma del agente móvil

El paradigma del agente móvil ha sido estudiado por muchos especialistas en diferentes campos. Por esta razón nos encontramos este concepto en un abanico muy amplio de sistemas y aplicaciones aunque no siempre tratado del mismo modo. Algunas de las aplicaciones más características de este paradigma son el comercio electrónico y los sistemas que implican algún tipo de negociación. En general, podemos decir que desde el punto de

vista de la transmisión de datos, el concepto del agente siempre resulta conveniente en los casos en los que se procesan datos remotos, el programa es mucho menor que los mismos, o las características del almacenamiento de esos datos hacen inconveniente su transmisión (por razones de tamaño, seguridad, etc). Evidentemente en aplicaciones tales como las de comercio electrónico en las que se gestiona dinero, la confianza de los usuarios es un prerrequisito para su uso. Este caso nos revela la importancia real de la seguridad en los agentes móviles y el porqué se han invertido tantos esfuerzos para conseguir soluciones de seguridad apropiadas. El gran número de trabajos que se encuentran en la literatura científica relacionada con la seguridad de los agentes móviles evidencia la importancia del tema.

Una de las características más valoradas de los agentes móviles es precisamente su movilidad, que les permite viajar de manera autónoma a través de la red. Sin embargo, esta es precisamente la propiedad que les hace más vulnerables a diferentes tipos de ataques. A continuación vamos a presentar estos ataques, junto con los que puedan realizar los agentes para atacar a las plataformas.

- Accesos no autorizados. Los agentes móviles pueden intentar acceder a servicios y/o recursos de una plataforma sin los permisos adecuados. Para contrarrestar este tipo de ataque, una plataforma de agente móvil ha de disponer de una política de seguridad para especificar las reglas de acceso para varios agentes, así como de un mecanismo para hacer cumplir las políticas.

- Suplantación de identidad. En este tipo de ataques un agente con carácter malicioso asume la identidad de otro agente con idea de obtener acceso a los recursos y servicios de la plataforma, o simplemente para causar algún tipo de daño. Asimismo, una plataforma puede suplantar la identidad de otra plataforma para obtener acceso a los datos del agente móvil. En ambos casos, el objetivo del ataque es que el agente o la plataforma cuya identidad fue mal usada asuma la responsabilidad de los daños [20].

- Denegación de servicio. Una plataforma con carácter malicioso puede causar daño a un agente visitante simplemente ignorando las peticiones de servicios y recursos del mismo, aunque estén disponibles en la plataforma. También puede causar daños forzando la terminación de la ejecución del agente sin notificación alguna, e incluso asignándole tareas de forma continua de manera que este nunca llegue a conseguir su objetivo. Asimismo, un agente malicioso puede causar daños consumiendo más recursos de los necesarios de la plataforma, como pueden ser el espacio de disco o el consumo de CPU. También podría alterar o eliminar información, pudiendo causar daños serios a la plataforma y perpetrando un ataque de denegación de servicio contra otros agentes visitantes [20].

- Escucha no autorizada. En este caso una plataforma con carácter malicioso puede monitorizar el comportamiento o las comunicaciones de un agente móvil con idea de extraer información sensible del mismo. Esto se usa normalmente con el código y los datos del agente cifrados. La monitorización puede revelar la identidad de las entidades con que se comunica el agente móvil, y los tipos de servicios requeridos por el agente móvil [20].

- Alteración. En el ataque por alteración, una plataforma maliciosa intenta modificar la información del agente móvil mediante la insercción, borrado y/o alteración del código, datos e incluso estado de ejecución del propio agente. La modificación del código de ejecución y el estado del agente móvil en ejecución puede dar como resultado que el agente lleve a cabo acciones perjudiciales a su dueño, o a otras plataformas, incluyendo la de origen del agente [20].

A continuación vamos a explorar los requisitos mínimos de seguridad deseables para cualquier sistema basado en agentes.

- Confidencialidad. Es importante asegurar que la información que lleva un agente móvil sea almacenada en una plataforma únicamente accesible para entidades autorizadas, es decir, la información que contiene un agente no debe ser accesible en ningún momento por ninguna parte que no esté autorizada a disponer de ella.

- Integridad. Es indispensable proteger el código, estado y datos del agente móvil ante cualquier tipo de modificación no autorizada. Esto puede conseguirse bien mediante prevencción o mediante la detección de modificaciones no autorizadas.

- Disponibilidad. Las plataformas normalmente están expuestas a una gran demanda de datos y servicio, aunque han de presentar un nivel mínimo de disponibilidad en el sistema para un funcionamiento eficiente del mismo. Además de esto, una plataforma debería ser capaz de afrontar cierto nivel de tolerancia a fallos y de recuperación de fallos inesperados tanto por parte del software como del hardware.

- Responsabilidad. Las plataformas necesitan mantener ficheros de historia de lo que se ha realizado, para mantener constancia de todas las acciones de los agentes móviles, para mantener el historial de las acciones de los mismos. Estos ficheros de historiales son necesarios cuando la plataforma necesite recuperarse de una violación de la seguridad o de un fallo en el sistema y para mantener la responsabilidad de las acciones realizadas por cada una de las partes.

- Anonimato. Como se mencionó anteriormente, las plataformas necesitan mantener constancia del recorrido y de las acciones de los agentes móviles para propósitos estadísticos. Sin embargo, las plataformas también han de realizar un balanceo entre sus necesidades para las auditorías de estos historiales y las necesidades de los agentes móviles de mantener sus acciones en privado.

2.2. Agentes software

Una vez que hemos hecho un repaso a las propiedades de seguridad más importantes que sería deseable tener presentes en los sistemas que usan agentes móviles. Vamos a hacer una descripción exhaustiva de las características y tipologías de los distintos tipos de agentes software, definiremos el concepto de sistema multiagente y describiremos las plataformas y estándares más relevantes relacionados con esta tecnología.

Un agente software es una abstracción que representa a una entidad software capaz de actuar con cierto grado de autonomía. Haciendo una breve reseña histórica, el concepto de

agente surgió en el campo de la robótica en el que los agentes físicos son entidades independientes que perciben el mundo que les rodea a través de sus sensores y que interactúan con él mediante el uso de actuadores.

El término "agente" describe una abstracción de un elemento software, y comparte algunas similitudes con los objetos en la programación orientada a objetos, la principal diferencia se encuentra en los aspectos de autonomía y movilidad. El concepto de agente nos proporciona una forma conveniente y poderosa de describir una entidad de software compleja capaz de actuar con cierto grado de autonomía para cumplir tareas en representación de personas u otras entidades.

Un agente no es un programa tradicional, ya que es capaz de reaccionar al entorno y tiene autonomía y persistencia. En la programación tradicional se han desarrollado programas con una rutina preestablecida, sin capacidad de decisión puesto que se basaban en ciertos elementos esenciales, como eran los bucles y los saltos para desarrollar el flujo del programa. Sin embargo, con la tecnología de agentes se abren las puertas a un nuevo modelo capaz de reaccionar frente a cambios en el entorno y con el grado suficiente de autonomía para tomar decisiones conforme al comportamiento propio y a estos cambios.

No debemos concebir a un agente como un objeto, de hecho, en la concepción del mundo como objetos existe el concepto de encapsulación. En cambio, el concepto de agente representa una entidad que es más autónoma, que como mencionábamos anteriormente tiene capacidad de decisión. Obviamente esta capacidad de decisión junto con la capacidad de reaccionar al entorno hace de los agentes una tecnología mucho más flexible que los objetos, hecho que se evidencia con la capacidad de reactividad propia de los agentes sumada a su capacidad de actuar socialmente con otros agentes.

Por último hemos de afirmar el hecho de que un agente no es un sistema experto, porque los sistemas expertos no se adaptan a su entorno, es decir, los sistemas expertos adquieren una gran cantidad de conocimiento experto mediante una adquisición de la información similar a un método de aprendizaje, en cambio, no están diseñados para tener comportamientos reactivos ni proactivos, así como tampoco tienen habilidad social, que es lo que realmente los diferencia de los agentes.

2.2.1. Características

El paradigma de la programación orientada a agentes móviles aporta una serie de ventajas e inconvenientes debido a la naturaleza del mismo. Esta naturaleza viene dada por una serie de características. A saber:

- Autonomía, esta característica permite a los agentes actuar sin ningún tipo de intervención humana directa y tener control sobre sus propios actos.
- Sociabilidad, los agentes tienen capacidad para interaccionar con otros agentes humanos o no, a través de algún tipo de lenguaje de comunicación.
- Capacidad de reacción, los agentes pueden actuar sin la intervención de un ser humano o de otros agentes, y tienen algún tipo de control sobre sus acciones y su estado interno.

- Iniciativa, un agente inteligente debe tener la capacidad para ejecutar tareas para conseguir su objetivo, separada de la propia del usuario. Los problemas que motivaron la aparición del concepto de agente son aquellos en los que el agente puede llegar a saber más que el usuario sobre una materia y/o sobre las estrategias a utilizar para resolver el problema.
- Movilidad, en el caso de los agentes móviles la movilidad es la habilidad que tiene un agente para trasladarse a través de una red telemática o mundial (como Internet). El mecanismo de transporte del agente utilizado varía desde TCP/IP a correo electrónico.
- Cooperación, permitida entre entidades de agentes. La complejidad de la cooperación puede variar desde un estilo de interacción cliente/servidor a negociaciones y cooperación basada en métodos de inteligencia artificial, tales como redes de contratos y protocolos. Esta cooperación puede necesitar del intercambio de información y representaciones de prerrequisitos para sistemas multiagentes.
- Reactividad, los agentes perciben estímulos de su entorno y reaccionan ante ellos posiblemente para cambiar lo que ocurre. La ejecución remota consiste en que un agente se transfiere a un sistema remoto donde es activado y ejecutado en su totalidad.
- Veracidad de los agentes, se basa en no comunicar información falsa a propósito (se supone). La benevolencia radica en la ayuda a otros agentes y no entra en conflicto con sus propios objetivos.
- Racionalidad, un agente siempre actúa de forma racional con miras a cumplir sus objetivos.
- Inteligencia, es otra característica muy significativa. Esta se refiere al método utilizado para desarrollar la lógica del agente o la inteligencia y está estrechamente relacionada con los lenguajes de agentes donde predominan dos aspectos: la creación de contenido pragmático del agente y la representación del conocimiento que proporciona los medios para expresar objetivos, tareas, preferencias y vocabulario apropiado para varios dominios.
- Aprendizaje adaptativo, se cambia su comportamiento basado en las experiencias previas.
- Operación asíncrona, característica que radica en el hecho de que el agente puede ejecutar tareas totalmente desacoplado de sus usuarios o de otros agentes, lo que significa que puede actuar a raíz de un evento particular. Esta hace que sean sistemas muy vulnerables a ataques externos.

2.2.2. Clasificación de los agentes

Podemos realizar una primera clasificación de los agentes atendiendo a su comportamiento. Respecto a la movilidad de los agentes tenemos los siguientes grupos:

- Agentes móviles, que se desplazan de una ubicación a otra durante su ciclo vital.
- Agentes estáticos, que permanecen siempre en la misma ubicación.

En relación a la forma de actuar de los agentes, tenemos:

- Agentes reactivos, que interactúan con su entorno siguiendo un mecanismo de estímulo/respuesta dependiente del estado del entorno en el que se encuentra.
- Agentes deliberativos, que contienen un motor de razonamiento interno que modela el entorno que les rodea.

Por otro lado, podemos realizar una clasificación de los agentes atendiendo a sus características que describíamos previamente. Usando estas características podemos clasificar los agentes en cuatro clases:

- Agentes cooperativos. Son agentes que colaboran con otros agentes.
- Agentes cooperativos con capacidad de aprendizaje. Son agentes que colaboran entre sí pero con cierta autonomía dada por su capacidad de aprendizaje.
- Agentes de interfaz. Son agentes que interactúan con humanos.
- Agentes inteligentes. Son agentes con capacidad de tomar decisiones basadas en el conocimiento.

De esta última clasificación cabe destacar que las categorías no son excluyentes, es decir, un agente puede pertenecer en mayor o menor medida a una o varias categorías. Esta clasificación es conocida con el nombre Nwana. Por ser la más precisa y extendida hemos dedicado especial atención a la descripción de la misma.

La clasificación de Nwana

En la figura 2.1 podemos ver una clasificación de los agentes propuesta por Hyacinth S. Nwana [21], según la cual existen siete tipos de agentes; agentes cooperativos, agentes de interfaz, agentes móviles, agentes informativos (conocidos como de Interenet), agentes reactivos, agentes híbridos y agentes inteligentes.

Estos agentes pueden coexistir en sistemas de agentes homogéneos, formados por agentes de características similares o heterogéneos en los que hay agentes de diversos tipos. También es necesario destacar que los agentes pueden competir entre ellos, ya que sus intereses pueden no coincidir. Este último aspecto es muy importante y es necesario tenerlo en cuenta a la hora de diseñar un sistema multiagente, ya que es posible que agentes antagónicos puedan actuar de forma maliciosa entre sí. A continuación vamos a analizar más detalladamente cada uno de los tipos de agentes propuestos por la clasificación de Nwana.

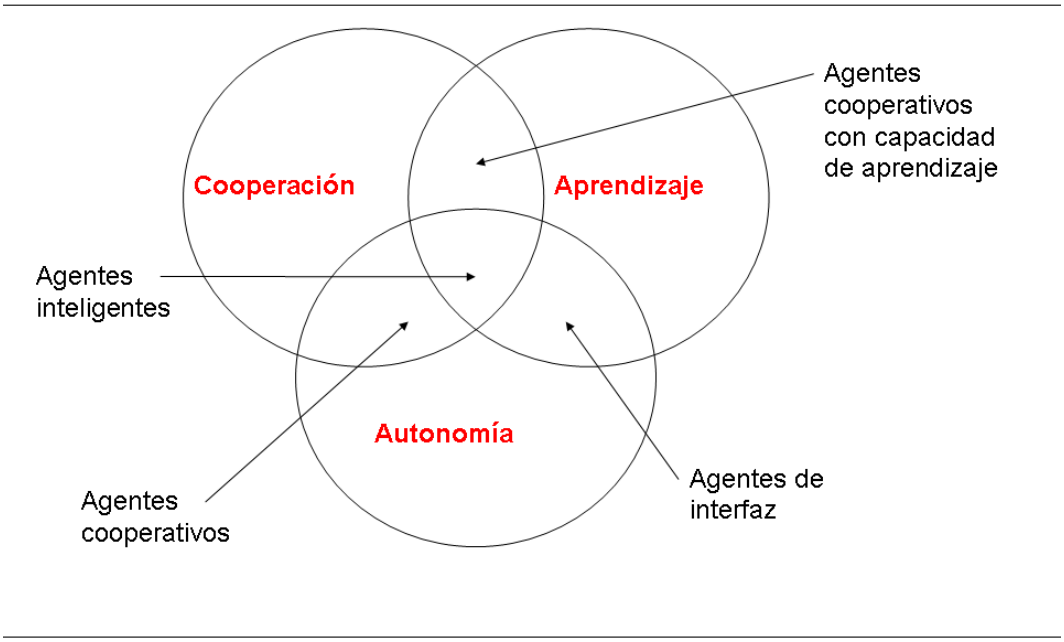

Figura 2.1: Clasificación de agentes software propuesta por Nwana.

Agentes cooperativos

El aspecto más importante de este tipo de agentes es, sin duda, la cooperación con otros agentes. Por ello, los agentes cooperativos se usan para enfrentar diversos problemas que resultan demasiado extensos para ser resueltos por un agente centralizado, ya sea por la escasez de recursos o por el riesgo que ello pueda conllevar. También resultan de gran utilidad para la resolución de otros problemas que hacen uso de fuentes de datos distribuidas, para sistemas que hacen uso de sistemas expertos distribuidos, para sistemas que requieren modularidad son bastante adecuados sobre todo mejorando parámetros de velocidad (haciendo uso del paralelismo), confiabilidad (gracias a la redundancia), flexibilidad y reusabilidad del conocimiento (mediante la compartición de recursos). Sin embargo, el diseño de sistemas basados en agentes cooperativos implica una serie de dificultades: la coordinación entre agentes es esencial en este tipo de sistemas, ya que permite la resolución de un gran número de problemas.

Un diseño erróneo de la coordinación puede derivar en problemas de anarquía e interbloqueos. Por otro lado, es necesario establecer si los agentes negocian entre ellos de manera confiable o si pueden engañarse. Además, la coordinación es necesaria desde el punto de vista de las restricciones de recursos y tiempo. También pueden surgir problemas de estabilidad, escalabilidad y rendimiento, además de los problemas de validación de los sistemas de agentes. Es necesario desarrollar una técnica de verificación y evaluación de sistemas de agentes para determinar si cumplen sus especificaciones y administran

correctamente los eventos no esperados.

Agentes de interfaz

Las características esenciales de los agentes de interfaz son la autonomía y el aprendizaje. Estos agentes colaboran con el usuario. Por lo que no requiere el uso de un lenguaje específico, como en el caso de colaboración entre agentes. Los agentes de interfaz se usan normalmente para ayudar al usuario en el aprendizaje, ya sea en el uso de un sistema operativo, hoja de cálculo u otra aplicación. Este aprendizaje lo lleva a cabo el agente de cuatro maneras posibles. Observando e imitando al usuario, a través de la respuesta del usuario, a través de instrucciones específicas del usuario y preguntando a otros agentes. El objetivo del uso de sistemas de agentes de interfaz es pasar de una comunicación unidireccional, en la que el usuario ordena a la máquina que lleve a cabo una serie de tareas, a una comunicación bidireccional, en la que hombre y máquina colaboran, lo cual permite que el sistema pueda llevar a cabo las tareas tediosas permitiendo ahorrar horas de trabajo al usuario. El diseño de sistemas basados en agentes de interfaz conlleva una serie de problemas, entre los que destacamos el hecho de demostrar que el conocimiento adquirido por los agentes pueden realmente ayudar a reducir la carga de trabajo del usuario y el determinar qué técnicas de aprendizaje son más adecuadas para qué dominios y por qué.

Agentes móviles

Los agentes móviles son procesos software capaces de desplazarse dentro de redes de área amplia (Wide Area Networks, WAN), como Internet, interactuando con hosts remotos, recogiendo información para el usuario y retornando al lugar de origen habiendo llevado a cabo las tareas solicitadas por el usuario. Todas estas características pueden habilitar la reducción de los costes de comunicación, el uso de recursos locales bastante escasos, la coordinación sencilla, etc. El agente lleva a cabo varias tareas, aunque debido a la computación asíncrona, sólo será necesario coordinar al propio agente, no a las múltiples tareas que realiza de manera independiente . Sin embargo, el diseño de sistemas basados en agentes móviles conlleva una serie de problemas, entre los que destacamos el del transporte, ya que es necesario determinar cómo realiza el agente la transferencia de un host a otro, cómo empaqueta su información para poder trasladarla consigo, etc. También hemos de considerar el problema de la seguridad, ya que es necesario evitar que los agentes puedan infectar los sistemas con virus o incluso actuar ellos mismos de manera maliciosa. La seguridad engloba tanto a la autenticación del agente para establecer mecanismos que permitan, determinar su identidad, a quién representan y si está libre de virus, como a la privacidad, basado en asegurar que los agentes no vulneren las leyes de privacidad establecidas.

Agentes de información/agentes de internet

Los agentes de información pueden ser móviles o estáticos, con capacidad de desplazarse por la red para obtener la información o recopilar la misma desde una ubicación determinada. Este tipo de agentes surgieron debido a la creciente demanda de herramientas de ayuda de gestión de gran cantidad de información que existe actualmente. Los

agentes se encargan de gestionar, manipular y recopilar información de diversas fuentes distribuidas. El motivo de desarrollar agentes de información es doble, primero, la necesidad de gestionar la explosión de información que existe actualmente, y segundo, los beneficios que puede reportar el negocio de la información en Internet. El problema clave de los agentes estáticos es que deben mantener información actualizada de sus índices hacia los datos, lo cual es muy complicado en un entorno caótico como Internet. Los problemas derivados del diseño de sistemas de agentes de información son similares a los que surgen con los agentes móviles y los agentes de interfaz, siendo los agentes de información estáticos los que presentan problemas más similares a los de los agentes de interfaz.

Agentes reactivos

Los agentes reactivos representan una categoría especial de agentes que no poseen un modelado simbólico de su entorno. En su lugar, actúan respondiendo a estímulos que reciben del entorno en el que se encuentran. El punto clave de los agentes reactivos es que no poseen una especificación, a priori, de su comportamiento. Pueden verse como una colección de módulos independientes que actúan de manera autónoma y son responsables de tareas determinadas. La comunicación entre estos módulos es mínima y de bajo nivel, por lo que no existe un modelo global del sistema. Por otra parte, los agentes reactivos tienden a operar como si se tratasen de sensores, usando una representación de los datos similar a la de éstos en contra de lo que hacen otros tipos de agentes, que usan una representación simbólica de alto nivel. La hipótesis clave sobre la cual gira la idea de los agentes reactivos es que puede desarrollarse un sistema inteligente mediante agentes simples que no tengan modelos simbólicos internos y cuya inteligencia se base en el comportamiento derivado de la interacción entre los módulos que lo forman. Un beneficio derivado del desarrollo de sistemas de agentes reactivos es que deberían ser más robustos y tolerantes a fallos que otros sistemas de agentes. Por otro lado los sistemas obtenidos son más flexibles y adaptables. El diseño de sistemas de agentes reactivos conlleva una serie de problemas: el reducido rango de aplicaciones basadas en agentes reactivos, la metodología de desarrollo para este tipo de sistemas no está claramente definida y la existencia de una serie de problemas no funcionales que deben ser resueltos, como la escalabilidad y el rendimiento.

Agentes híbridos

Un agente híbrido es aquel que reúne características de varios de los diferentes tipos de agentes. El desarrollo de sistemas basados en este tipo de agentes suele implicar un diseño específico para una aplicación concreta sin una base determinada, con los problemas que ello conlleva.

Esta clasificación tan detallada de todos los tipos de agentes software abre las puertas a un gran número de posibles aplicaciones de esta tecnología en diversos tipos de escenarios, y más aún si consideramos el caso de los sistemas de agentes híbridos, en los que contamos con un elevado número de posibles combinaciones.

2.3. Sistemas multiagente

2.3.1. Definición

Se define un sistema multiagente como un conjunto de agentes que trabajan en colaboración para resolver problemas y que se caracteriza porque cada agente no tiene capacidad por sí solo de solucionar el problema, no existe un sistema global de control, los datos están distribuidos y la computación es asíncrona. Es importante distinguir entre sistema multiagente y sistema basado en agentes. En un sistema basado en agentes aunque se usan también agentes para resolver un problema, éste podría haber sido abordado sin hacer uso del propio paradigma por no cumplir los requisitos planteados.

Un sistema multiagente está constituido por un conjunto de entidades inteligentes, a los que conocemos por "agentes", que coordinan sus habilidades para la resolución de problemas individuales o globales. Estos sistemas considerados como un todo, exhiben características particulares que se presentan a continuación.

2.3.2. Clasificación

Los sistemas multiagente pueden ser homogéneos o heterogéneos. Los primeros están compuestos por agentes de un solo tipo, y como cabe esperar los segundos se componen de diferentes tipos de agentes. Mientras que los sistemas homogéneos están pensados para desarrollar aplicaciones específicas determinadas por el tipo de agentes que componen el sistema, los sistemas heterogéneos tienen un propósito más amplio proporcionando una serie de beneficios con respecto a los homogéneos. Permiten desarrollar aplicaciones que proporcionen servicios de valor añadido basadas en los diferentes sistemas que cooperan, y sirven para solucionar problemas que surgen al usar sistemas antiguos y al interoperar con estos mediante el uso de agentes específicos que se encargan de traducir los mensajes de otros agentes al sistema de comunicación propio del sistema antiguo y viceversa.

Existen dos posibles arquitecturas para el sistema multiagente, una en la que cada agente gestiona su propia coordinación con los demás y otra en la que existen agentes especiales encargados de coordinar a los diferentes agentes. La primera aproximación tiene la desventaja de que la escalabilidad no está asegurada, por lo que la segunda suele ser la más aceptada. En esta última, los agentes encargados de la coordinación suelen realizar funciones como encontrar agentes que proporcionen servicios determinados, establecer la comunicación entre diferentes entornos y asegurar que la comunicación entre agentes sea correcta. Los sistemas multiagente proporcionan una serie de ventajas con respecto a otras aproximaciones de diseño. Esto permite resolver problemas demasiado grandes para un agente centralizado, evitan que exista un único punto de error, permiten resolver problemas inherentemente distribuidos y ayudan a resolver problemas donde el "conocimiento experto" es distribuido.

2.3.3. Características de los sistemas multiagentes

Organización social

Representa la forma en que está constituido el grupo de agentes en un instante dado. La organización social está relacionada con la estructura de los componentes funcionales del sistema, sus características, sus responsabilidades, sus necesidades y la manera de realizar sus comunicaciones. Esta organización puede ser estática o dinámica, dependiendo de las funciones o tareas de cada agente.

Se puede considerar que una sociedad de agentes está constituida por tres elementos: un grupo de agentes, un conjunto de tareas a realizar y un conjunto de recursos. La realización de las tareas por parte de los agentes puede organizarse de varias formas, como por ejemplo que cada agente ejecute una de las tareas, o bien, las tareas se divididan en subtareas, por medio de algún mecanismo de descomposición de problemas, siendo estas subtareas las realizadas por los agentes. Las tareas que debe realizar un agente dependerán, entre otros factores, del rol que asume este agente en el sistema. Un claro ejemplo lo tenemos en un sistema que representa a una oficina. En este sistema un agente persona asume el rol de secretaria y realiza las labores relacionadas con ese rol; este mismo agente podría asumir el rol de jefe y realizaría labores muy diferentes, relacionadas con su nuevo rol. Para la realización de tareas un agente puede necesitar recursos del sistema, en este caso tiene que coordinarse con los otros agentes del sistema que deseen usar el mismo recurso.

La organización en los sistemas multiagente depende del tipo de comunicación y el modo de cooperación entre agentes, así como del tipo de agentes que conforman el grupo. En general se pueden distinguir tres tipos de configuraciones organizacionales:

- Estructura Centralizada: En este tipo de configuración existe un agente que controla la interacción de los demás agentes del sistema, porque tiene la información o la funcionalidad para hacerlo.

- Estructura Horizontal: Este tipo de configuración existe cuando todos los agentes que integran un sistema están al mismo nivel, es decir, no hay ningún agente que haga las veces de maestro o supervisor, ni tampoco agentes esclavos.

- Estructura Jerárquica: Esta configuración existe cuando los agentes trabajan diferentes niveles de abstracción de un problema, es decir, la configuración es de niveles. En un mismo nivel se establece una configuración horizontal, si hay más de un agente. Para resolver un problema cada agente divide el problema en subproblemas que él puede solucionar con la cooperación de los agentes que están al mismo nivel y subproblemas que sabe que los agentes de niveles inferiores de la jerarquía pueden hacerlos por sí mismos.

- Estructura "ad hoc": Esta configuración puede ser una mezcla de las tres anteriores, se caracteriza porque la dinamicidad de la estructura está regida por el ajuste mutuo entre los pequeños grupos de agentes en el sistema.

Escoger una u otra estructura de organización depende de las funciones que deben cumplir los agentes del sistema, de sus características y de la complejidad que requiera el sistema.

Cooperación

En un sistema multiagente existen dos tipos de tareas a realizar: las tareas locales y las tareas globales. Las tareas locales son las tareas relacionadas con los intereses individuales de cada agente y las tareas globales son las tareas relacionadas con los intereses globales del sistema. Estas tareas globales son descompuestas y cada subtarea es realizada por un agente, de acuerdo a sus habilidades y bajo el supuesto de que la integración de la solución de las subtareas llevará a la solución global. La descomposición de la tarea global no necesariamente garantiza la independencia de cada una de las subtareas, por ello se necesitan mecanismos de cooperación que permitan compartir resultados intermedios que lleven al progreso en la resolución de las tareas de otros agentes y al progreso de la solución global que debe alcanzar el sistema.

Para que los agentes puedan cooperar de manera eficiente, cada uno de ellos debe tener ciertas características:

- Es necesario tener un modelo bien definido del mundo, que le permita localizar a los demás agentes, saber cómo comunicarse con ellos, qué tareas pueden realizarse, etc.
- Así como la posibilidad de integrar información de otros agentes con la suya, para formar conceptos globales o conocimiento conformado por varios agentes.
- Y el poder interrumpir un plan que se esté llevando a cabo para ayudar o atender a otros agentes para que puedan cooperar entre sí cuando los agentes lo necesiten.

Esta cooperación depende mucho de la configuración organizacional del grupo de agentes. Si la estructura es centralizada los agentes dependientes piden colaboración prácticamente de forma permanente al agente maestro, si la estructura es jerárquica, la cooperación puede hacerse por niveles (en un mismo nivel) o de niveles superiores a niveles inferiores y si la estructura es horizontal la cooperación se hace entre todos los agentes.

Existen varios modelos de cooperación, dentro de los cuales se pueden mencionar:

- Cooperación compartiendo tareas y resultados: Los agentes tienen en cuenta las tareas y los resultados intermedios de los demás para realizar las tareas propias.
- Cooperación por delegación: Un agente supervisor o maestro descompone una tarea en subtareas y las distribuye entre los agentes esclavos. Después, el supervisor integra las soluciones para hallar la solución al problema inicial.
- Cooperación por ofrecimiento: Un agente maestro descompone una tarea en subtareas y las difunde en una lista a la que tienen acceso los agentes que integran el sistema, esperando que ellos ofrezcan su colaboración de acuerdo a sus habilidades. El supervisor escoge entre los ofrecimientos y distribuye las subtareas.

Coordinación

La coordinación entre un grupo de agentes les permite considerar todas las tareas a realizar y coordinarlas para no ejecutar acciones no deseables, por ejemplo:

- Los agentes no generen y comuniquen subsoluciones que lleven al progreso en la solución de un problema.
- Los agentes no deben generar resultados redundantes.
- Distribución inapropiada de la carga de trabajo entre los agentes. Esta coordinación está relacionada con la planificación de acciones para la resolución de tareas, porque estos planes permiten:
  - Conocer a alto nivel y predecir el comportamiento de otros agentes del sistema.
  - Intercambiar resultados intermedios que lleven al progreso en la solución de la tarea global.
  - Evitar acciones redundantes.

Hay varios modelos de coordinación de acciones entre agentes, entre los que destacamos:

- Coordinación Global: Cuando el sistema multiagente determina y planifica globalmente las acciones de los diferentes agentes.
- Coordinación Individual: Cuando el sistema multiagente le da completa autonomía a los agentes, es decir, cada agente decide qué hacer y resuelve localmente los conflictos que detecte con otros agentes. Además de los modelos existen dos tipos de coordinación:
- Coordinación orientada por los problemas: En este tipo de coordinación, los agentes deben coordinar los planes de realización de acciones para prevenir interbloqueos, repetición de acciones y creación de inconsistencias.
- Coordinación orientada por la cooperación: En este tipo de coordinación los agentes no se coordinan a nivel de planes, sino a nivel de acciones. Esto significa que los agentes se coordinan en el momento de ejecutar acciones.

Negociación

Para que los mecanismos de cooperación y coordinación sean exitosos en un sistema de agentes que actúan independientemente, debe existir un mecanismo adicional, por medio del cual los integrantes de un sistema se puedan poner de acuerdo cuando cada agente defiende sus propios intereses, llevándolos a una situación que los beneficie a todos teniendo en cuenta el punto de vista de cada uno. Este mecanismo es llamado negociación.

Los procesos de negociación tienen como resultado la modificación o confirmación de las creencias de cada agente involucrado, en lo relacionado con los demás agentes y con el mundo en el que se desenvuelve.

La negociación se puede mirar bajo una perspectiva racional, la cual describe la negociación como un proceso de seis pasos que consisten en definir el problema, en identificar los aspectos que lo representan, en ponderar los diferentes criterios, generar alternativas, evaluar estas alternativas, y por último formular la solución.

En los sistemas multiagentes la mayoría de las veces el problema está bien definido, es decir, el primer paso no es relevante. Los demás pasos se pueden o no seguir, dependiendo del problema para el cual un grupo de agentes busca una solución.

Los mecanismos de negociación utilizan diferentes reglas en lo relacionado con la distribución de recursos del sistema:

- Regla equitativa: consiste en dividir los recursos disponibles en proporción a los aportes de cada miembro de grupo.
- Regla igualitaria: establece una distribución de recursos a partes iguales para todos los miembros.
- Regla según las necesidades: consiste en la distribución de acuerdo a las necesidades individuales de cada miembro del grupo.
- Regla según el pasado: rige la distribución siguiendo patrones o experiencias en negociaciones pasadas.

Hay otro tipo de reglas usadas en negociación, relacionadas con la toma de decisiones:

- Consenso o unanimidad, se toma una decisión cuando todos los miembros de un grupo están de acuerdo con dicha decisión. Es muy posible que se llegue a esto después de negociar varias veces la decisión.
- Se decidirá cuando la mayoría de los miembros de un grupo está de acuerdo con dicha decisión. La definición de "mayoría" depende del sistema y se puede relacionar con el número de votos a favor de una decisión o con el peso de tales votos.

La negociación se caracteriza por tener los siguientes elementos: un número adecuado de agentes involucrados en el proceso y un conjunto mínimo de acciones que se llevan a cabo en el proceso, como: proponer, evaluar, refutar, contraproponer, aceptar, rechazar, modificar, etc. Este conjunto es llamado lenguaje: "El principal componente de la negociación como actividad social es el lenguaje". Este conjunto de acciones puede ser visto como un conjunto de actos de habla con una lógica y una semántica especial, en el que se propone un lenguaje para negociación, basado en actos de habla.

Para que una negociación sea exitosa es necesario un protocolo que facilite y en lo posible garantice la convergencia de ideas a una solución común. Un protocolo establece un conjunto de pasos que debe seguir un proceso de negociación, así como las posibles respuestas de un agente a las acciones de otro agente.

Control

El control es el mecanismo básico que proporciona apoyo para la implementación de mecanismos de coordinación en un sistema multiagente. El control se relaciona directamente con la determinación de las subtareas más importantes a realizar en un momento dado. Así como la determinación del contexto en que deben ser usados dentro de la solución de cada subtarea. También es importante estimar el tiempo de generación de la solución por cada subtarea. Así como evaluar si se ha generado la solución de un problema.

El control puede ser considerado desde dos puntos de vista, por un lado el control global y por otro el control local. El control global se relaciona con tomar decisiones basándose en datos obtenidos y consolidados a partir de la información de todos los agentes del sistema, mientras que el control local se relaciona con tomar decisiones basándose solo en datos locales. Estos dos controles deben ser balanceados por varias razones: En primer lugar, el aumento del poder de control local nos lleva a que los agentes actúen eficientemente desde el punto de vista "rapidez en la toma de decisiones y en la ejecución de tales decisiones". Pero se puede llegar a la pérdida de mecanismos de cooperación efectivos y como consecuencia a la realización de tareas no deseables. A su vez, el aumento del poder del control global nos puede llevar a aumentar los costos de computación del sistema, porque como consecuencia cambia dinámicamente la información de los agentes. Para mantener la coherencia en la información global se deben revisar e incorporar de forma constante los cambios en cada uno de los agentes.

Para mantener el balance entre el control global y el control local, los componentes del control local de cada agente deben proveer al control cooperativo una descripción de su estado actual y de actividades esperadas. A la vez que han de interactuar con el control cooperativo para modificar las actividades locales y estar más "on-line" con las necesidades de los otros agentes. Y por su puesto, ser suficientemente sofisticado en la toma de decisiones de manera que pueda tolerar cierto nivel de error en el control cooperativo.

Existen diferentes mecanismos y estrategias para la implementación del control en un sistema multiagente. El mecanismo básico propuesto es la creación y manejo de estructuras a las que puedan acceder todos los agentes y en las que se representen, organicen e integren, a alto nivel, las metas globales y locales del sistema. Cada una de estas metas se relaciona con la intención de realización de alguna tarea. La estructura global de metas no necesita ser totalmente desarrollada para la resolución distribuida de problemas, la estructura puede ser construida a medida que la solución del problema progresa.

Para poder manejar estas estructuras de objetivos, los agentes que integran el sistema multiagente deben tener la habilidad de transmitir sus estructuras locales de metas. La transmisión de las metas puede indicar a los otros agentes qué tipo de información es más necesaria para el desarrollo de una solución. Por otro lado, la generación de submetas a partir de las metas facilita la generación de resultados intermedios que lleven a la consecución de la meta inicial. De esta forma, la incertidumbre en el control cooperativo disminuye porque se genera más información y hay una mayor comunicación. Sin embargo, la comunicación de estructuras de metas genera costos adicionales de comunicación en el sistema.

A partir de la información de la estructura de metas se debe decidir cuáles son las metas de mayor importancia para su realización. Para ello es importante distinguir entre la creencia local en un resultado parcial (creencias) y la importancia de incorporar tal resultado en una solución (evaluación global de la importancia de un resultado). La evaluación local de la importancia de metas externas, determina el balance entre responder a las necesidades de otros agentes y a las propias. La decisión del balance preciso depende de cómo los agentes, desde su perspectiva local, juzguen sus resultados parciales y la importancia de sus metas desde una perspectiva cooperativa. Entre las estrategias para la implementación de los mecanismos de control se mencionan:

- Metacontrol estático asociado con la estructura organizacional de un sistema : Para

alcanzar una resolución cooperativa óptima es necesario eliminar la incertidumbre que se genera de manera natural en un sistema distribuido, dado que no se cuenta en todo momento con la información actualizada de cada uno de los componentes del sistema. Esto implica que cada decisión debe ser reevaluada constantemente a la luz de la nueva información generada. Pero esto es bastante costoso desde el punto de vista de procesamiento y comunicaciones.

Otra opción, que no requiere reevaluación constante y por ello es menos costosa, es la toma de decisiones de control cooperativo por medio de un proceso de decisión de dos niveles. Un primer nivel de diseño organizacional se refiere a decisiones estratégicas que no necesitan ser constantemente reevaluadas para obtener un funcionamiento razonable. Y otro nivel, llamado nivel de agente, que se refiere a decisiones tácticas que son constantemente reevaluadas. El nivel de diseño organizacional define políticas a largo plazo que pueden ser usadas a nivel de agente, para restringir el conjunto de decisiones o información que debe ser evaluado para tomar decisiones. Estas políticas definen la estructura organizacional del sistema distribuido para resolución de problemas. De alguna manera, la estructura organizacional representa decisiones precomputadas acerca de la estructura de metas de cada agente. Para diferentes aspectos de control cooperativo, las políticas individuales especifican estrategias globales a largo plazo que pueden ayudar a la solución coherente de problemas entre agentes. Las políticas guían y restringen las decisiones de control local para que estén en consonancia con la estrategia global.

- Metacontrol dinámico asociado con la planificación global-parcial: Para que los agentes cooperen de manera efectiva necesitan una vista razonablemente precisa de la estructura de metas de otros agentes. Estos necesitan ser capaces de reconocer y anticipar dónde hay interacción o subproblemas redundantes, cuándo planea trabajar sobre los problemas, estudiar la dificultad de los problemas y la flexibilidad que tiene el agente para reasignar sus actividades. Un conjunto de preguntas interrelacionadas se derivan de manera natural: qué aspectos dinámicos de la estructura de metas son importantes, la precisión de la vista global de metas, cómo ha de capturarse la evolución de la estructura de metas. Las propuestas para proveer y operar sobre esta información son guiadas por dos principios. El primero se basa en satisfacer la vista del control cooperativo y en que los agentes no tengan que ser totalmente coherentes en su comportamiento, este principio es importante en situaciones donde el control permite reducciones significativas en la cantidad de cómputo y procesamiento requerido para implementar los algoritmos de control cooperativo. El segundo principio es que el control cooperativo efectivo del control local sofisticado. Los agentes deben ser capaces de representar y razonar no sólo sobre sus actividades locales sino también de acomodarse a los requisitos de otros agentes. La planificación global-parcial se basa en la vista de metas a corto plazo de los agentes. Esta vista también contiene el orden esperado en el cual los agentes intentarán alcanzar las metas y estima el tiempo de ejecución y medidas de calidad de la solución.

  Se puede considerar esto como un plan de alto nivel para la resolución local de problemas. Hay una tensión obvia entre la posibilidad de reasignación de problemas globales y el grado de predicción en el plan de alto nivel. A menor predicción, se

> requiere mayor comunicación y procesamiento para alcanzar un comportamiento cooperativo coherente. Los agentes intercambian esta vista parcial y aumentada de la estructura local de metas esperada. Por medio de la combinación de fragmentos de estructuras de metas de diferentes agentes un agente construye planes de metas globales y parciales. Un agente usa esta información para comprender cómo sus actividades se relacionan con otros agentes.

Esta información guía a un agente, para reorganizar sus metas locales, buscando resultados tales como: explotar los resultados parciales disponibles de otros agentes, proporcionar resultados parciales que puedan restringir el campo de búsqueda de la solución a las metas de otros agentes, evitar soluciones redundantes a metas, excepto cuando sea deseable, y por supuesto reevaluar la importancia de alcanzar una meta.

2.3.4. Estándares de sistemas multiagente

En la actualidad no existe ningún tipo de estándar formal con respecto a los sistemas multiagentes propuestos por los organismos de estándares más conocidos (ANSI, ISO, IEEE,ETSI y CEN). Sin embargo, sí que existen consorcios formados por diferentes organizaciones con la intención de estandarizar el sector.

ARPA-KSE son las siglas de "Knowledge sharing effort". Este es un consorcio para el desarrollo de normas que faciliten la compartición y reutilización de bases de conocimiento y de sistemas basados en el conocimiento. Han desarrollado el KQML Ontolingua (Knowledge Quering and Manipulation Language) que es una herramienta para la definición de ontologías.

The Agents Society: Creada para extender el desarrollo de sistemas de agentes, se encarga del intercambio y recopilación de información sobre agentes.

OMG (Object Management Group): Han desarrollado MASIF (Mobile Agent System Interoperabilities Facility), que es una definición de interfaces para la transferencia y localización de agentes en sistemas multiagente. Está implementado sobre CORBA.

FIPA (Foundation for Intelligent Physical Agents). Recoge todas las vistas que se tienen de un sistema de agentes (gestión, seguridad, movilidad, comunicación, etc).

De estos consorcios vamos a destacar FIPA, que es ampliamente reconocido internacionalmente y es el estándar que sigue Jade [22], el framework usado para las implementaciones realizadas en este trabajo de tesis.

El estándar FIPA

FIPA es un consorcio internacional creado en 1995 y que está formado por distintas universidades y empresas. El objetivo del consorcio es crear un estándar público, para lo que proporciona una serie de especificaciones normativas (referencias tecnológicas) e informativas (referencias de aplicaciones). Las referencias tecnológicas especifican una plataforma que permite el desarrollo de sistemas multiagente abiertos, mientras que las referencias a aplicaciones son aplicaciones reales que motivan el uso de estos sistemas. Las referencias tecnológicas establecen normas de comportamiento e interfaces externas que aseguran la interoperabilidad con otros sistemas similares sin imponer una implementación específica.

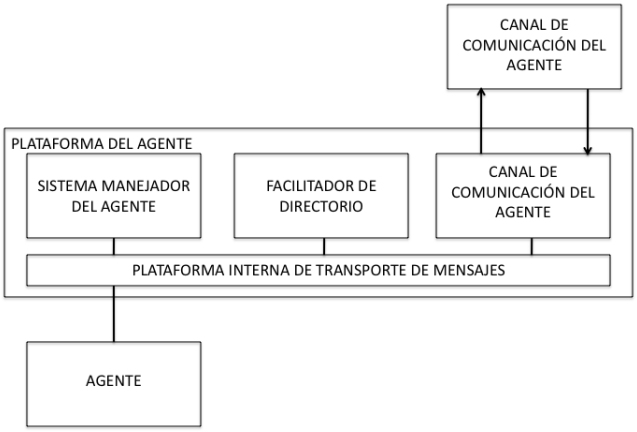

Figura 2.2: Plataforma de agentes del estándar FIPA.

El estándar FIPA describe también una plataforma de agentes, definiendo únicamente el comportamiento externo (interfaz) de un sistema abierto para que sistemas heterogéneos puedan interactuar. FIPA establece un modelo lógico referente a la creación, destrucción, registro, localización y comunicación de agentes.

En la figura 2.2 podemos ver la arquitectura de una plataforma FIPA y los elementos que la forman.

- Agent Platform: Infraestructura que permite la generación y ejecución de agentes.
- Agent Management System (AMS): Es el elemento de gestión principal que mantiene el estado de la plataforma y de los agentes que ésta contiene. El AMS ofrece los servicios de creación, destrucción y control del cambio de estado de los agentes, así como la supervisión de los permisos para el registro de nuevos agentes, el control de la movilidad de los agentes, la gestión de los recursos compartidos y la del canal de comunicación.
- Servicio de Nombres (ANS) o "páginas blancas" (Nombre - Dirección).
- Directory Facilitator (DF): Ofrece un servicio de "páginas amarillas", en este los agentes se registran indicando los servicios que ofrecen, de esta manera, cuando un agente pregunta por un servicio, el DF le responde con la lista de agentes que ofrecen dicho servicio.
- Agent Communication Channel (ACC): Realiza el envío de mensajes entre agentes del sistema u otra plataforma diferente, de manera asíncrona.
- Internal Platform Message Transport (IPMT): Es una infraestructura de comunicaciones que permite que dos agentes se comuniquen.

Servicios de la plataforma FIPA. Además de la plataforma de agentes, FIPA describe la interacción con los usuarios definiendo una serie de servicios: UPS (User Personalization Service): Servicio que se encarga de registrar, actualizar, consultar y gestionar los modelos de los usuarios. El UDMS (User Dialog Management Service): Servicio que hace de envoltorio software de interacción con el agente, permitiendo la interacción entre agentes y usuarios humanos mediante mensajes ACL.

Por otro lado FIPA también define un servicio de ontología. Una ontología es un conjunto de símbolos o términos junto a su correspondiente interpretación o significado. La finalidad de la ontología es la de facilitar la comunicación y la compartición de la información entre diferentes sistemas. En FIPA, cuando un agente se registra en el DF informa de las ontologías que conoce.

El servicio de ontología de FIPA realiza las siguientes funciones: mantiene un conjunto de ontologías de uso público accesible a los agentes. También traduce expresiones entre diferentes ontologías. Responde a consultas sobre términos de las ontologías que gestiona. Facilita la identificación y uso de ontologías compartidas entre los agentes. Descubre nuevas ontologías y las pone a disposición de todos los agentes.

Comunicación entre agentes en FIPA. El estándar FIPA ha desarrollado el lenguaje ACL (Agent Communication Language), una evolución de KQML, que comunica los agentes mediante mensajes y cuya semántica se define formalmente haciendo uso de lógica modal. Para dar soporte a la comunicación entre agentes, FIPA define el Agent Communication Channel (que ya analizamos al describir la arquitectura de la plataforma de agentes).

Para la comunicación se usan mensajes ACL basados en actos comunicativos que se encargan del paso de información, solicitud de la información, negociación, realización de acciones y manejo de errores. FIPA ha definido unos protocolos que describen diferentes patrones de comunicación entre agentes y que permiten guiar las conversaciones.

La estructura del mensaje ACL contiene los elementos necesarios para la comunicación eficiente entre los agentes. Los parámetros de un mensaje ACL son:

- Performative: Denota el tipo de acto comunicativo del mensaje ACL. Es un parámetro obligatorio en todo mensaje ACL.
- Sender: Denota la identidad del emisor del mensaje. Es decir, el nombre del agente que envía el mensaje.
- Receiver: Denota la identidad de los receptores del mensaje. Puede tratarse de un receptor único o de un conjunto no vacío de receptores.
- Reply-to: Indica el agente al que deben dirigirse los siguientes mensajes en lugar de al emisor.
- Content: Denota el contenido del mensaje expresado en lenguaje formal. El significado del contenido del mensaje deberá ser interpretado por el receptor.
- Language: Denota el lenguaje en el que se expresa el contenido del parámetro anterior.

- Encoding: Denota la codificación usada para el contenido.
- Ontology: Denota la ontología usada para la expresión del contenido.
- Protocol: Denota el protocolo de interacción empleado por el emisor del mensaje.
- Conversation-id: Identificador que acompaña a todos los mensajes correspondientes a una conversación.
- Reply-with: Expresión que será usada por el agente que responde para identificar el mensaje.
- In-reply-to: Denota una expresión relativa a una acción anterior a la que responde el mensaje.
- Reply-by: Denota una fecha u hora que indica el límite temporal en el que el emisor espera recibir una respuesta.

JADE: Un framework para el desarrollo de agentes basado en FIPA

JADE (Java Agent DEvelopment framework), inicialmente desarrollado por Telecom Italia Lab (TILAB), se continúa desarrollando por el JADE Internacional Board fundado en 2003 por TILAB y Motorola con el objetivo de continuar su colaboración en el proyecto LEAP. Es un consorcio sin ánimo de lucro cuya misión es promover, dirigir e implementar la evolución de JADE y cuyo objetivo es la adopción de JADE por la industria de la comunicación móvil como middleware estándar para aplicaciones de agentes móviles inteligentes P2P con capacidad de interoperar con diferentes terminales y redes.

JADE es un framework de código abierto completamente desarrollado en Java que simplifica la implementación de sistemas multiagente a través de su middleware que cumple el estándar FIPA. Además JADE posee un conjunto de herramientas para el proceso de depuración y de despliegue. La plataforma de agentes puede distribuirse a través de varias máquinas y la configuración puede controlarse mediante su GUI remoto, permitiendo modificar la configuración en tiempo de ejecución moviendo agentes de una máquina a otra cuando sea necesario.

La arquitectura de comunicación ofrece un sistema de mensajería en el que JADE gestiona la cola de mensajes ACL entrantes de cada agente de manera privada, permitiéndoles a éstos acceder a dicha cola usando varios modos: bloqueo, polling, timeout o mediante coincidencia de patrones. JADE posee una implementación completa del modelo de comunicación de FIPA en la cual los componentes están claramente diferenciados y completamente integrados: protocolos de interacción, envoltorios, ACL, lenguajes de contenido, esquemas de codificación, ontologías y protocolos de transporte. El mecanismo de transporte se adapta de manera transparente eligiendo el protocolo más adecuado según el caso. Entre los protocolos disponibles se encuentran Java RMI, IIOP y HTTP, permitiendo además añadir protocolos nuevos de manera sencilla como veremos en capítulos sucesivos de esta tesis. La mayoría de los protocolos de interacción definidos por el estándar FIPA se encuentran disponibles y pueden ser instanciados tras definir el comportamiento de cada estado del protocolo.

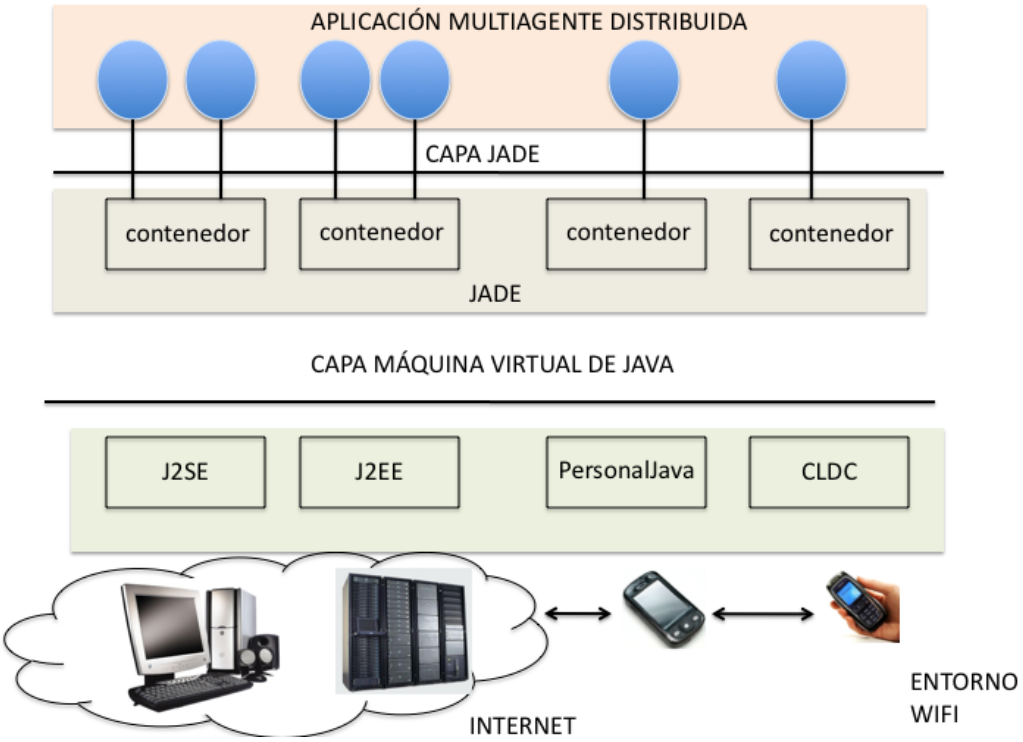

Figura 2.3: Arquitectura de JADE.

JADE posee una ontología de gestión de agentes (SL), además de soportar ontologías y lenguajes de contenido definidos por el usuario que pueden ser implementados y registrados con los agentes además de usados automáticamente por el framework. JADE también está integrado con JESS, lo que permite añadir capacidad de razonamiento a los agentes.

En la actualidad JADE ha sido adoptado por una serie de compañías, de las cuales algunas son miembros del consorcio FIPA. Entre las compañías que usan JADE se encuentran BT, CNET, NHK, Imperial Collage, IRST, KPN, Universidad de Helsinki, INRIA, ATOS y muchas otras.

El framework de JADE incluye tanto las librerías necesarias para el desarrollo de aplicaciones de agentes como el entorno de ejecución que proporciona los servicios básicos que deben activarse para que los agentes puedan funcionar. Cada instancia del entorno de ejecución de JADE se conoce como *contenedor*. El conjunto de todos los contenedores forma una plataforma y proporciona una capa homogénea que oculta a los agentes la complejidad de las capas subyacentes.

JADE es compatible con J2ME CLDC/MIDP1.0 pudiendo funcionar sobre redes GPRS, así como sobre otros tipos de red. Debido a los escasos recursos requeridos por JADE, puede desplegarse en sistemas con restricciones, aunque también puede integrarse en arquitecturas más complejas como .NET o J2EE.

2.4. Trabajos previos en protección de agentes

A pesar de los beneficios que nos proporciona el uso de los agentes móviles, su uso extendido está restringido por razones de seguridad y, en concreto, por la carencia de mecanismos de seguridad apropiados. Entendemos por seguridad toda aquella acción que tiende a garantizar el cumplimiento de alguna de seis propiedades básicas: autenticación, autorización, confidencialidad, integridad, disponibilidad y no-repudio. Mostremos un ejemplo de ello: "Cuando el agente del proceso A se envía a la máquina donde reside el proceso B, hay un potencial obvio para que el agente provoque problemas". Podría potencialmente hacerlo de diferentes maneras. La cuestión de seguridad predomina en las discusiones de agentes móviles. Java es un lenguaje que soluciona algunos agujeros de seguridad. Por ejemplo, Java no tiene punteros, así que es intrínsecamente complicado que una aplicación en Java pueda acceder a un proceso que se encuentra en memoria. Además, las máquinas virtuales de Java poseen una clase llamada SecurityManager que permite establecer una política de seguridad (accesos a recursos,...). Sin embargo, es muy difícil asegurarse que un proceso no use más de un cierto número de ciclos del procesador.

El tener un control de la seguridad es un grave problema, ya que los agentes son programas que viajan de una máquina a otra, de forma similar a como lo realiza un virus. Se deben controlar los siguientes aspectos: protección de la máquina frente a agentes y protección de los agentes contra la máquina. Vamos a describir detalladamente cada una de ellas y las peculiaridades de sus soluciones.

2.4.1. Protección de los agentes contra la máquina

Cuando se transfiere código ejecutable a través de la red, existe la posibilidad de un ataque. Los agentes al igual que ocurre con los programas usados en redes, pueden ser susceptibles a alteraciones, ataques, espionaje, y clonaciones. Es importante ser conscientes de que siempre existe la posibilidad de que un intruso intente afectar la integridad de un sistema y por tanto llevar a cabo las medidas de prevención adecuadas ayuda a minimizar los riesgos. Algunos de los ataques que son ejecutados en contra de agentes son: espionaje de código, espionaje de información, manipulación de código y de información, ejecución incorrecta del código, identidad falsa de un host y negación de ejecución.

Un agente puede llevar consigo información confidencial, por lo que se debe prevenir en todo momento la modificación del código ejecutable. Es posible asegurar la integridad de un agente que proviene de otro host utilizando la firma digital o el cifrado. Sin embargo, es difícil detectar o prevenir los ataques que pueda realizar un host mientras se ejecuta un agente. Los hosts pueden intentar sacar provecho del agente modificando el código, los datos, las comunicaciones o incluso los resultados, ya que, como antes mencionábamos, tienen control total sobre la ejecución del agente.

A este problema se le conoce como el de los "Hosts Maliciosos", este es considerado por muchos autores el más difícil de resolver en lo relacionado con la seguridad en sistemas de agentes móviles [23, 24, 25]

Algunas soluciones propuestas

Para prevenir una alteración en el código y en la información, los agentes se pueden proteger usando métodos criptográficos. De forma similar los agentes pueden ser autenticados con un host o incluso con otros agentes, mediante métodos de clave pública como es la firma digital. De hecho, teniendo en cuenta que los agentes se pueden ejecutar en hosts con diversos grados de confianza, sería ingenuo no esperar un comportamiento malicioso por su parte. Los hosts pueden sacar provecho del agente modificando el código, los datos, el modo de ejecución, el estado, las comunicaciones, el itinerario o incluso los resultados, ya que tienen control total sobre la ejecución. Precisamente esta es la razón por la cual no se pueden evitar ataques de denegación de servicio, puesto que el host tiene en sus manos el código del agente para ejecutarlo a placer, forzándolo a terminar antes de tiempo o incluso a migrar a otro destino. Para un agente móvil es imposible almacenar en claro una clave secreta, puesto que el host tiene acceso de lectura y modificación a la misma [26], para esto es imprescindible disponer de un entorno confiable para realizar las operaciones criptográficas.

Ataques

Como ya mencionábamos en la introducción de esta sección, existen multitud de razones por las cuales un host podría iniciar un ataque contra el agente, e incluso estos ataques podrían ser de índoles muy diversas, bien por obtener beneficios en la ejecución del agente, o simplemente por pura diversión. A continuación vamos a definir los diferentes tipos de ataques que se pueden realizar sobre un agente móvil [24]:

- En el ataque conocido como "denegación de servicio", el host impide la ejecución del agente mediante algún tipo de medio. Anteriormente comentábamos que efectivamente este tipo de ataques es imposible de evitar, por la sencilla razón de que el control de la ejecución del agente es del mismo host. Las únicas soluciones conocidas para esta vertiente del problema radican en la detección y el castigo de los hosts que lo practiquen. Sin embargo, también se consideran ataques de denegación de servicio ciertos cambios aleatorios que pueda realizar un host malicioso en el código o los datos, puesto que esto puede llevar al agente a comportarse de manera inesperada. A diferencia del caso anterior, este tipo de ataques si pueden evitarse mediante el uso de las técnicas desarrolladas en la elaboración de esta tesis, como veremos en los sucesivos capítulos de este trabajo.

- Otro tipo de ataque identificado es el ataque de repudio, consistente en la negación de una acción que tuvo lugar en cierto momento. Sin embargo, este tipo de ataques podemos evitarlos siempre y cuando toda la información intercambiada entre el agente y la agencia esté correctamente firmada por la entidad emisora, pudiendo incluso buscar responsabilidades.

- Como antes mencionábamos, no es posible asegurar la confidencialidad de ciertas partes del agente, que son indispensables, tales como el código, los datos, las comunicaciones o los resultados, ya que estos son indispensables para la ejecución del agente. Sin embargo, mediante escuchas en el código, los hosts podrían utilizar esta

información para obtener cierto beneficio en la ejecución del agente. Este problema plantea serias dificultades, puesto que es muy difícil evitar y/o detectar las escuchas de cualquier dato no cifrado que transporte el agente. Asimismo es una tarea difícilmente evitable el hecho de que el host guarde una copia de los datos cifrados para realizar un posterior análisis criptográfico.

- Íntimamente relacionado con el caso anterior nos encontramos la manupulación. Mediante el uso de ciertas técnicas como el cifrado y la firma digital podríamos asegurar propiedades muy interesantes como la confidencialidad, integridad y autenticidad del código, datos o resultados provenientes de otros hosts. En cambio, resulta considerablemente más complicado detectar o prevenir las manipulaciones realizadas sobre un agente en su ejecución, puesto que el host puede manipular cualquiera de los contenidos del agente durante la ejecución del mismo. Mediante el uso de las técnicas desarrolladas en este trabajo de tesis, conseguimos evitar este tipo de ataques como más adelante se describe.

- Otro tipo de ataques que hay que considerar es el basado en las confabulaciones entre hosts. Este consiste en que ciertos hosts tienen en común la pertenencia al itinerario de un agente y establecen un acuerdo para sacar provecho de la ejecución del agente. Este tipo de ataques es de difícil detección y prevención. Un hecho que demuesta esta afirmación, radica en que todas las propuestas actuales para proporcionar seguridad a los agentes móviles son vulnerables a las confabulaciones. Por otro lado existen técnicas para la protección del itinerario [27] que se podrían emplear en la limitación de los posibles efectos de las confabulaciones. A lo largo de este trabajo de tesis se ha hecho especial hincapié en abordar este problema. Una prueba que lo evidencia la encontramos en los dos mecanismos desarrollados en los siguientes capítulos.

Medidas

Atendiendo a la clasificación de Bierman y Cloete [28] vamos a clasificar las distintas propuestas atendiendo a si están enfocadas a proporcionar la detección de los posibles ataques o a la prevención de los mismos.

Enfocadas a la Detección de Ataques: Dentro de esta categoría nos encontramos aquellas propuestas encaminadas a la detección de las manipulaciones producidas sobre el agente tras la ejecución del mismo. Todas las propuestas enmarcadas dentro de este conjunto pretenden disuadir a los hosts maliciosos de realizar los ataques, puesto que estos podrían ser sancionados. La aplicación de este tipo de medidas resulta más sencilla que las englobadas en la siguiente categoría. Sin embargo, la implantación real de estas implica la necesidad de una TTP (tercera parte confiable) que arbitre entre todas las partes y que pueda llevar a cabo los castigos a los hosts cuando proceda.

- La medida conocida como "Replicación y Voto" introducida por Minsky et al. [29] consiste en la agrupación de los hosts en distintas etapas por las que los agentes van circulando. Esto se ha de realizar con la restricción de que los hosts pertenecientes a una misma etapa han de ser mutuamente independientes y proporcionar los mismos

datos y recursos. La idea consiste en la ejecución paralela de distintas réplicas de los agentes en los hosts y en el envío de otras réplicas hacia la siguiente etapa, de forma que se establecen ciertas etapas de chequeo en las que se comparan los resultados. Para los casos en los que haya muchas muestras a comparar se elegirán los resultados mayoritarios, mientras que los minoritarios se consideran comprometidos. Esta propuesta presenta una serie de inconvenientes a saber. La replicación de un gran número de agentes implica un gasto excesivo de recursos y de tiempo de ejecución, lo cual provoca muchos retrasos. Por otro lado, es interesante enfatizar el hecho de que se presupone que los resultados de todas las réplicas han de ser los mismos si los hosts actúan de forma honesta, es decir, todos los hosts de una misma etapa han de tener los mismos recursos y datos, lo cual se contradice con la independencia de los hosts requerida, es decir, que todos los hosts deben tener intereses distintos para atacar a un agente.

- Estado de referencia. Los agentes pueden llevar consigo fechas de caducidad que permiten la ejecución de una tarea en un tiempo determinado. Este modelo es similar al usado en los certificados digitales. El código del agente que puede ir oculto no tendrá validez una vez que concluya la tarea o la fecha de asignación. En el caso de que se cambie la ejecución y el comportamiento de un agente implementado en un sistema de seguridad, se tiene que conocer el estado o bien el momento y el lugar en que fue alterado. Una manera de protección para este tipo de ataques es la utilización de mecanismos que emplean estados de referencia, lo que significa que son agentes de estado creados por una parte confiable o host de referencia que detectan ataques de modificación o host no autorizados. Un estado de referencia es la combinación de partes variables de un agente móvil, ejecutado por un host, mostrando un comportamiento de referencia. Lo que se puede detectar es la diferencia entre las variables de un agente ejecutado tanto en un host no confiable como en un host de referencia. Los estados de referencia pueden ayudar a medir las diferencias de estados entre agentes, con el fin de prevenir ataques en contra de los agentes. Esta propuesta la introdujo Hohl en [30] y se centra en detectar modificaciones en la ejecución mediante la comparación de un estado de referencia con el estado de un agente tras la ejecución en un host. Esta propuesta presenta una serie de inconvenientes, como son la carga de ejecución de los hosts que se ve incrementada en gran medida, ya que la verificación radica en otra ejecución completa del agente en el siguiente host de confianza.

- Traza de ejecución. es un mecanismo que permite la detección de cualquier posible mala conducta por parte de una plataforma, es decir, modificación no apropiada de código, estado o flujo de ejecución del agente móvil. Esta técnica se basa en trazas criptográficas que son recolectadas durante la ejecución del agente en diferentes plataformas. Estas trazas no son más que historiales de las acciones realizadas por un agente móvil durante su ciclo de vida. Esto permite al propietario del agente comprobar su historial de ejecución y ver si contiene alguna modificación hecha sin autorización por alguna plataforma maliciosa. La técnica de trazas de ejecución introducida por Giovani Vigna [31] asume que todas las partes involucradas poseen un par de claves (privada y pública) que pueden ser usadas para producir firmas

digitales, con el objetivo de identificar las partes involucradas. Esto se realiza en la ejecución del agente, que va tomando trazas de las instrucciones que alteran el estado del mismo debido a la entrada de datos externos. Habitualmente el tamaño de estas trazas es considerablemente grande para ser enviados de forma constante a la agencia origen, por lo cual permanecen en la agencia actual cierto tiempo. Finalmente, en vez de enviar las trazas a la agencia origen, se envía un hash de la misma, con idea de evitar un posible ataque de repudio. El proceso es el siguiente, si la agencia origen sospecha de que algo va mal en un host, le demanda al mismo las trazas de la ejecución, junto con los datos de entrada que contienen y se ejecuta al agente de nuevo. El chequeo es bastante sencillo si ambas ejecuciones coinciden el host se considerará honesto, de lo contrario no. Entre las ventajas de esta propuesta nos encontramos el hecho de que se proporcionan pruebas del comportamiento malicioso de los hosts. Aunque por supuesto, también presenta una serie de inconvenientes. El tamaño considerable de las trazas hace totalmente desaconsejable el envío de las mismas a la agencia origen desde todas las agencias dentro del plan del agente en sí. Una solución parcial es que la agencia origen sea la encargada de solicitar las trazas en caso de sospecha. Lo cual nos hace plantearnos cómo se puede tener indicios de que un host ha actuado de forma maliciosa. Además del inconveniente de la falta de mecanismos automáticos para la implantación de esta técnica.

Enfocadas a la Prevención de los Ataques:

- Código ofuscado. La ofuscación es una técnica en la que el productor de código móvil hace cumplir la política de seguridad aplicando una transformación al código que conserva su comportamiento, antes de ser enviado a ejecutarse a diferentes máquinas, que por otra parte son confiables en distintos niveles. La ofuscación permite proteger el código analizado y procesado por el host. Como consecuencia, el host no podría modificar el comportamiento del código móvil o ver información crítica que esté oculta dentro del código, como puede ser una clave secreta, un número de tarjeta de crédito, etc. Esta técnica es usada en muchos ámbitos de aplicación, en concreto para la protección de los agentes se utiliza una variante de la misma basada en blackbox con limitación de tiempo [32].

- Ejecución de Funciones Cifradas. Tradicionalmente, para asegurar la confidencialidad y la integridad de los datos se han usado funciones criptográficas [33], en esta técnica las agencias ejecutan directamente el código cifrado, sin tener la posibilidad de extraer información del mismo y por tanto no pueden modificar los mismos a su favor. Sin embargo, esto lleva consigo la dificultad de encontrar las funciones que puedan ejecutarse de forma cifrada. En [34] se introduce un homomorfismo con capacidad para realizar funciones aritméticas cifradas de forma remota. Aunque esto tiene sus limitaciones, puesto que las funciones aritméticas no cubren todas las necesidades de un agente móvil. Con lo cual sólo resuelven el problema de forma parcial.

- Modelos Basados en la Confianza. Esta propuesta de Ordille [35] consiste en limitar la ejecución únicamente a aquellas máquinas confiables. Vamos a considerar que una

máquina es confiable siempre y cuando de ella no se espere ningún tipo de actividad anómala o maliciosa. Sin embargo, esta propuesta presenta una serie de inconvenientes. La principal de estas radica en el número de agencias que podemos considerar que son confiables en sistemas abiertos como Internet. Otro de los inconvenientes es cómo identificar qué agencias son confiables y cuales no, es decir, podríamos establecer unas listas de reputaciones que nos indiquen si una agencia es confiable o no, pero pensemos en el caso de una agencia que hasta el momento se ha comportado de forma confiable, pero en cierto punto esta, por cualquier tipo de razón, decide sacar partido de la ejecución del agente. Y por supuesto, esta solución no proporciona medidas para ello. Existen mejoras para el esquema anterior, basados en la idea de utilizar modelos de delegación de confianza [36, 37], que proporciona la inferencia de las relaciones de confianza de ciertos elementos a partir de otras relaciones de confianza existentes.

- Entropía de Agentes Móviles. Se define el concepto de entropía [38] de un agente móvil como la relación entre las intenciones que tiene un agente de realizar una tarea y la probabilidad de que el evento en sí suceda. Partiendo de este conocimiento, podemos presuponer que efectivamente las agencias maliciosas van a atacar con mayor frecuencia a ciertos tipos de agentes y bajo ciertas condiciones. La estrategia que se esconde tras el uso de esta técnica radica en ocultar las intenciones del agente en ejecución. Para ello se han diseñado dos técnicas de índoles diferentes. En primer lugar nos encontramos con el concepto de esparcimiento de intenciones "intention spreading" basado en disminuir la entropía del agente mediante la inserción de nuevas tareas no demandadas por el usuario, dificultando el pronóstico del atacante sobre las intenciones reales del agente. Esta técnica está de alguna forma ligada a la técnica de ofuscación de código explicada en 2.4.1. En segundo lugar nos encontramos con la reducción de intenciones "intention shrinking" consistente en la división de las tareas a realizar por un agente en diferentes agentes cooperativos, manteniendo estos un nivel alto de entropía. Sin embargo, la aplicación de estas técnicas presenta severos inconvenientes. La primera de las estrategias no evita que la agencia maliciosa pueda sacar provecho de todas las tareas realizadas por el agente en ejecución, además de mostrar un alto consumo de recursos de forma innecesaria. Respecto a la segunda propuesta, esta comparte las limitaciones de la aplicación de la técnica de agentes cooperativos detallado en 2.2.2.

- Uso de Agentes Cooperativos. Roth [25] presenta su propuesta de protección mutua de agentes cooperantes, partiendo de la base de que en un sistema abierto como Internet es difícil que varias agencias estén confabuladas para actuar en contra de un agente, puesto que es complicado que entre ellas haya una relación de confianza. Esta propuesta radica en la compartición de secretos y decisiones entre varios agentes que actúan en cooperación, con la consideración de que una parte individual del secreto no proporciona información sobre el total. Además el almacenamiento de resultados confidenciales y la toma de decisiones se realizan en el agente cooperativo, que además viaja por una ruta disjunta. Al igual que ocurre con otras técnicas comentadas anteriormente, esta presenta varias restricciones en su uso. En primer lugar, es importante resaltar el hecho de que el rendimiento del sistema se puede ver

severamente dañado, puesto que dónde antes se enviaba un agente ahora se envían varios. Otra de las desventajas de esta propuesta radica en el hecho de que todos los agentes cooperantes han de permanecer en la red hasta que el último finalice. Otro aspecto a considerar consiste en que el sistema ha de garantizar que la comunicación entre los agentes cooperantes se mantenga, puesto que sino esto podría provocar retrasos, e incluso la obstrucción a la consecución de las tareas principales. Y por supuesto, qué ocurre si alguno de los agentes cooperantes abandona su ejecución por cualquier motivo ajeno, como podría darse el caso de que la máquina en la que esté corriendo la agencia sea apagada por cualquier motivo. Y por último, no debemos olvidar que esta estrategia asume que no hay confabulación, sin embargo no podemos obviar este hecho. A pesar de las desventajas que puede presentar esta solución, es cierto que aporta ideas muy ventajosas para el desarrollo de nuevos mecanismos. En concreto una de las propuestas desarrolladas en este trabajo de tesis tiene como semilla la propuesta de los agentes cooperantes, sin embargo llevada a otros niveles, solventando la mayoría de las limitaciones de esta propuesta, para más detalle consultar capítulo 4.

- Generación de claves de entorno. Esta propuesta se describe en [39] y se basa en que el agente lleva su código cifrado y hace uso de una clave que se encuentra en su entorno para descifrar el mismo. El proceso es el siguiente, el agente permanece en un estado de inactividad hasta que el host deposite la clave de entorno en un lugar predeterminado, ya sea local o remoto, y en tal momento se procederá a descifrar el código. Sin embargo, este esquema presenta una serie de inconvenientes. Entre ellos destacamos el malgasto de recursos, puesto que es obligatorio monitorizar el entorno en busca de la clave de forma continua. Pero el más relevante, sin duda alguna, consiste en que el agente estará protegido frente a posibles ataques de la agencia mientras este permanezca cifrado, pero desde el momento en que el agente quede en claro estará a merced de la agencia. A pesar de los inconvenientes presentados por esta propuesta, puede resultar de interés si se proporciona algún tipo de mecanismo para solucionar los inconvenientes. Sin duda, es muy buena iniciativa de partida.

- Uso de Hardware específico. En la literatura nos encontramos algunas iniciativas que pretenden resolver el problema planteado de las agencias (o hosts) maliciosas mediante el uso de hardware específico [40, 41]. La principal ventaja de este tipo de propuestas consiste en que ni tan si quiera el dueño del dispositivo puede manipular la ejecución del agente sin que este quede inutilizado. Entre las diferentes propuestas dentro de este conjunto nos encontramos algunas que se basan en el uso de tarjetas inteligentes [42] o en TPM como se describe en el capítulo 3, e incluso otros autores proponen ideas muy similares como es el caso de Yang [43], cuya idea básica consiste en delegar la ejecución del agente a una tercera parte confiable. En concreto es un hardware específico para tal propósito, y del cual se tiene "certeza" de que no realice ninguna acción maliciosa. Por supuesto, este tipo de soluciones tienen ciertas restricciones. Sin embargo, una de las soluciones desarrolladas en el marco de este trabajo de tesis está dentro de este tipo de soluciones.

2.4.2. Protección de la máquina contra otros agentes

De forma similar a lo que ocurre con los agentes sucede con las agencias que albergan la ejecución de los agentes. Al producirse transferencia de código ejecutable a través de la red existe la posibilidad de que el código tenga malas intenciones, o incluso que este contenga algún agujero de seguridad que, aunque no sea malintencionado, puede provocar que otras partes, que por su parte lo sean, lo aprovechen. Al igual que los agentes pueden ser objeto de ataques por parte del host donde se ejecutan, el agente puede ser el que sea malicioso e intente corromper la máquina en la que se encuentra ejecutandose para obtener beneficio de ello o simplemente por alterar el funcionamiento correcto del sistema. Algunos de los ataques que los agentes pueden ejecutar contra la agencia pueden ser; obtener información leyendo ficheros alojados en la máquina o leyendo directamente la memoria RAM, la denegación de servicios a otros procesos de la máquina, un mal uso de los recursos disponibles de la máquina, causar molestias a la máquina (jugar con los dispositivos externos, manejar la GUI,...), la modificación del funcionamiento de la máquina para que colabore en sus acciones, tales como sabotear a otros agentes, etc. A diferencia del problema del host malicioso, este problema ha sido abordado obteniendo mejores resultados. A continuación vamos a mostrar los más relevantes.

Algunas soluciones propuestas

Sandboxing

SandBoxing [1] es una técnica software usada para proteger a una plataforma de agentes frente a agentes móviles. En un entorno de ejecución (plataforma), el código local es ejecutado con todos los permisos y tiene acceso a recursos críticos del sistema. Por otro lado, el código remoto es ejecutado dentro de un área restringida llamada "sandbox". La restricción afecta a ciertas instrucciones de código, como son el acceso a las propiedades del sistema local y la invocación de programas en la máquina local. Esto asegura que un agente móvil de carácter malicioso no pueda causar ningún daño al entorno de ejecución dónde se está ejecutando. El Sandbox hace cumplir una política de seguridad fuerte para la ejecución de código remoto. La política especifica las reglas y restricciones que el código del agente móvil debería cumplir. La implementación más común de Sandboxing está en el intérprete de Java, dentro de los navegadores web con java. Un intérprete de Java contiene tres componentes de seguridad principales: ClassLoader, Verifier, y Security Manager. El ClassLoader convierte código remoto en datos estructurados que pueden ser añadidos a la jerarquía local de clases. Además, cada clase remota tiene asociada un subtipo de la clase ClassLoader.

Antes de que el código remoto se cargue, el Verifier realiza una serie de comprobaciones de seguridad, con el objetivo de garantizar que sólo se ejecute el código legítimo de Java. El código remoto debería ser un código válido para la máquina virtual, y no ha de desbordar la pila ni usar los registros de forma inadecuada. Adicionalmente, las clases remotas no pueden sobrescribir los nombres locales y sus operaciones son comprobadas por el Security Manager antes de la ejecución. En el JDK de Java, las clases son etiquetadas como locales y remotas. Las clases locales realizan sus operaciones sin ninguna restricción mientras las clases remotas deberían someterse a un proceso de comprobación que implementa la

política de seguridad de la plataforma. Esto implementa dentro del Security Manager. Si una clase remota pasa la verificación, entonces le serán garantizados ciertos privilegios de acceso a los recursos del sistema y continúa ejecutando su código. Además, se lanzará una excepción de seguridad.

Un problema con la técnica Sandboxing es que un fallo en cualquiera de los tres componentes de seguridad mencionados puede provocar un fallo del sistema. Por tanto, se puede ver incrementado el tiempo de ejecución de código remoto legítimo, pero esto se puede solucionar haciendo uso de una combinación de las técnicas de firma de código con el Sandboxing.

Firma de código

La técnica de firma de código asegura la integridad del código descargado desde Internet. Permite a la plataforma verificar que el código no ha sido modificado desde que fue firmado por su creador. La firma de código no puede revelar lo que el código puede hacer o garantizar que es seguro en su ejecución.

La firma de código hace uso de una firma digital y una función hash. Una implementación conocida de ésta técnica es el "Microsoft Authenticode", el cual es típicamente usado para firmar código como controladores ActiveX y applets de Java. La firma de código permite verificar la identidad del creador del código, pero esto no garantiza que sea seguro.

Combinación de firma de código y sandboxing

El JDK de Java combina las ventajas de ambas técnicas. Si el receptor del código confía en el firmante del código, entonces éste se ejecutará como si fuera código local, es decir, con todos los privilegios. Por otra parte, si el receptor del código no confía en el firmante, entonces el código se ejecutará dentro del Sandbox. La principal ventaja de esto es que permite la ejecución de código de entidades no confiables.

2.5. Tecnologías de base para la solución

En esta sección vamos a describir las tecnologías que hemos utilizado como base en la implementación de nuestras soluciones. En primer lugar describimos la tecnología de trusted computing, por ser esencial para nuestra primera solución. En segundo lugar, describimos la computación protegida por ser el germen de nuestra solución de protección de agentes basada en software. Por último terminamos dando una descripción de la técnica conocida como código portador de prueba y otra de tarjetas inteligentes por ser complementarias a ambas soluciones.

2.5.1. Uso de hardware confiable: trusted computing group

Se conoce como hardware confiable a todo dispositivo, ya sea un microcontrolador, un coprocesador o una tarjeta inteligente, que ofrece una serie de facilidades que permiten manejar de manera segura información crítica. Este hardware permite realizar operaciones

con datos de manera que un posible atacante no pueda tener acceso a ningún tipo de información sobre dichos datos. Un concepto básico del hardware confiable es que es resistente a intentos de modificación del propio dispositivo, lo que se conoce como tamper proof (a prueba de falsificación).

Existen diferentes maneras de protegerse ante los ataques físicos:

- Tamper evidence, que consiste en que el dispositivo posee mecanismos que provocan que cualquier ataque físico deje huella y por lo tanto pueda ser detectado.
- Tamper resistance, el dispositivo posee características que hacen difícil atacarlo físicamente.
- Tamper detection, en este el dispositivo posee características que permiten detectar cuándo se está realizando un ataque (por ejemplo con el uso de sensores).
- Y por último el tamper response, en este el dispositivo posee mecanismos que destruyen la información cuando se produce un ataque, lo que evita el acceso a la misma.

Hay que destacar que la seguridad que proporciona el hardware confiable tiene un límite físico determinado por sus características, sin embargo, la comunicación entre los dispositivos seguros y los no seguros siempre es susceptible de ser atacada.

Existe una gran diversidad de ataques físicos diferentes que podría sufrir el hardware confiable. En primer lugar, el acceso directo a componentes internos del dispositivo. En segundo lugar, lo que se conoce como "micro probing", que consiste en observar o manipular el dispositivo. O las técnicas conocidas como "ingeniería inversa", que analizan el dispositivo para identificar sus componentes y cómo se relacionan. También nos encontramos con las técnicas de lectura de memoria (mediante congelado por nitrógeno líquido, que aumenta el tiempo que la RAM retiene datos y lectura posterior de los datos). Y con otro tipo de técnicas no invasivas (acceso con rayos UV, rayos x, etc.).

Además de ataques directos sobre el dispositivo, el hardware confiable suele ser resistente a otros tipos de ataques no invasivos. Los conocidos como "tempest attacks", que consisten en reconstruir datos a partir de las radiaciones electromagnéticas del dispositivo, los conocidos como "side channel attacks" que pueden llevarse a cabo de diferentes maneras:

- Mediante un análisis temporal, que consiste en medir el tiempo que consumen las operaciones criptográficas respecto a los datos de entrada y el algoritmo implementado.
- O mediante un análisis de potencia, basado en medir las fluctuaciones de consumo de corriente cuando el dispositivo realiza determinadas operaciones.
- Y por último también se puede realizar mediante un análisis de fallos, consistente en provocar fallos en el sistema y analizar cómo reacciona el mismo.

Por otro lado, también es posible atacar los dispositivos de hardware confiable realizando ataques software de las API criptográficas. La realización de este tipo de ataques

es posible gracias a ciertos problemas existentes en dichas API. En primer lugar la integridad de las claves es insuficiente. Ya que se pueden producir problemas derivados de la retrocompatibilidad, que implican hacer uso de algoritmos más vulnerables, e incluso otros tipos de ataques a la integridad de las claves. En segundo lugar, se puede realizar un chequeo insuficiente de los parámetros de la función (ataques sobre el sistema de recuperación de códigos PIN, etc.). O se puede dar el caso de que ocurra un cumplimiento insuficientemente estricto de las políticas de seguridad.

Hay que destacar que el hardware confiable no es completamente seguro, y que siempre es posible realizar algún tipo de ataque. Sin embargo, el objetivo de este hardware es dificultar los ataques, de manera que se produzcan en menor medida. Por otro lado, también es importante resaltar que gran parte de los ataques que se producen tienen como objetivo el software que a su vez hace uso del hardware confiable. Por lo que es importante realizar el diseño del software con la mayor cautela posible, ya que de nada sirve disponer del hardware de seguridad más avanzado del mercado si no se hace un buen uso específico del mismo. Una vez descrita nuestra concepción de hardware confiable, vamos a ver el hardware que usamos en esta tesis, el Trusted Platform Module (TPM). El TPM es un microcontrolador desarrollado por el Trusted Computing Group (TCG) que almacena claves criptográficas, claves de acceso y certificados digitales protegiéndolos de ataques externos, tanto físico como lógicos (software).

Antes de analizar con más detalle el TPM, vamos a conocer un poco más qué es el TCG y cual es su objetivo. El TCG es una organización sin ánimo de lucro formada en 2003 por diversas empresas (AMD, HP, Microsoft, Sony, Sun Microsystems, etc.) con el objetivo de desarrollar, definir y promover estándares abiertos para computación segura basada en hardware y otros desarrollos tecnológicos relacionados con la seguridad. Las especificaciones del TCG tienen como objetivo obtener entornos de desarrollo más seguros sin que ello comprometa la integridad funcional, privacidad o derechos individuales. éstas especificaciones adoptan las creadas por el TCPA (Trusted Computing Platform Alliance) promoviendo de manera conjunta la mejora de dichas especificaciones y su extensión hacia diversas plataformas como servidores, PDA, teléfonos digitales, etc.

La estructura del TCG está diseñada para favorecer una amplia participación, una gestión eficiente y una amplia adopción de las especificaciones. Para ello se siguen una serie de principios. El primero es que la pertenencia al TCG está abierta a un amplio rango de organizaciones, además de empresas que apoyan los objetivos del TCG. Existen diferentes niveles de pertenencia al TCG y los beneficios están claramente definidos. El segundo principio se basa en la política de licencia de patentes entre miembros que se realiza de forma razonable y no discriminatoria. El tercer principio radica en la estructura de los comités, grupos de trabajo, etc. se deciden por mayoría absoluta. Y por último está el plan de mercado definido.

Actualmente, el TCG cuenta con más de 170 miembros entre los que figuran empresas como Microsoft, Sun Microsystems, AMD, HP, IBM, Infineon, Intel, Lenovo, etc. El TCG cuenta además con un consejo asesor, designado por la junta directiva, cuyo objetivo es aconsejar sobre un amplio espectro de temas relacionados con la computación confiable. Este consejo está compuesto por expertos de una diversa variedad de campos como la privacidad, seguridad, arquitectura de computadores, servicios financieros, comercio electrónico, etc. El TCG se compone de varios grupos de trabajo encargados de desarrollar

diferentes estándares y aplicaciones. Existe un grupo concreto para la especificación del TPM cuya arquitectura proviene de la *computación confiable* y el grupo de trabajo de TPM define la implementación de dicha arquitectura. Los miembros del grupo de trabajo son especialistas con conocimientos de seguridad relativos al uso de módulos criptográficos y técnicas criptográficas de clave pública, así como sus algoritmos y protocolos.

El módulo de plataforma confiable (trusted platform module *TPM)*

El TPM es un microcontrolador que almacena claves criptográficas, contraseñas y certificados digitales. Normalmente se trata de un chip integrado en la placa base de un PC, aunque puede ser usado en cualquier dispositivo que requiera las funciones que este proporciona. La estructura del TPM asegura que la información que contiene sea más segura frente a ataques externos tanto software como físicos. El TPM permite que puedan realizarse diferentes procesos seguros como firma digital, intercambio de claves, etc. El acceso a los datos contenidos en el TPM puede impedirse en el caso de que la secuencia de arranque no sea la correcta. Estas características permiten realizar acciones y ejecutar aplicaciones críticas, como correo electrónico seguro, acceso Web seguro y protección local de datos, de manera mucho más segura. Todas estas características del TPM pueden ser integradas también en otros componentes del sistema.

El TPM proporciona una serie de funcionalidades tales como funciones de clave asimétricas para la generación de pares de claves usando el generador hardware de números aleatorios. Así como también proporciona funciones de firma mediante clave privada y cifrado-descifrado de algoritmos de clave pública, lo cual permite almacenar datos de manera segura. Las claves privadas creadas en el TPM están protegidas incluso cuando están en uso. También proporciona almacenamiento seguro de valores hash representativos de la configuración en los PCRs (Platform Control Registers) y comunicación segura de dichos valores mediante autorización del propietario del TPM. Esto permite realizar una atestación de la configuración basada en la cadena de confianza usada para crear los valores hash. Para ello se usan claves AIK (Attestation Identity Keys) que sólo pueden usarse en el caso de que el valor del PCR sea el mismo que cuando la clave fue creada. La clave EK (Endorsement Key) permite al propietario del TPM comprobar que las claves de identidad fueron efectivamente generadas en un TPM, mediante la confirmación de la calidad de la clave, sin necesidad de identificar qué módulo en concreto generó dicha clave. Y por último, mencionar que también se proporciona la funcionalidad necesaria para la inicialización y gestión del propio TPM, esto permite a su propietario desactivarlo, resetearlo y cambiar el propietario.

La especificación del TCG

A continuación vamos a analizar la especificación realizada por el TCG de sistemas basados en hardware confiable. Para empezar, un concepto importante es el de las plataformas confiables. Una plataforma confiable es aquella que nos proporciona un entorno seguro para la ejecución de software.

Consideremos la figura 2.4, que representaría una plataforma confiable compuesta por un PC con un TPM. Esta plataforma debe proporcionar una serie de funcionalidades básicas para ser considerada confiable. En primer lugar tenemos las funciones protegidas.

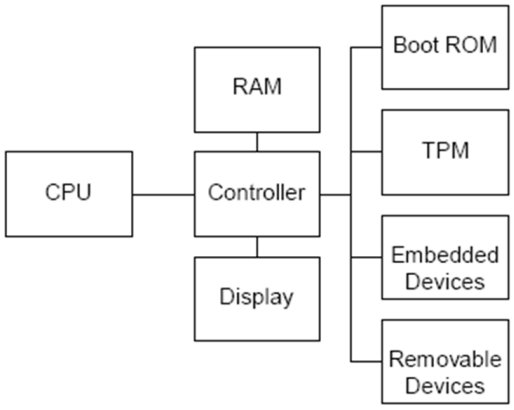

Figura 2.4: Componentes del TPM.

Estas son una serie de comandos que tienen permiso exclusivo para acceder a ubicaciones protegidas. Con ubicaciones protegidas nos referimos a aquellas zonas (memoria, registros) dónde pueden realizarse operaciones sobre datos sensibles de manera segura. El TPM implementa estas funciones protegidas que permiten realizar informes de mediciones de integridad de la configuración, además de proporcionar ubicaciones seguras para proteger dichas mediciones. Por otro lado, el TPM almacena claves criptográficas que permiten autenticar la información. También disponemos de la atestación, consistente en un proceso que permite a una entidad obtener datos precisos de otra. La atestación puede verse desde varios puntos de vista. La atestación realizada por el TPM es una operación que proporciona veracidad a datos contenidos en el TPM. Se realiza firmando digitalmente datos internos del TPM usando una AIK. La veracidad, tanto de los datos firmados como de la AIK la determina la entidad que los solicita. La AIK se obtiene mediante protocolos que hacen uso de una Autoridad de Certificación (Certification Authority, CA) o protocolos de atestación segura. La atestación de la confiabilidad de una plataforma es una operación que asegura que puede confiarse en una plataforma para que esta proporcione medidas de la integridad. Se realiza usando un conjunto (o subconjunto) de las credenciales asociadas con la plataforma, que permiten crear las credenciales de la AIK. La atestación de la configuración de una plataforma es una operación que proporciona veracidad a un conjunto de datos sobre la integridad de la misma. Se realiza firmando digitalmente mediante una AIK un conjunto de registros PCR. La autenticación de la plataforma, es una operación que asegura la autenticidad de la identidad de una plataforma, la cual puede o no estar ligada a un usuario o a las acciones llevadas a cabo por este. Esta operación se realiza firmando con una clave "no migrable" del TPM. Debe tenerse en cuenta que el número de claves de este tipo en un TPM es ilimitado, siendo por tanto ilimitado el número de identidades. También tenemos soporte para llevar a cabo la medida, registro e informe de integridad. La medida de la integridad es el proceso que obtiene características del sistema que puedan afectar a la integridad (confiabilidad) del mismo. El punto de partida de esta medida se conoce como Raíz de confianza (Root of Trust, ROT), que proporciona

un estado inicial confiable del sistema. Informar a cerca de de la integridad es el proceso de atestación de los datos contenidos en los PCRs. Por último tenemos el registro, que consiste en mantener constancia de las diferentes medidas de la integridad realizadas para un uso posterior. La filosofía de estos tres procesos consiste en que, aunque permiten que la plataforma alcance una configuración no deseada o no confiable, impiden que ésta pueda proporcionar información falsa sobre su estado.

Las raíces de confianza son componentes de la plataforma confiable en las que debe confiarse, ya que es imposible detectar cuando alguno de ellos actúa de manera no segura. Un conjunto completo de raíces de confianza debe proporcionar al menos la funcionalidad necesaria para describir las características de la plataforma que afectan a la confiabilidad de la misma.

En una plataforma confiable existen comúnmente tres raíces de confianza. Una se encarga de las medidas (Roots of Trust for Measurement, RTM). El RTM es un motor de computación capaz de realizar medidas de la integridad confiables. Normalmente se trata del propio motor de cómputo de la plataforma, controlado por el núcleo del RTM (Core Root of Trust for Measurement, CRTM), que son las instrucciones ejecutadas por la plataforma cuando actúa como RTM. El RTM es también la raíz de la cadena de confianza transitiva, que veremos más adelante. Otra raíz de confianza para almacenamiento (Roots of Trust for Storage, RTS): que es un motor de cómputo capaz de mantener una relación de valores precisos sobre la integridad de la plataforma. Y por último una para el informe(Roots of Trust for Report, RTR): que es un motor de cómputo capaz de informar de forma fiable sobre los valores contenidos en el RTS. Asumimos que el funcionamiento de cada raíz de confianza se considera fiable y por lo tanto no es necesario supervisarlos.

Las raíces de confianza poseen partes que carecen de ubicaciones seguras y funciones protegidas. Estas partes se conocen como Trusted Building Blocks (TBB) y normalmente se corresponden con las instrucciones para el RTM y para la inicialización del TPM. Normalmente los TBB son específicos de cada plataforma. Un ejemplo de TBB es la combinación del CRTM, la conexión del almacenamiento del CRTM con la placa base, la conexión del TPM a la placa base y el mecanismo para determinar la presencia física como podemos apreciar en la figura 2.5. El TBB es confiable, desde el punto de vista del comportamiento no afectará a las metas de la plataforma segura.

La combinación de los TBB y las raíces de confianza forman un límite de confianza dentro del cual la medición almacenamiento y reporte de los datos pueden llevarse a cabo para una configuración mínima. Sistemas más complejos pueden requerir que las mediciones de integridad las realice otro código ROM además del CRTM. Para ello, debe establecerse la confianza con el código ROM, analizando dicho código antes de transferirle el control de la ejecución. Los TBB deben establecerse de manera que los dispositivos de medición no amplíen el límite del TBB más allá del límite de confianza.

Se conoce como confianza transitiva o inductiva al proceso por el cual la ROT establece una descripción confiable de un segundo grupo de funciones. Basándose en esta descripción, una entidad interesada puede determinar la fiabilidad de este segundo grupo de funciones. En el caso de determinar si el grupo es confiable, el límite de confianza se extiende a la raíz de confianza hasta incluir el grupo. De esta manera, puede irse extendiendo de manera iterativa el límite de confianza abarcando nuevos grupos de funciones. En la figura 2.6 podemos observar cómo el sistema va ampliando el límite de confianza

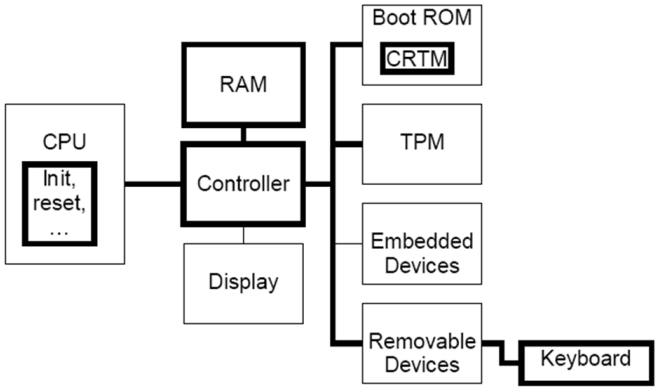

Figura 2.5: Ejemplo de TBB (Resaltado en negrita).

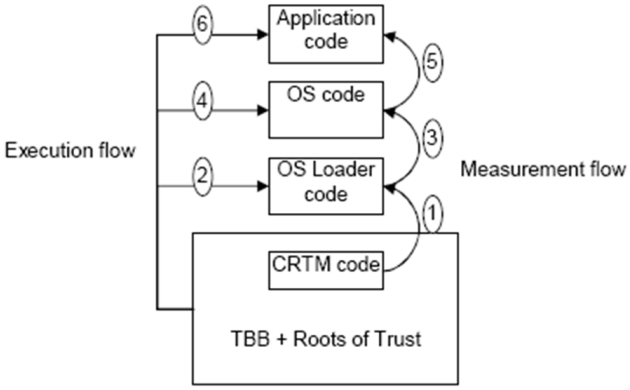

Figura 2.6: Esquema que muestra cómo se expande el límite de confianza.

desde la raíz de confianza, incorporando sucesivamente diferentes bloques que pasan a ser confiables.

Además el TPM proporciona un sistema de medición de la integridad que genera informes del estado operativo de la plataforma, almacenando dos tipos de medidas. En primer lugar los parámetros medidos se refieren a lo que se mide en realidad. Y los valores hash, que no son más que un valor obtenido tras aplicar una función hash a la medición de cierto parámetro. Los valores hash son almacenados en el TPM usando la funcionalidad de RTR y RTS, mientras que los parámetros medidos pueden ser almacenados en cualquier lugar que el sistema de medición determine. De hecho pueden incluso no almacenarse, siendo computados cuando sean necesarios.

Los datos medidos describen propiedades y características del componente medido. Es responsabilidad del implementador de núcleo de medición entender la sintaxis y la semántica de los campos medidos con el suficiente detalle como para poder producir

resultados que sean usados por los consumidores de los eventos de medida.

El registro de medidas almacenadas (Stored Measurement Log, SML) contiene secuencias de medidas relacionadas. Dichas secuencias comparten un mismo valor hash, ya que cada nueva medida se enlaza a las anteriores y se recalcula de nuevo el valor hash. Esta técnica permite relacionar valores y preservar la secuencia de las operaciones. La representación algebraica de la extensión de los valores se calcula como sigue:

PCR[n]<-SHA-1(PCR[n]+datos de medición)

EL SML no se almacena dentro del TPM ya que puede ser demasiado grande y no necesita la protección proporcionada por el TPM ya que los ataques contra el SML podrían detectarse. Aun así, el SML puede sufrir ataques de denegación del servicio, por lo que los implementadores deberían poder replicar o regenerar el registro.

El RTR tiene dos funciones: proporcionar ubicaciones seguras para el almacenamiento de informes de integridad y atestar la autenticidad de los valores almacenados según la identidad de las plataformas seguras. Los PCRs pueden implementarse en memoria volátil o no volátil y deben estar protegidos frente a ataques software. También deben tenerse en cuenta medidas para protegerlos frente a ataques físicos. Los informes de integridad se firman digitalmente usando una AIK para autenticar los valores de los PCRs. La firma incluye un *nonce* para evitar ataques de repetición.

El TPM posee una clave interna llamada Endorsement Key (EK). Esta clave se usa para establecer el propietario del TPM y para emitir las credenciales de las AIK. El propietario del TPM puede crear una clave raíz de almacenamiento (Storage Root Key, SRK), que sirve para envolver otras claves del TPM pudiendo almacenarlas de manera segura fuera del TPM. El reporte de la seguridad puede usarse para determinar la configuración de una plataforma en un momento dado.

El protocolo consta de varias etapas:

1. Una entidad solicita uno o más valores de los PCRs de la plataforma.

2. Un agente de la plataforma con acceso al TPM obtiene las entradas del SML.

3. El agente de la plataforma recibe los valores del TPM.

4. El TPM firma los valores de los PCRs usando una AIK.

5. El agente de la plataforma obtiene las credenciales correspondientes al TPM. Los valores de los PCRs firmados, las entradas del SML y las credenciales son entregados a la entidad que los solicitó.

6. La entidad solicitante verifica los datos obtenidos. Los valores recibidos son comparados con los valores que posee el solicitante. Además se evalúan las credenciales y se comprueba la firma.

Este protocolo es independiente del mecanismo de transporte o entrega de los datos. Un objetivo de la atestación es permitir que una entidad determine si un TPM ha firmado un mensaje e incluso determinar qué TPM lo ha hecho. Para ello puede ser necesaria la

participación de una CA que emita las credenciales de la AIK que atestiguan la confiabilidad de la plataforma. Estas credenciales sustituirían a la parte pública de la EK, que no es apropiada para su distribución pública. El TPM debe probar a la CA que la AIK es de uso exclusivo suyo. Para ello, la CA emite las credenciales cifrándolas con la EK, de manera que sólo el TPM que generó dicha TPM pueda descifrarla. Por su parte, la CA no debe revelar información privada relacionada con el TPM.

- Credenciales del TCG: El TCG define cinco tipos de credenciales, cada tipo está pensado para proporcionar la información necesaria para llevar a cabo una operación específica: credenciales de aprobación *(Endorsement Credentials)*, credenciales de conformidad *(Conformance Credentials)*, credenciales de plataforma *(Platform Credentials)*, credenciales de validación *(Validation Credentials)* y por último credenciales de identidad *(Attestation Identity Credentials)*.

- Credenciales de aprobación: Son emitidas por la entidad que generó la EK, que normalmente es el fabricante del TPM. Las credenciales contienen el nombre del fabricante del TPM, el número de modelo del TPM, la version del TPM, la clave pública de la EK y por último la firma del fabricante del TPM. La clave pública de la EK, aun siendo pública, debe ser de acceso restringido como señalamos anteriormente, debido a que identifica de manera única el TPM y por extensión a la plataforma.

- Credenciales de conformidad: Son emitidas por cualquier entidad con la credibilidad suficiente como para evaluar un TPM o una plataforma que lo contenga. La evaluación puede ser llevada a cabo por el fabricante de la plataforma, su distribuidor o por una entidad independiente. Estas credenciales indican que el evaluador acepta el diseño del TBB y su implementación de acuerdo con unas directrices de evaluación establecidas. Cuando el evaluador firma las credenciales avala los resultados de la evaluación. Pueden emitirse diferentes credenciales de conformidad para una misma plataforma, una para el TPM y otras para los diferentes componentes del TBB.

El contenido de las credenciales incluye el nombre de la entidad evaluadora, el nombre del fabricante de la plataforma, el número de modelo de la plataforma, la versión de la plataforma (si es aplicable), el nombre del fabricante del TPM, el número de modelo del TPM y por último la versión del TPM. Es importante reflejar el hecho de que las credenciales de conformidad no contienen información que identifique de manera única a la plataforma.

- Credenciales de plataforma. Son emitidas por el fabricante de la plataforma, su distribuidor o por cualquier entidad con la credibilidad suficiente. Sirven para identificar al fabricante de la plataforma y para describir las propiedades de la misma. También contiene valores hash de las credenciales de aprobación y de conformidad. Estas credenciales contienen información privada, ya que están asociadas con una plataforma específica. Las credenciales contienen la siguiente información: el nombre del fabricante de la plataforma, el número de modelo de la plataforma, la versión de la plataforma (si es aplicable), la credencial de aprobación y la credencial de

conformidad (o credenciales si son varias). Estas credenciales proporcionan evidencia de que la plataforma contiene un chip TPM identificado por la credencial de aprobación.

- Credenciales de validación. Son emitidas por una entidad de validación, normalmente los fabricantes de los componentes de la plataforma. Contienen valores hash de referencia representativos del estado de los componentes tomados en el momento de su fabricación y que indican el estado justo en el que el mismo funciona de manera adecuada. Las credenciales de las componentes deben contener el nombre de la entidad de validación, el nombre del fabricante, el número de modelo, la versión, los valores de referencia medidos y por último las funcionalidades del mismo. Las credenciales deben ser publicadas y distribuidas de manera que puedan ser usadas para comprobar el funcionamiento correcto de los componentes.

- Credenciales de Identidad. Identifican la clave privada de la AIK usada para firmar valores PCR. Son emitidas por un servicio confiable que verifica las credenciales y preserva las políticas de privacidad de los clientes. El emisor de las credenciales prueba que el TPM es propietario de la AIK y que éste está ligado a unas credenciales de aprobación, plataforma y conformidad válidas. El emisor también garantiza que cumplirá las expectativas de privacidad de los clientes, como pueden ser la protección de la información, que identifica al cliente, usada en el proceso de generación de las credenciales. Las credenciales de identidad contienen la identificación del fabricante del TPM, el número de modelo del TPM, así como el nombre del fabricante del mismo, el número de modelo de la plataforma y por último la ubicación de la documentación de conformidad de la plataforma.

El TPM como elemento de seguridad en el intercambio de información

En el intercambio de mensajes basado en clave asimétrica, los mensajes destinados a un único individuo se cifran con la clave pública de dicho individuo para que sólo él pueda leerlo. Por otro lado, podemos proteger el mensaje frente a falsificaciones firmando con la clave privada del emisor. Sin embargo, una mala gestión de las claves puede derivar en una pérdida de la seguridad. El TPM proporciona seguridad en la comunicación, ya que permite la gestión de claves y configuraciones. Estas características junto con la capacidad de ligar el uso de claves a un estado determinado de la configuración del sistema permite que la comunicación se realice de manera segura.

En lo que respecta al intercambio de mensajes protegidos, vamos a hacer una breve descripción de las cuatro clases de intercambio de mensajes protegidos que define el TCG; el ligado (Binding), firmado (Signing), sellado-ligado (Sealed-Binding, Sealing) y sellado-firmado (Sealed-Signing).

- Enlazado (Binding): Se trata de la operación clásica de cifrado del mensaje a transmitir mediante el uso de una clave pública. El emisor usa la clave pública del receptor para cifrar el mensaje, de manera que el contenido del mismo sólo sea accesible si se está en posesión de la clave privada correspondiente. Si la clave privada se encuentra en un TPM como clave "no migrable", sólo el TPM que creó la clave puede usarla

y por lo tanto el mensaje sólo será accesible desde dicho TPM. Por otro lado, si la clave fuera migrable esta característica desaparecería. Este mecanismo es clave para una de las aportaciones fundamentales de esta tesis como se verá en capítulos sucesivos.

- Firmado (Signing): Se trata de la operación clásica de firma, que asocia la integridad de un mensaje con la clave usada para generar la firma. El TPM puede etiquetar ciertas claves para que sólo puedan ser usadas para firmar, de manera que no puedan ser malinterpretadas y consideradas como claves de cifrado.

- Sellado-Ligado (Sealed-Binding): Los mensajes a los que se les aplica esta operación quedan ligados a un estado de la plataforma, de esta manera el mensaje sólo puede ser descifrado si la plataforma del receptor se encuentra en un estado concreto. Esta operación asocia un mensaje con una serie de valores de los PCRs y con una clave asimétrica "no migrable". De esta manera el receptor sólo puede descifrar el mensaje si el TPM posee la clave de cifrado correspondiente y los valores de sus PCR son los determinados por el emisor.

- Sellado-Firmado (Sealed-Signing): Esta operación enlaza una firma con unos valores determinados de los PCRs, añadiendo por lo tanto la prueba de que la plataforma que firmó el mensaje se encontraba en un estado determinado. En este caso, el receptor de la firma indica al emisor que la firma debe incluir una serie de valores de los PCRs. El emisor, obtiene los valores de los PCRs y los incluye en el mensaje a firmar. El verificador analiza posteriormente los valores de los PCRs incluidos en la firma, de manera que puede determinar el estado de la configuración de la plataforma emisora en el momento en el que se realizó la firma.

El RTS protege claves y datos confiados al TPM. Para ello el RTS gestiona una pequeña porción de memoria volátil en la cual se cargan las claves para realizar operaciones de firma y cifrado. Las claves no usadas pueden ser cifradas y almacenadas fuera del TPM para obtener espacio para otras claves que se vayan a usar. La gestión del espacio de caché para claves es gestionado de forma externa por el Gestor de Caché de Claves (Key Cache Manager, KCM), que actúa de interfaz entre el dispositivo de almacenamiento donde se almacenan las claves inactivas de manera indefinida.

El RTS está optimizado para almacenar objetos pequeños, aunque puede almacenar diferentes tipos de objetos como claves simétricas, claves asimétricas, claves de acceso, cookies, resultados de autenticación y datos en general.

Existen dos tipos de claves integradas en el TPM, la EK y la SRK, que no pueden ser eliminadas, aunque es posible crear una nueva clave SRK para un nuevo propietario de la plataforma. Esto último provoca que los datos controlados por la SRK anterior no queden "en-claro". La clave SRK generada por el TPM y su clave de acceso se cifran usando la EK cuando se establece el propietario del TPM. La clave SRK se usa para proteger otras claves de manera que puedan ser almacenadas en el exterior del TPM.

Atributos de las claves. Todas la claves gestionadas por el RTS pueden ser "migrables" o "no-migrables", según se pudieran transferir de un TPM a otro. Este atributo se establece

en el momento de creación de la clave y no puede cambiarse. Semánticamente, una clave "no migrable" está permanentemente ligada con el TPM que la creó. La migración de una clave no migrable permitiría que una plataforma se pudiera hacer pasar por otra perpetrando un ataque de falsificación o suplantación de identidad. Una clave AIK es un ejemplo claro de clave que nunca debería ser migrable, de ahí que en estas el atributo esté fijado como "no migrable".

Las claves "migrables" pueden intercambiarse entre diferentes TPM, permitiendo que un par de claves pueda ser usado por un usuario en diferentes dispositivos. Esto permite que pueda realizarse una comunicación segura entre dos individuos aunque estos cambien de plataformas.

Los atributos de las claves no pueden ser aplicados a datos genéricos, por lo que para extender la propiedad de ser "no migrable" a unos datos determinados debemos cifrarlos con una clave "no migrable" y almacenarlos mediante el RTS. De esta manera, los datos sólo pueden ser descifrados usando las funcionalidades del TPM y por lo tanto sólo donde esta se encuentre. Hay que tener en cuenta, que una vez descifrados los datos, éstos pueden ser migrados a cualquier parte.

El TCG define siete tipos de claves, cada una de las cuales tiene sus propias restricciones de uso. A grandes rasgos las claves podrían clasificarse en claves de firma o de almacenamiento. Una clasificación más precisa distinguiría entre la siguientes claves:

- Claves de firma (Signing keys): Son claves asimétricas de propósito general que se usan para firmar datos de una aplicación y mensajes. Pueden ser "migrables" o "no migrables", pudiendo en cada caso ser exportadas a otros TPM o no. El TPM puede firmar datos de la aplicación con una clave e imponer restricciones para la migración de dichos datos.

- Claves de almacenamiento (Storage keys): Son claves asimétricas de propósito general usadas para cifrar datos u otras claves. Estas claves se usan normalmente para envolver otras claves o datos para puedan almacenarse externamente.

- Claves de identidad (Identity keys, AIK): Son claves de firma "no migrables" cuyo propósito exclusivo es firmar datos generados por el TPM (valores PCR, etc.).

- Claves de conformidad (Endorsement keys, EK): Son claves de cifrado "no migrables" pertenecientes a una plataforma. Se usan para descifrar datos del propietario en el momento en el que se establece el mismo en la plataforma. También se usan para descifrar mensajes asociados con la creación de una AIK. No deben usarse para cifrar o firmar.

- Claves de ligado (Bind keys): Son claves que pueden ser usadas para cifrar datos pequeños (por ejemplo, claves simétricas) en una plataforma y descifrarlos en otra.

- Claves externas (Legacy keys): Son claves creadas fuera del TPM. Se importan al TPM para firmar o cifrar. Por definición son claves "migrables".

- Claves de autenticación (Authentication keys): Son claves simétricas usadas para proteger sesiones de transporte relacionados con un TPM.

El TPM dispone de un espacio reducido de almacenamiento, sin embargo, existen escenarios de uso que implican la necesidad de un espacio ilimitado para almacenar datos. Por este motivo es necesario disponer de espacio de almacenamiento externo al TPM y de un gestor de caché el key caché management (KCM).

- Almacenamiento externo, para disponer de un espacio de almacenamiento ilimitado para claves, el RTS empaqueta las claves destinadas a almacenamiento externo y de forma cifrada. De esta manera se construyen bloques de datos que pueden almacenarse fuera del TPM en cualquier dispositivo de almacenamiento disponible. Estos bloques de datos están ligados a un TPM en particular y pueden estarlo también a una configuración determinada de la plataforma. Para identificar estos bloques pueden usarse manejadores o el valor hash de su contenido que permitan identificar a la aplicación encargada de gestionar dichos bloques.

- El gestor de caché de claves (KCM). El TPM dispone de interfaces que permiten que aplicaciones externas puedan gestionar los recursos. El KCM se encarga de gestionar las claves. Las claves asociadas a una configuración determinada de la plataforma pueden cargarse en memoria, aunque no se cumplan los requisitos de configuración. Sin embargo, este conjunto de requisitos sí deberá satisfacerse para que pueda hacerse uso de la clave.

En definitiva el KCM funciona como un programa externo al TPM encargado de gestionar los movimientos de claves entre la memoria del TPM y los dispositivos de almacenamiento externo. El KCM gestiona los espacios para claves disponibles en el TPM y se encarga de determinar cuándo es apropiado reemplazar una clave por otra.

Sin embargo, el TPM no proporciona información de cuándo el espacio para claves está agotado o cuándo una aplicación particular necesita usar una clave. Esto implica que la aplicación debe informar al KCM de dichos eventos. También es posible que el KCM implemente un interfaz de comunicación con el TPM a través del cual las aplicaciones obtengan los servicios que ésta presta. Sin embargo, esta última opción tiene asociados ciertos riesgos desde el punto de vista de la seguridad, ya que implica que el KCM tiene conocimiento de las claves de acceso, por lo que lo convierte en un claro objetivo de ataques. El KCM proporciona interfaces que permiten preparar la transición de las claves desde el TPM hacia los dispositivos de almacenamiento externo. Estas claves no deben ser jamás usadas sin cifrar. Por otro lado, el KCM debe implementar un modelo de indexación, acceso y almacenaje de las claves cifradas almacenadas en dispositivos externos. El KCM debe también gestionar los pasos necesarios para usar las claves en el TPM.

A continuación vamos a analizar el diseño lógico del TPM y sus componentes. El diseño del TPM permite que pueda desarrollarse desde un punto de vista hardware o software.

En la figura 2.7 podemos observar los diferentes bloques que componen un TPM con soporte para funcionalidad RTR y RTS. El funcionamiento de las piezas que componen el TPM se considera fiable sin necesidad de supervisión. Esta confianza en los componentes se deriva de buenas prácticas de ingeniería, un buen proceso de fabricación y de la revisión de la industria correspondiente.

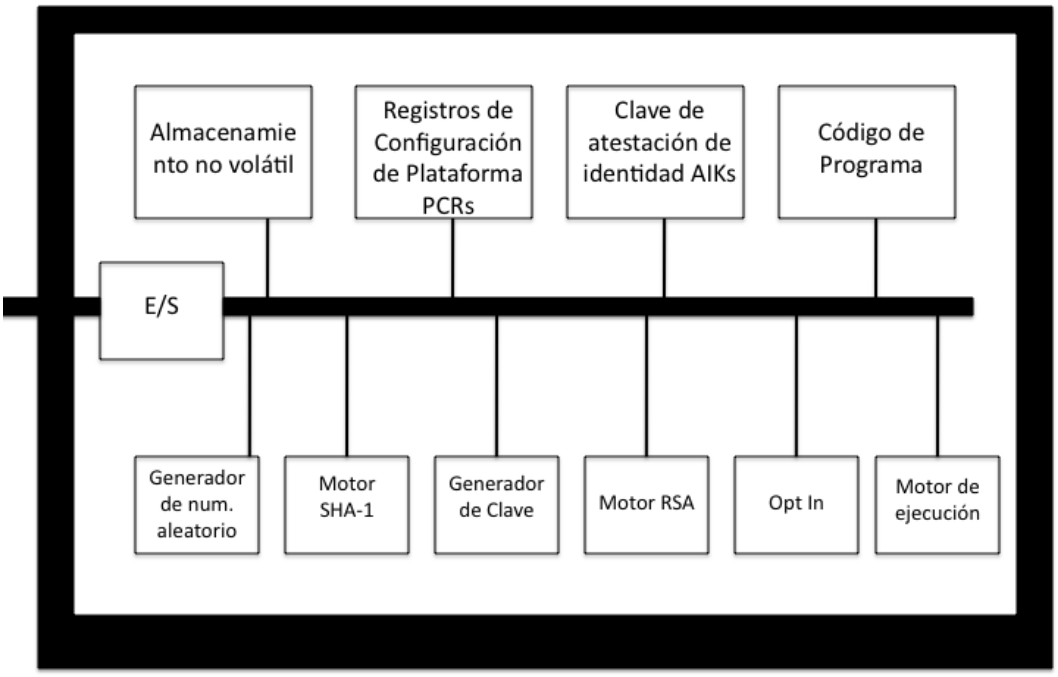

Figura 2.7: Bloques que componen un TPM.

Los componentes discretos de un TPM son:

- Entrada/Salida (E/S). Los componentes de E/S gestionan el flujo de la información a través del bus de comunicación. Estos componentes llevan a cabo protocolos de codificación/decodificación válidos para la comunicación sobre buses externos e internos. También se encargan de enviar mensajes a los componentes apropiados imponiendo políticas de acceso, además de realizar otras funciones del TPM que requieren control de acceso.

- Almacenamiento no volátil. Este tipo de almacenamiento se usa para almacenar la EK, la SRK, las claves del propietario y otros indicadores persistentes. Los PCRs pueden implementarse tanto en memoria volátil como en memoria no volátil, ya que se resetean siempre que el sistema se inicia o la plataforma deja de recibir corriente. El TCG impone la existencia de un mínimo de 16 registros, de los cuales, del 0 al 7 están reservados para el uso del TPM, mientras que del 8 al 15 están disponibles para uso del sistema operativo y las aplicaciones.

- Atestation Identity Keys (AIK). Las AIK deben ser persistentes, sin embargo se recomienda que sean almacenadas cifradas de forma externa al TPM. Cuanto más espacio disponible haya en el TPM para mantener las AIK cargadas mayor será la velocidad de ejecución de los programas que hagan uso de ellas.

- Código de programa. El código de programa contiene un firmware encargado de

medir la integridad de los dispositivos que forman la plataforma. Esto es lo que se conoce como CRTM, e idealmente debe encontrarse en el interior del TPM, sin embargo en ocasiones puede localizarse externamente.

- Generador de número aleatorios (Random Number Generator, RNG). El TPM contiene un generador de bits aleatorios real que se usa para generar números aleatorios. El RNG se usa para generar claves, valores *nonce* y para reforzar la entropía de las claves de acceso.

- Motor Sha-1. Es un motor que calcula valores hash Sha-1 y que se usa para generar firmas, empaquetar claves cifradas y para otros usos generales.

- Generador de claves RSA. El TCG establece el algoritmo RSA como estándar para el uso en TPM. El motor de generación de claves RSA se usa para crear claves de firma y de almacenamiento. Los TPM deben soportar claves RSA de más de 2048 bits, y algunas claves (SRK, AIK) deben ser de al menos 2048 bits.

- Motor RSA. El motor RSA se usa para firmar con claves de firma, cifar/descifrar con claves de almacenamiento y descifrar con la EK.

- Opt-In. Estos componentes implementan la política del TCG que obliga que los TPM sean entregados a los clientes en el estado que ellos deseen. El estado inicial puede ser deshabilitado, desactivado o habilitado, preparado para que el usuario establezca el propietario del TPM. Los mecanismos Opt-In proporcionan la lógica y los interfaces para determinar el estado de la presencia física y asegurar que se deshabiliten las operaciones de otros componentes del TPM cuando sea necesario.

- Motor de ejecución. Se encarga de ejecutar el código de programa. Realiza la inicialización del TPM y las mediciones necesarias.

El TCG no especifica ningún interfaz de comunicación ni arquitectura de buses, ya que se considera que dependen de decisiones de implementación. El TCG define un interfaz de serialización que puede transportarse a través de prácticamente cualquier tipo de bus o interconexión.

El TCG exige que el TPM esté protegido ante ataques físicos. Esto incluye que el módulo esté ligado a otros componentes de la plataforma (por ejemplo la placa base) de manera que no pueda ser fácilmente desensamblado y añadido a otra plataforma. El TPM debe ser resistente a la falsificación proporcionando mecanismos que permitan detectar ataques físicos. Las implementaciones software de los TPM deben incluir componentes que sean equivalentes a esos mecanismos de detección además de asegurar que sólo pueda ligarse un único TPM a una plataforma.

2.5.2. Técnica de código portador de prueba "proof-carrying-code"

Lee y Necula [44] introdujeron la técnica del "código portador de prueba" (PCC). Esta técnica requiere que el propietario del código proporcione una prueba formal de que el mismo cumple con la política de seguridad del receptor. Al mecanismo PCC, se le

considera "autocertificado" porque no se requiere ninguna parte criptográfica o confiable adicional. Esto implica un bajo coste de chequeo después de lo cual el programa puede ser ejecutado sin una comprobación que requiera un alto coste computacional. Además, cualquier modificación al código puede ser detectada. Estas ventajas hacen que esta técnica sea muy útil no sólo para agentes móviles, sino también para otros usos como redes activas y sistemas operativos extensibles.

La técnica del código portador de prueba también tiene algunas limitaciones, que tienen que ser tratadas antes de que pueda usarse de forma directa. El problema principal con PCC es la generación de la prueba, especialmente por la dificultad inherente en el proceso de generación de la misma. Actualmente existen muchos grupos de investigación trabajando en automatizar el proceso de generación de la prueba.

PCC posee varias características que proporcionan ventajas sobre otros mecanismos para la ejecución segura de código remoto:

1. La infraestructura del consumidor es automática y de bajo riesgo.
2. El esfuerzo reside en el productor: el consumidor de código solamente tiene que ejecutar un proceso de verificación de pruebas rápido y simple.
3. Esta técnica no requiere relaciones productor/consumidor confiables.
4. PCC es flexible: puede utilizarse con un amplio rango de lenguajes y de políticas de seguridad.
5. Técnica muy versátil que no solo se puede usar para seguridad.
6. La generación de PCC puede ser "automatizada" en cierta medida.
7. El código PCC se verifica estáticamente.
8. Los programas PCC permiten detectar cualquier modificación (accidental o maliciosa) del programa y/o su prueba, mientras siguen garantizando la seguridad.
9. Además puede usarse en combinación con otras técnicas.

La idea de PCC es fácil de comprender, pero su implementación eficiente presenta serias dificultades y su uso algunas desventajas. A continuación se enumeran las principales desventajas que surgen del uso de PCC:

1. La arquitectura PCC es muy sensible a los cambios de las políticas de seguridad, lo cual puede ser beneficioso o todo lo contrario en función del caso en concreto.
2. Dado que el PCC es un proceso cooperativo, el productor de código debe estar involucrado en la definición de la seguridad del consumidor de código.
3. La codificaciones triviales de pruebas de propiedades de programas son muy largas.
4. La generación de predicados acerca de propiedades del código es una tarea dificultosa.

5. Probar la condición de verificación es una tarea complicada y costosa.

6. Verificar la prueba no es una tarea fácil si lo que se quiere es una prueba concisa y que el chequeador sea pequeño, rápido e independiente de la política de seguridad.

7. Establecer la política de seguridad es una tarea costosa.

8. Garantizar la corrección de la arquitectura es un procedimiento muy costoso.

Comparando la lista de beneficios con la de dificultades, queda clara evidencia que la estrategia de diseño de PCC consiste en que la carga de la seguridad debe estar en el productor y no en el consumidor de código. Esto habilita al PCC a trabajar con una infraestructura pequeña, confiable y automática, por lo que el "trabajo difícil" es realizado por el productor de código que, por normal general, se halla en una mejor posición para entender el código o para usar herramientas interactivas para este propósito.

2.6. Tarjetas inteligentes

El uso de las tarjetas inteligentes está cada día más extendido, de hecho, ya es habitual verlo en manos de muchos usuarios en su vida diaría. Este tipo de dispositivos representa un avance cualitativo en el camino hacia la seguridad de la información práctica. Esto se refleja en el hecho de que hasta su aparición la producción de firmas digitales y otras primitivas criptográficas se encontraba limitada por la necesidad de disponer de un entorno de computación seguro y fiable. Entendiendo por estos conceptos algo bastante restringido. Sin embargo, a la hora de llevarlo a la práctica esto implica la imposibilidad de usar estas técnicas en entornos de computación con un alto grado de movilidad.

La aparición de las tarjetas inteligentes representa un avance cualitativo en esta faceta solucionando el problema, al ser dispositivos seguros, resistentes a ataques físicos y portátiles, con la capacidad de almacenar información relevante y realizar la computación necesaria para la producción de firmas digitales y otras primitivas criptográficas.

El concepto de tarjeta inteligente se ha definido de formas muy diversas, entre las que encontramos "tarjeta dotada de un circuito integrado con memoria y capaz de tomar decisiones" o "dispositivo computacional portátil resistente a ataques con capacidad de almacenamiento de datos". Un significado frecuentemente asociado al término es el de "tarjeta dotada de un circuito integrado con interfaz ISO7816". En cualquier caso ha de quedar claro que quedan fuera de esta categoría las tarjetas magnéticas (con una capacidad de almacenamiento reducida a 300 bytes y sin capacidad de cómputo), las tarjetas de memoria (con capacidad de almacenamiento superiores llegando a varios Gigabytes) y las tarjetas PCMCIA (con capacidad de proceso y memoria elevadas al igual que su coste de fabricación).

Entre las diferentes características de las tarjetas inteligentes, existen varias significativas para diferenciarlas entre sí. En primer lugar tenemos la capacidad de memoria de almacenamiento, entre las que encontramos desde 32 Kb hasta de varios Megabytes. Otro aspecto a considerar es el medio de conexión. Las primeras tarjetas inteligentes tenían conexión cableada, sin embargo cada vez más se van viendo las que presentan conexiones inalámbricas. Otro factor a considerar en las tarjetas es el de la velocidad del procesador.

La frecuencia del procesador suele estar relacionada con el lector. Puesto que las tarjetas no disponen de generador de reloj interno y por tanto reciben la señal de reloj del lector. Esto hace que, aunque existan tarjetas capaces de funcionar a frecuencias muy altas de reloj (33Mhz), en la práctica se ven limitadas a la velocidad que es capaz de proporcionar cada lector y que suele rondar los 5Mhz. Evidentemente, los avances en la tecnología de control de reloj hacen que algunas tarjetas puedan funcionar internamente a estas elevadas velocidades de reloj, independientemente de la velocidad proporcionada por el lector. Otro aspecto a considerar es el referente al software contenido en la tarjeta, tanto a nivel de sistema operativo como de aplicaciones, así como la posibilidad de instalar aplicaciones de forma dinámica, que determinan en gran medida la utilidad de las mismas. Y por último la inclusión de coprocesadores que aceleren las operaciones criptográficas puede ser una característica importante, debido a que la utilidad de estos dispositivos está muy ligada a la seguridad de la información.

2.6.1. El lenguaje javacard

El lenguaje *javacard* consiste en un conjunto de instrucciones, a su vez, es un subconjunto de Java, por lo tanto todas las construcciones del lenguaje *javacard* existen en Java y se comportan de la misma manera. Este hecho queda demostrado en que como parte de un ciclo estándar de desarrollo, un applet *javacard* se compila en un archivo de clase Java (.class) por un compilador Java normal, sin ningún tipo de opción especial (aunque el fichero compilado será procesado posteriormente por herramientas específicas para la plataforma *javacard*). No obstante, muchas características del lenguaje Java no son compatibles con *javacard* (en particular algunos tipos básicos (char, double', float y long); los enums; los arrays de más de una dimensión; los hilos (threads);... y algunas características son opcionales y están ausentes en la mayoría de las tarjetas inteligentes (pe. el tipo int, en particular, que es el valor por defecto de un tipo de expresión de Java, la recolección de basura). El bytecode de *javacard* gestionado por la máquina virtual de *javacard* es un subconjunto funcional de Java (Java 2 Standard Edition) aunque utiliza una codificación diferente optimizada para ocupar menos espacio.

Las librerías estándares de *javacard* difieren de las de Java, y el subconjunto común es mínimo. Por ejemplo, la clase del Java Security Manager no es compatible con *javacard*, donde las políticas de seguridad son ejecutadas por la máquina virtual de la tarjeta. Las técnicas de codificación utilizadas en la programación de *javacard*s difieren significativamente de las que se utilizan en un programa Java. Aún así, como *javacard* utiliza un subconjunto del lenguaje Java se acelera la curva de aprendizaje, y permite utilizar un entorno Java para desarrollar y depurar un programa *javacard*.

Podemos afirmar que *javacard* es una tecnología que permite ejecutar de forma segura pequeñas aplicaciones Java (applets) en tarjetas inteligentes y similares dispositivos empotrados. Este lenguaje ofrece al usuario la capacidad de programar aplicaciones que se ejecutan en la tarjeta de modo que ésta tenga una funcionalidad práctica en un dominio de aplicación específico (pe. identificación, pago, etc.). Esta tecnología tiene un uso bastante extendido en las tarjetas SIM (utilizadas en teléfonos móviles GSM) y en tarjetas monedero electrónico. La primera tarjeta Java fue presentada en 1997 por varias empresas entre las que estaban Axalto (división de tarjetas de Schlumberger) y Gemplus. Los

productos *javacard* están basados en la Plataforma *javacard* cuyas especificaciones han sido desarrolladas por Sun Microsystems. La última versión de la plataforma *javacard* es la especificación 2.1.1, liberada por Sun en el 2000.

Entre las principales características de esta tecnología figuran la portabilidad y la seguridad. Este lenguaje tiene por objeto la definición de un estándar de tarjeta inteligente que permite al mismo applet funcionar en diferentes tarjetas inteligentes, de forma muy parecida a cómo un applet de Java se ejecuta en diferentes ordenadores. Al igual que en Java, esto se consigue utilizando la combinación de una máquina virtual (la Máquina virtual de *javacard*), y unas librerías cuya API está completamenteespecificada. La portabilidad, en todo caso, sigue siendo obviada en muchos casos por cuestiones de tamaño de la memoria, en rendimiento y tiempo de ejecución (por ejemplo para los protocolos de comunicación o algoritmos criptográficos). La tecnología *javacard* fue desarrollada originalmente con el propósito de asegurar la información sensible almacenada en las tarjetas inteligentes. La seguridad está determinada por diversos aspectos de esta tecnología:

- Applet. El applet se ofrece como una máquina de estados que sólo procesa los comandos recibidos a través del dispositivo lector enviando y respondiendo con códigos de estado y datos.

- Separación de applets. Las distintas aplicaciones están, separadas unas de otras por un cortafuegos que limita el acceso y control de los elementos de datos de un subprograma a otro.

- Encapsulación de datos. Los datos se almacenan en la aplicación y las aplicaciones *javacard* se ejecutan en un entorno aislado (la tarjeta de Java VM), separada del sistema operativo y del equipo en que se lee la tarjeta.

- Criptografía. En esta plataforma están implementados los algoritmos criptográficos más comúnmente utilizados como DES, 3DES, AES, RSA (incluyendo el uso de criptografía de curva elíptica). También se ofrecen otros servicios como la firma electrónica o la generación de claves de intercambio también están soportados.

2.7. Computación protegida

Al igual que sucede en otros ámbitos de la seguridad informática, en el de la protección del software no existen esquemas absolutamente seguros. El objetivo de cualquier esquema de protección de software consiste en dificultar la labor de los usuarios deshonestos tanto como sea posible, de modo de el beneficio obtenido no compense el esfuerzo a realizar para llevar a cabo el ataque.

En el diseño de esta técnica se usa como elemento básico un procesador resistente a ataques físicos [48]. En los primeros pasos de esta técnica se consideró que el uso de las tarjetas inteligentes, en gran medida por su evolución de proceso, hicieron que se considerasen a estos dispositivos como la opción más adecuada. No obstante, con los recientes avances en la computación protegida y la versatilidad de esta técnica, no se condiciona en absoluto el uso de tarjetas inteligentes. De hecho, esta misma técnica se puede aplicar

a cualquier elemento hardware de capacidades similares a las tarjetas inteligentes, tales como ciertos tipos de llaves electrónicas, tokens, e incluso TPMs.

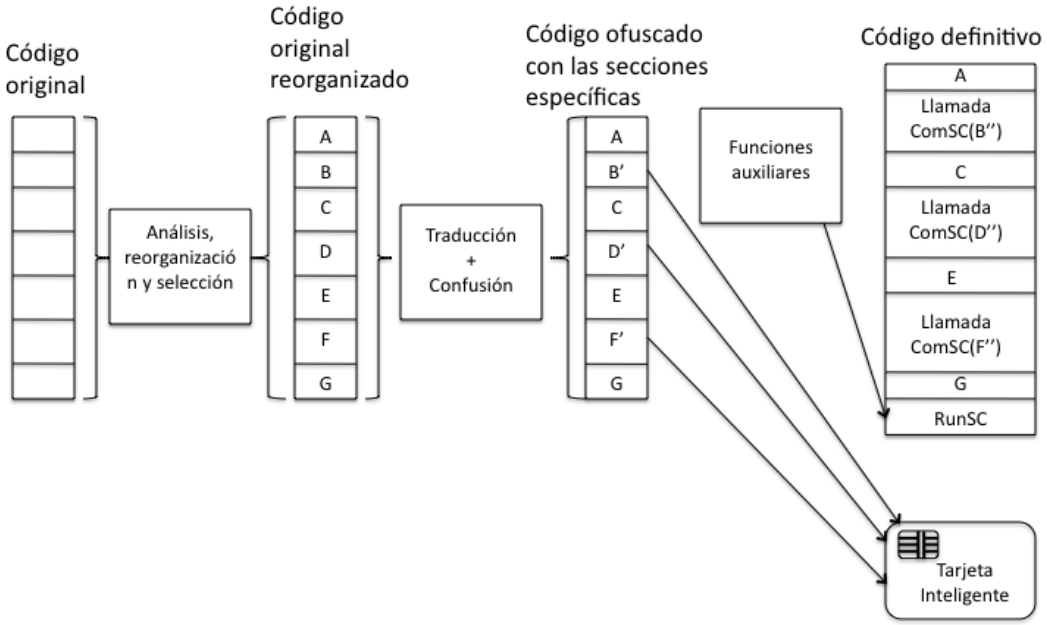

Figura 2.8: Transformación del código en el esquema base.

En un esquema de protección de software que se basara únicamente en el uso de tarjetas inteligentes, como se muestra en la figura 2.8, se podrían sustituir ciertas secciones de código de la aplicación a proteger por otras de funcionalidad equivalente que serían procesadas en la tarjeta. De tal forma que el software funcionaría únicamente cuando se dispusiera de la tarjeta, puesto que se encontraría distribuido entre ésta y el ordenador. En este caso se resuelven ciertos ataques al código, de hecho, excluyendo los ataques físicos, el único ataque posible habría de consistir en el análisis tanto de los datos enviados a la tarjeta como a de las respuestas de la misma, con el objeto de deducir las funciones realizadas en su interior. Aún así, protegiendo un número adecuado de funciones con cierta relevancia y complejidad suficientemente altas, el ataque se puede frustrar en la práctica.

2.8. Conclusiones del capítulo

A lo largo de este capítulo hemos presentado la tecnología de los agentes software. Hemos realizado un estudio de los diferentes tipos de agentes y sistemas de agentes. A continuación hemos presentado todas las alternativas existentes hasta el momento para proporcionar seguridad a los sistemas de agentes, tanto para proteger a los agentes como a los entornos en los que estos se ejecutan A lo largo de este estudio hemos incluido ciertas

reflexiones a cerca de las características de interés de algunos métodos analizados con el objeto de aprovechar su utilidad para la construcción de nuestra solución.

La última parte de este capítulo incluye una serie de tecnologías que han servido como base para la construcción de nuestras dos soluciones.

Capítulo 3

Protección basada en Hardware Criptográfico

3.1. Introducción

A lo largo de la historia más reciente se han propuesto diferentes mecanismos de protección para sistemas de agentes. En el capítulo 2 de esta tesis describimos un conjunto amplio de ellos. Entre los que nos encontramos con un conjunto centrado en proteger a las plataformas frente a agentes, mientras otros buscan proteger a los agentes de posibles ataques de plataformas. Además se muestra la existencia de mecanismos para proporcionar una protección en ambos sentidos, tanto hacia los agentes como hacia las plataformas, que como he mencionado con anterioridad, representan el caso más completo.

En el capítulo 1 introductorio de esta tesis comentamos que el trabajo queda estructurado en dos bloques bien diferenciados. El primero de ellos consiste en el diseño y desarrollo de un mecanismo de seguridad para sistemas de agentes basado en un elemento hardware y el segundo basado en la técnica de computación protegida al que dedicamos el capítulo 4. Para ello se ha elaborado una librería basada en el uso de un dispositivo hardware confiable. Usando este elemento he implementado un mecanismo de migración confiable que aporta la seguridad necesaria al proceso de migración de los agentes de un sistema.

Como vimos en el capítulo 2, un sistema multiagente se compone de una serie de agentes que realizan tareas de manera autónoma o colaborando entre sí. Estos agentes necesitan un entorno en el que ejecutarse, a este entorno lo conocemos como agencia. Una agencia no es más que una ubicación que aloja agentes y les proporcionan los medios para que estos se puedan ejecutar. En la figura B.2 podemos ver un ejemplo del despliegue de un sistema multiagente clásico, en el que se muestran varias agencias distribuidas y conectadas entre sí con un número variable de agentes ejecutándose en las mismas.

En un sistema como el de la figura B.2 los agentes han de disponer de los medios necesarios y accesibles para realizar sus tareas en las agencias, desplazándose de una a otra cuando fuera necesario de acuerdo a su itinerario programado. Sin embargo, como ya he comentado, esta arquitectura implica una serie de problemas relacionados con la seguridad. Esto se pone en práctica debido a que los agentes se ejecutan en el entorno de una agencia remota, estando a merced del comportamiento de la misma. Y por otro

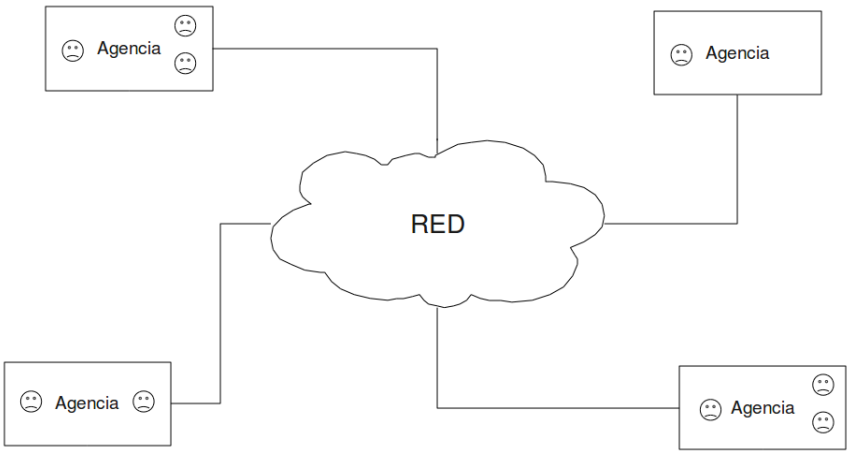

Figura 3.1: Esquema de agencias distribuidas en una red

lado los agentes podrían tener ciertos propósitos maliciosos bien contra otros agentes del sistema o bien frente a las agencias mismas.

La seguridad puede definirse de formas muy variadas, habitualmente se designa un sistema seguro como aquel que cumple algunas de las propiedades de seguridad tradicional como son la confidencialidad, la autenticación, la integridad, el control de acceso, el no repudio y la disponibilidad. Desde nuestro punto de vista estas propiedades deben darse en un cualquier sistema que se denomine seguro y por lo tanto debe ser un objetivo prioritario a perseguir en el momento del diseño de cualquier sistema.

Anteriormente comentábamos que en un sistema multiagente la seguridad puede verse vulnerada por las agencias capaces de atacar a los agentes del sistema. Por esta razón un agente no debe confiar en las agencias obviando que sean seguras. De la misma manera, las agencias también pueden ver vulnerada su seguridad por los agentes que aloja [19]. Entre los posibles problemas de seguridad que pueden darse en un sistema multiagente nos encontramos los siguientes, de una forma esquemática:

Desde el punto de vista de las agencias:

- Los agentes pueden atacar a la agencia de manera que esta pierda su funcionalidad total o de forma parcial.
- Los agentes pueden acceder a datos privados de la agencia para los que no tienen autorización.
- Los agentes pueden engañar a la agencia proporcionando una identidad falsa, haciéndose pasar por otros agentes con privilegios sobre los recursos de la plataforma.

Desde el punto de vista de los agentes:

- Los agentes pueden ser atacados por la agencia en la que se encuentran de manera que dejen de funcionar.
- Las agencias pueden acceder a datos privados de los agentes para los que no tienen autorización.

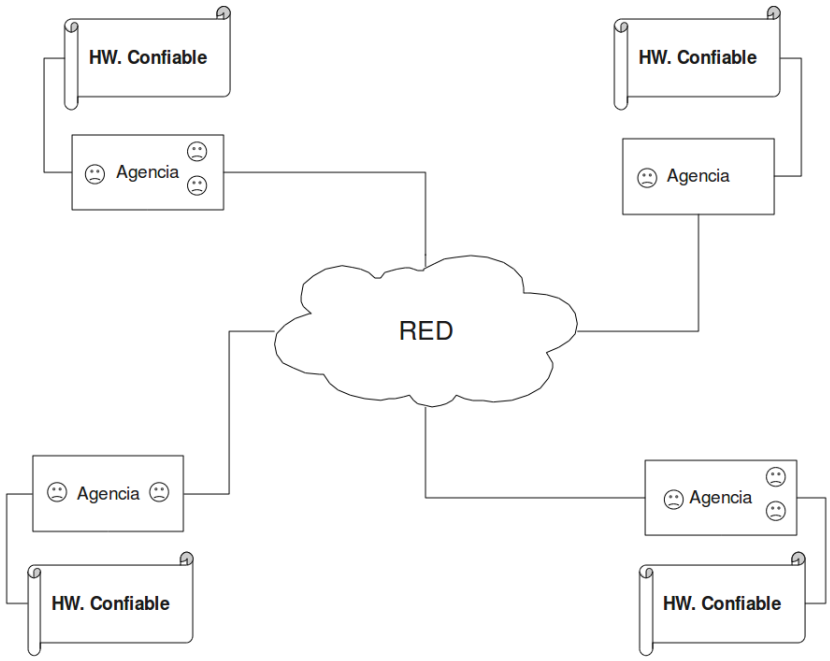

Figura 3.2: Esquema de agencias distribuidas en una red junto a sus respectivos dispositivos hardware confiables.

- Los agentes pueden ser atacados por otros agentes de manera que dejen de funcionar.
- Los agentes pueden acceder a datos privados de otros agentes para los que no tienen autorización.

Para intentar resolver estos problemas realizamos un proceso de investigación de las soluciones que existían así como de las posibles estrategias para la construcción de una solución robusta. En el artículo *Mutual Protection for Multiagent Systems* [49] se presenta una revisión de los trabajos existentes hasta el momento. Además de presentar una primera estrategia de una solución basada en hardware que propone el uso de un dispositivo hardware confiable que proporciona una base sobre la cual construir el sistema multiagente robusto.

En la figura 3.2 vemos un diagrama de un sistema multiagente que hace uso de hardware confiable. Se puede apreciar que cada agencia dispone de un dispositivo hardware para certificar que la agencia efectivamente es confible, de manera que las otras agencias puedan tener la certeza de que dicha certificación proviene únicamente de la agencia en cuestión, y no nos encontremos ante un ataque de suplantación de identidad.

El dispositivo hardware confiable nos proporciona varias funcionalidades. En primer lugar está dotado de una memoria segura para guardar datos sobre la configuración de la plataforma, incluyendo tanto claves de cifrado como cualquier otra información sensible. Además de proporcionar un mecanismo de certificación que permite a la agencia firmar datos de manera que estos queden ligados a la posesión física del dispositivo. La seguridad de un sistema de estas características viene dada por el uso del dispositivo hardware confiable.

Nuestro modelo asume que un agente inicia su recorrido desde una agencia segura, y aunque es posible, y de hecho bastante probable, que el agente requiera migrar a otras agencias para llevar a cabo sus tareas y conseguir sus objetivos. Ello implica que el agente debe asegurarse que cada una de las agencias a las que migra, y por tanto visita, son confiables, previamente antes de realizar la migración en sí. Nuestro mecanismo se basa en proteger el proceso de migración de los agentes móviles. Se puede evidenciar que está especialmente diseñado para este tipo de tecnología, aunque puede ser extendido para otros fines.

En este mecanismo, cada agencia dispone de una serie de valores almacenados que determinan el estado de su configuración, a partir de los cuales se puede realizar una estimación de si el estado de la agencia es confiable o no. La manera en la que cada agencia obtiene y reporta estos valores es mediante el uso de funcionalidades del hardware específico como se detalla a lo largo del presente capítulo.

Este hardware permite tomar ciertas medidas correspondientes al estado del sistema y almacenarlas en registros internos del dispositivo, siendo únicamente accesibles mediante las funcionalidades ofrecidas por el mismo. Los valores medidos son almacenados de manera segura en el interior del propio hardware por lo que no pueden ser accedidos y ni modificados sin autorización. De esta forma las agencias pueden comunicar los valores de configuración previamente almacenados a otras agencias para que estas puedan determinar su grado de seguridad. Siguiendo esta metodología, un agente antes de migrar solicita a la agencia en la que se encuentra que determine la confiabilidad de la agencia destino. Con lo cual un agente que se encuentra en una agencia que considera segura, puede extender el ámbito de confianza incluyendo otra agencia, una vez comprobado que es segura.

Un módulo de plataforma confiable (TPM) se comporta de la forma esperada para un propósito particular [31]. Entre las múltiples funcionalidades que nos ofrece esta tecnología estamos interesados en tres aspectos del Trusted Computing, la atestación remota, la clave encadenada al estado y el aislamiento. La atestación remota es el proceso por el cual una plataforma declara su estado de operativo de entorno/software actual y es almacenado en un conjunto de registros dedicados del hardware confiable llamados "registros de configuración de plataforma" *(Platform Configuration Registers "PCRs")*. La clave encadenada es el proceso mediante el que se pueden usar los datos únicamente si el estado de la plataforma encaja perfectamente con un valor predefinido. Por último el aislamiento se refiere al soporte tanto hardware como software necesario para aislar la ejecución. Recordamos al lector que los tres los tres mecanismos están perfectamente descritos en el capítulo 2.

Los primeros trabajos en la aplicación de hardware confiable como método de protección de sistemas de agentes móviles se le atribuyen a Wilhelm en sus trabajos basados en el uso de terceras partes confiables en los hosts de los sistemas [41]. A pesar de ser una gran idea teórica, en su tiempo parecía demasiado futurista.

3.1.1. Estudios previos

Vamos a analizar el conjunto de requisitos mínimos que tiene que satisfacer nuestra librería de migración. Los requisitos funcionales de la librería deben proporcionar un

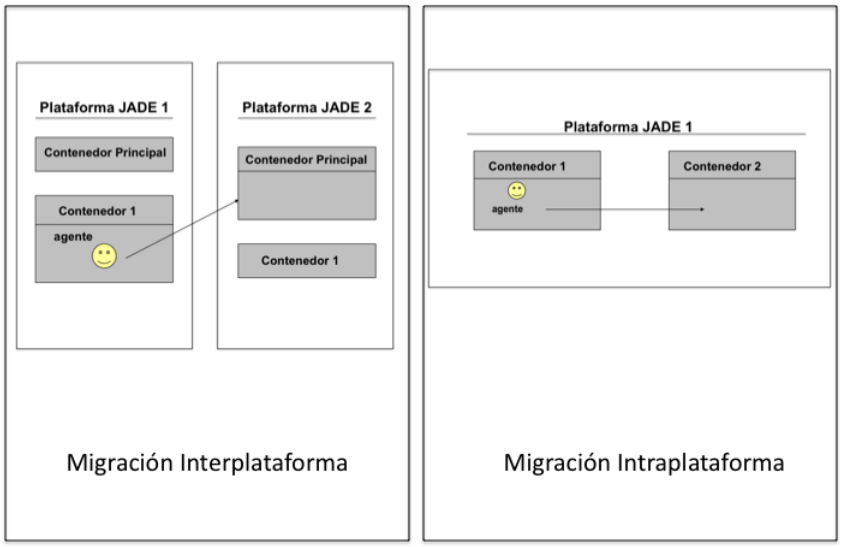

Figura 3.3: Migración inter e intra plataforma.

mecanismo de migración segura de agentes para JADE, de tal manera que el agente pueda extender su límite de confianza añadiendo agencias seguras, tanto dentro de su plataforma como de otras plataformas remotas [50] y [51].

Antes de continuar con este estudio hacemos referencia a ciertos aspectos que asumimos como ciertos. En primer lugar, cada agencia contiene un módulo TPM instalado y funcionando de forma correcta. Además el sistema operativo de esta agencia tiene soporte para el uso de este hardware específico. En el inicio autenticado del sistema, una vez que la máquina en la que está alojada la agencia es puesta en funcionamiento se toman medidas del estado de la misma y se almacena en los PCRs del TPM. Se ha de tener confianza en la plataforma en la que actualmente se está ejecutando el agente, en otras palabras, esta plataforma ha de considerarse confiable.

Cualquier información estática del agente ha de estar firmada digitalmente por la plataforma de origen del agente. La consistencia de los registros viene caracterizada porque el uso de los PCRs para el almacenamiento de las medidas representativas del estado de la plataforma ha de ser consistente con el resto de plataformas del sistema, aunque como veremos más adelante, esta condición puede resultar muy rigurosa. Por último, hemos de considerar que cada plataforma tiene asociada el uso de al menos una de las AIKs con una CA privada que es conocida para todas las agencias confiables del sistema.

Como se describe en el capítulo 2, un sistema JADE está compuesto por una plataforma que contiene un contenedor principal, en el cual se despliegan los agentes. A esta plataforma pueden añadirse contenedores adicionales, que podrían ser remotos y, a su vez, diferentes plataformas tendrán la capacidad de interactuar entres sí, permitiendo así que los agentes transiten de unas a otras. Por lo tanto, en JADE los contenedores jugarían el papel de agencias sobre las que desplegar los agentes.

Para solucionar los problemas de seguridad en sistemas de agentes sobre la plataforma

JADE, hemos desarrollado una librería que permite a los agentes realizar la migración de manera segura. Para ello la librería comprueba si el contenedor destino es seguro y lleva a cabo la migración únicamente si puede confiar en la agencia destino.

También hemos de considerar el hecho de que cada agencia debe ofrecer cierta funcionalidad local que permita a un agente migrar de manera segura a una plataforma destino. Asimismo, cada agencia deberá ofrecer la posibilidad a que cualquier otra agencia del sistema verifique que efectivamente es confiable. Hemos obtenido como resultado una librería que implementa un protocolo que permite a una agencia intercambiar información sobre su configuración con otra agencia, de manera que cada una de las agencias tenga certeza de que la información proviene realmente de la agencia que la proporcionó. La complejidad del problema provoca la aparición de soluciones bastante ineficientes basadas en el uso de elementos software, por lo que nos planteamos el uso de hardware criptográfico específico. Como indicamos anteriormente usamos un TPM, puesto que nos facilita, en gran medida, la tarea de establecer un entorno seguro, permitiendo además almacenar valores de la configuración de las agencias y comunicar de manera segura dichos valores a las agencias que lo soliciten.

Hemos identificado una serie de características deseables de nuestra solución. Como objetivo adicional, que consideramos de relativa importancia, tenemos la posibilidad de la integración de nuestra solución en las plataformas actuales. En concreto dentro del marco de la plataforma JADE, es decir, el mecanismo ha de resultar fácil de integrar en cualquier solución existente basada en JADE. De forma que la utilización de nuestra solución no implique modificaciones drásticas a la misma. Con idea de obtener un uso más extendido de nuestra solución a medio plazo vista.

Otra característica importante de nuestra solución es la transparencia, de forma que el funcionamiento de la librería no debe ser visible al usuario. No debemos de olvidarnos de la genericidad, que es otra característica deseable que proporciona nuestra librería. Como en cualquier solución de seguridad aplicada a sistemas distribuidos, la escalabilidad es imprescindible, máxima si pensamos en el campo de aplicación que tienen los sistemas basados en agentes en los emergentes sistemas de computación ubicuos. Por último, la librería debe ser extensible, es decir, ha de permitir la introducción de futuras de mejoras y ampliaciones de una forma sencilla y eficiente.

El resultado final es una librería que tiene como objetivo dotar de seguridad a los sistemas de agentes móviles desarrollados con JADE, basado en la implementación de un mecanismo de migración seguro. Esta migración segura se consigue mediante la comprobación de la confiabilidad de la agencia destino antes de llevar a cabo la migración, lo que permite que el agente se ejecute siempre en un entorno confiable. Por otra parte, la librería debe integrarse en alguna de las plataformas de agentes que se están usando actualmente.

Entre las distintas opciones nos decantamos por implementarla sobre JADE. Tras un estudio de las alternativas observamos que JADE es la plataforma más utilizada por la comunidad de desarrolladores de agentes. Parece obvio que para que la librería tenga una acogida relevante en la comunidad de agentes es necesario que la utilización de la misma no implique realizar modificaciones drásticas a la plataforma. La librería facilita la adaptación de sistemas existentes para que hagan uso de los mecanismos de seguridad implicando esto modificaciones mínimas. Así nuestra librería proporciona un mecanismo genérico, de

manera que pueda ser fácilmente adaptado a soluciones concretas. El funcionamiento de la librería debe ser transparente al usuario de la misma y la librería debe permitir la introducción de futuras mejoras y ampliaciones de una manera sencilla.

A continuación, vamos a analizar la especificación de la librería, determinando de manera precisa la funcionalidad que presta y mostrando mediante los diagramas correspondientes a la arquitectura de la misma.

En una plataforma JADE los agentes pueden migrar de una agencia otra, sin embargo, esta migración puede darse de dos maneras:

- Desplazamiento (move): El agente se desplaza desde la agencia origen a la agencia destino.
- Clonación (clone): El agente se copia en la agencia destino, de manera que a partir de entonces existirán dos agentes clónicos ejecutándose en ubicaciones diferentes.

De aquí obtenemos que los casos de uso que definen la librería serían los que aparecen en el diagrama de la figura 3.4.

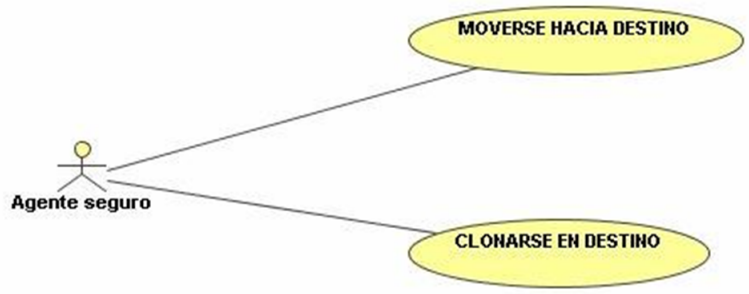

Figura 3.4: Casos de uso de la librería.

En el diagrama se observa que existen dos casos de uso desde el punto de vista del agente que representan la funcionalidad que ofrece la librería. Sin embargo, desde nuestro punto de vista ambos son similares y la forma en que finalmente se realiza la migración no es relevante. Como se ha comentado, vamos a explotar las distintas funcionalidades que nos proporciona el TPM en nuestra librería. Concretamente, nos vamos a centrar en el establecimiento de una raíz de confianza sobre la cual ejecutamos nuestro software.

En la figura 3.5 vemos un diagrama de clases que muestra cómo cada agencia puede alojar varios agentes y dispone de las funcionalidades de un TPM. El TPM juega el papel de elemento seguro dentro de la agencia, permitiendo que esta pueda demostrar a otras agencias que es confiable [19].

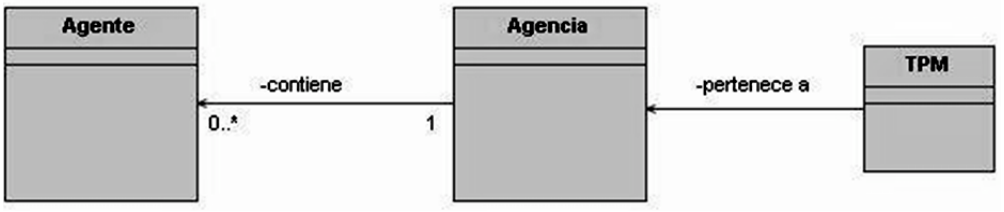

Figura 3.5: La agencia debe tener acceso a la funcionalidad del TPM.

Vamos a ver el camino seguido para obtener nuestro protocolo de migración segura, para ello vamos a analizar con detalle una serie de protocolos que nos permiten la elaboración de un nuevo protocolo que reúna las características apropiadas para que la librería proporcione la seguridad necesaria en el proceso de migración. Como anteriormente comentábamos, a partir de este momento usaremos el término migración para referirnos tanto al desplazamiento como a la clonación de los agentes, ya que desde el punto de vista de los protocolos analizados ambos son equivalentes.

En principio, contemplamos el uso de un dispositivo hardware de seguridad genérico, sin cerrarnos las puertas al uso de uno determinado, para conseguir entornos seguros de ejecución de los agentes móviles, por medio de la protección del propio proceso de migración. Aunque como indicamos anteriormente, para el desarrollo de la solución tuvimos que hacer una elección, siendo el TPM el seleccionado. Analizamos el primer protocolo para obtener la información de la configuración de una plataforma concreta de forma remota.

El protocolo de la figura 3.6 muestra los pasos a realizar para migrar de manera segura a una agencia destino. Estos pasos son los siguientes:

1. El agente ag1 solicita a la agencia A1 que verifique si la migración a la agencia A2 es segura según unos requisitos (A2req).

2. A1 en colaboración con su TPM (tpm1), contacta con el TPM de la plataforma destino (tpm2) y obtiene la configuración de la misma(A2conf).

3. Se analiza la configuración de la agencia A2 para ver si cumple los requisitos establecidos.

4. Si los requisitos se cumplen el agente ag1 migra a la agencia A2.

Se asume que el agente debe confiar en su plataforma, para permitir así la comprobación de la seguridad de la migración. Consideramos interesante mencionar el hecho de que esto coincide con la filosofía de Trusted Computing, en la cual se parte de una raíz de confianza para ir extendiendo el límite de confianza en base a los procesos de atestación. Otra base importante que aparece en este protocolo es el uso de un TPM para que la obtención y comunicación de los valores de configuración se realice de manera segura, ya que las funcionalidades ofertadas por este encajan perfectamente con nuestras necesidades.

Puesto que hasta este punto teníamos claro que el uso de un hardware criptográfico era necesario para implementar este modelo de migración segura, se decidió estudiar a fondo la especificación de Trusted Computing Group. En el TGC Specification Architecture Overview [52], encontramos un documento que muestra una visión general de la arquitectura propuesta por el TCG.

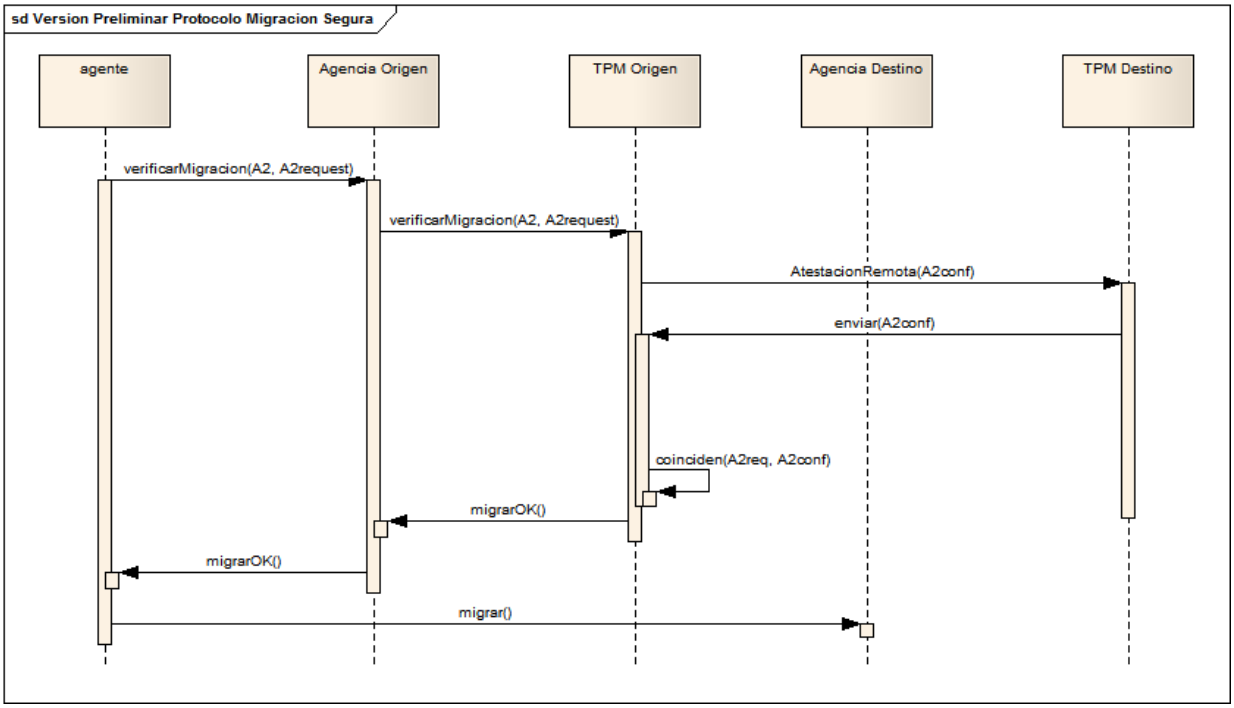

Figura 3.6: Descripción a alto nivel de la primera versión del protocolo de migración segura.

Entre las ideas más relevantes que aporta a nuestra solución resaltamos el hecho de que un agente de la agencia, usando el TPM, obtiene los valores firmados de los registros PCR. Estos registros PCR forman parte del TPM y el valor almacenado en ellos es el resultado de aplicar una función hash que toma como entrada información momentánea del estado del sistema. En esta información se tiene constancia de las aplicaciones que están en ejecución en dicho instante y la secuencia de activación de las mismas entre otras cosas. Por otro lado, el protocolo muestra también la forma en que el agente obtiene las credenciales de la plataforma, que junto con los valores firmados de los PCR permiten determinar si la configuración de la agencia destino es confiable o no y por tanto determinar la seguridad del proceso.

La idea del uso de credenciales es interesante, ya que unas credenciales certifican que los valores PCR firmados pertenecen a la agencia que se toma como objeto de atestación. Basándose en el mismo principio que una PKI. De este modo podemos considerar que las credenciales se usan para autenticar a la agencia, mientras que los registros PCR atestiguan el estado de la misma. Para entender la utilidad de estas credenciales vamos a ver dos protocolos que nos permiten realizar una atestación remota de la configuración. El primer protocolo, es un protocolo de atestación bidirecciónal. Este hace uso de una Autoridad de Certificación (Certification Authority, CA). Vemos una descripción de cada uno de los pasos del protocolo.

1. La agencia origen solicita los datos firmados de su configuración usando la funcionalidad del TPM.

2. La agencia origen envía a la agencia destino los datos firmados de su configuración junto con la clave pública de la AIK con la que fueron firmados.

3. La agencia destino valida la autenticidad de la clave recibida mediante la clave pública de la CA que certificó la clave.

4. La agencia destino verifica los datos firmados que ha recibido usando la clave pública de la AIK.

5. La agencia destino verifica que los valores de configuración recibidos se encuentran dentro del conjunto de valores aceptados y que, por lo tanto, la configuración de la agencia origen es confiable.

6. La agencia destino solicita los datos firmados de su configuración.

7. La agencia destino envía a la agencia origen los datos firmados de su configuración junto a la clave pública de la AIK con la que se generó la firma.

8. La agencia origen valida la autenticidad de la clave recibida mediante la clave pública de la CA que certificó la clave.

9. La agencia origen verifica los datos firmados que ha recibido usando la clave pública de la AIK.

10. La agencia origen verifica que los valores de configuración recibidos se encuentran dentro del conjunto de valores aceptados y que, por lo tanto, la configuración de la agencia origen es confiable. En este momento existe una relación de confianza mutua entre las agencias.

La solución planteada mediante el uso de una CA externa nos proporciona ciertas ventajas aunque también plantea algunos inconvenientes. Las ventajas son evidentes, especialmente en lo relacionado con la seguridad. Sin embargo, es importante resaltar el hecho de que hemos de mantener un gran número de certificados en el sistema, especialmente si el número de agencias del sistema llega a alcanzar envergadura suficiente.

Sin embargo la principal desventaja en el uso de una CA externa radica en la pérdida de autonomía en los propios agentes. Como ya se ha explicado en capítulos previos, una de las ventajas del uso de agentes móviles es precisamente por su autonomía. Esta característica se puede ver mermada por la implicación del uso de una CA de la que depende de forma directa el agente para su ejecución. Este hecho nos ha llevado a replantearnos esta solución puesto que, en cierta medida, va en la dirección opuesta a la filosofía del agente software con sus características especiales de autonomía. Por ello decidimos profundizar en el estudio de un segundo protocolo de atestación remota.

Este protocolo es conocido como atestación anónima directa (Direct Anonymous Attestation, DAA). En nuestra solución, la idea es que reemplace el uso de una CA por una autoridad emisora de certificados para claves DAA. El protocolo podemos verlo en la figura 3.7. Como veremos, esta alternativa no es ideal para todos los casos. De hecho hemos estudiado cierto conjunto de sistemas de agentes para los que es mejor usar la idea inicial basada en el uso de una autoridad de certificación (CA) y otros para los que el uso

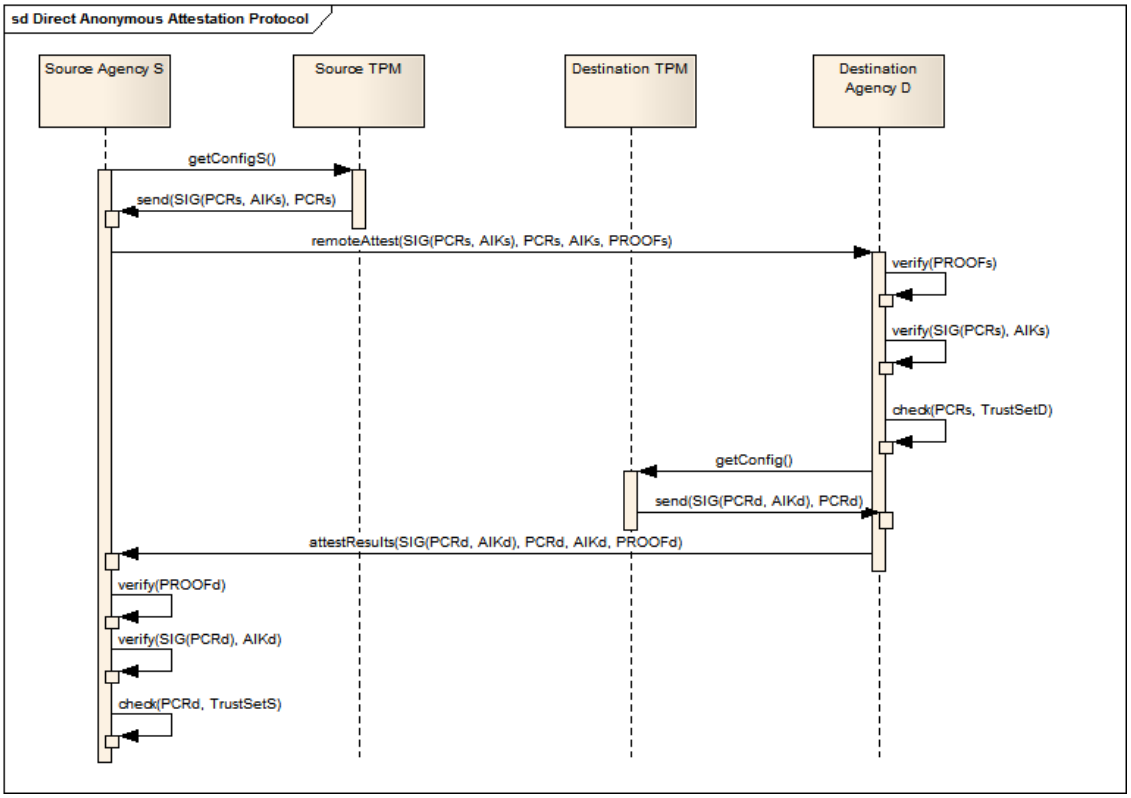

Figura 3.7: Protocolo de atestación anónima directa.

de un protocolo de atestación anónima directa (DAA) se ajusta mejor. En concreto, el modelo basado en un protocolo que hace uso de una CA externa tiene la ventaja de que ambas partes obtienen la información completa de la identidad del resto de las partes, con las ventajas evidentes de esta característica. En cambio el modelo basado en un protocolo de DAA, que evidentemente es mucho más eficiente, no ofrece toda la información de las partes a atestar. Evidentemente esta característica lo posiciona de forma ventajosa frente al basado en CA en un modelo en el que tengamos como prioridad absoluta la privacidad. En cambio, en otro tipo de modelo, más acorde con las situaciones reales, es imprescindible conocer las partes con la consiguiente carga de responsabilidad relacionada.

Los pasos que sigue el protocolo son los siguientes:

1. La agencia origen solicita a su TPM los datos firmados de su configuración.

2. La agencia origen envía a la agencia destino los datos firmados de su configuración junto a la clave pública de la AIK con la que fueron firmados y una prueba criptográfica de que posee:

   Un certificado para la AIK generado usando una clave DAA.

   Un certificado para la clave DAA generado por un emisor de certificados DAA.

3. La agencia destino verifica la prueba criptográfica y por lo tanto la validez de la AIK.

4. La agencia destino verifica los datos firmados que ha recibido usando la clave pública de la AIK.

5. La agencia destino verifica que los valores de configuración recibidos se encuentran dentro del conjunto de valores aceptados y que por lo tanto la configuración de la agencia origen es segura.

6. La agencia destino solicita los datos firmados de su configuración haciendo uso del TPM.

7. La agencia origen envía a la agencia destino D los datos firmados de su configuración junto a la clave pública de la AIK con la que fueron firmados y una prueba criptográfica de que posee:

   Un certificado para la AIK generado usando una clave DAA.

   Un certificado para la clave DAA generado por un emisor de certificados DAA.

8. La agencia origen verifica la prueba criptográfica y por lo tanto la validez de la AIK.

9. La agencia origen verifica los datos firmados que ha recibido usando la clave pública de la AIK.

10. La agencia origen verifica que los valores de configuración recibidos se encuentran dentro del conjunto de valores aceptados y que por lo tanto la configuración de la agencia origen es segura. En este momento existe una relación de confianza mutua entre ambas agencias.

Ambos protocolos tienen la ventaja de que permiten a la entidad verificadora confiar en que la AIK usada para firmar los valores de configuración pertenece a la misma entidad que generó la firma y que esta dispone de un TPM cuya funcionalidad se ha usado para generar la AIK. Por su parte, el protocolo basado en CA presenta dos desventajas añadidas a las ya comentadas:

- Es necesario crear un certificado para cada AIK, lo que convierte a la CA en un cuello de botella. Y más aún si tratamos con un sistema en el que hay un número elevado de peticiones simultáneas al CA como puede ocurrir en un sistema multiagente real.

- La entidad verificadora y la CA pueden actuar conjuntamente vulnerando la seguridad del sistema en un caso hipotético. Esta posibilidad es bastante remota, aunque es evidente que utilizar una infraestructura de cierto volumen como es el caso de la basada en CA para encontrarnos con otros problemas de seguridad merece cierta meditación a cerca de su utilidad real.

A pesar de que estos inconvenientes no aparecen en el protocolo DAA, este presenta otros inconvenientes. El principal de todos viene dado por la falta de un protocolo para que la entidad verificadora pueda determinar que la entidad verificada posee efectivamente un certificado de la AIK. Con lo cual en caso de que se intente atacar al sistema se podría detectar pero no sería tan obvio localizar el origen de este ataque.

En nuestro desarrollo hemos estudiado ambos protocolos, sin embargo la librería de migración segura usa el protocolo basado en el uso de una CA, debido a que usamos la librería tpm4java descrita en el capítulo 2, la cual por su parte no tiene implementada la funcionalidad necesaria para desarrollar un protocolo de atestación anónima directa. Tras un análisis de las distintas alternativas, hemos desarrollado un protocolo basado en CA más eficiente, que describimos en detalle a continuación:

1. La agencia usa un TPM, instalado en la máquina en la que se está ejecutando, que le proporciona los valores de configuración contenidos en los PCR.

2. Se utilizan las funcionalidades proporcionadas por el TPM instalado para producir la firma de los valores de los PCR, haciendo uso de una AIK específica para cada agencia destino. De tal forma que el receptor de los datos conozca con certeza la identidad del TPM que los firmó.

3. Una CA (Autoridad de certificación) se encarga de generar las credenciales necesarias para que se pueda verificar de forma correcta la identidad de la AIK.

4. Junto con los valores firmados de los PCR la agencia proporciona las credenciales de la AIK para que el solicitante pueda verificar correctamente que los datos provienen del TPM de la agencia.

A continuación analizamos detalladamente el protocolo que implementa nuestra librería, este se muestra en la figura 3.9.

Los pasos que sigue el protocolo son los siguientes:

1. El agente ag solicita a su agencia la migración hacia la agencia destino.

2. La agencia origen envía a la agencia destino una petición de atestación.

3. La agencia destino acepta la petición y envía un nonce (valor compuesto por bits aleatorios usados para evitar ataques de repetición) y los índices de los PCR cuyos valores desea conocer.

4. La agencia origen utiliza su TPM para obtener los valores de los PCR solicitados por la agencia destino junto con el nonce todo en un valor de firma.

5. La agencia origen obtiene las credenciales de la AIK de su repositorio de credenciales.

6. La agencia origen envía a la agencia destino el producto de la firma. También envía las credenciales de la AIK, que contienen la clave pública correspondiente a la clave privada con la que fueron firmados los datos. Además envía también un nonce y los índices de los PCR cuyos valores desea conocer.

7. La agencia destino valida la autenticidad de la clave recibida verificando las credenciales mediante la clave pública de la CA que generó dichas credenciales.

8. La agencia destino verifica la firma de los valores de los PCR y el nonce que ha recibido usando la clave pública de la AIK.

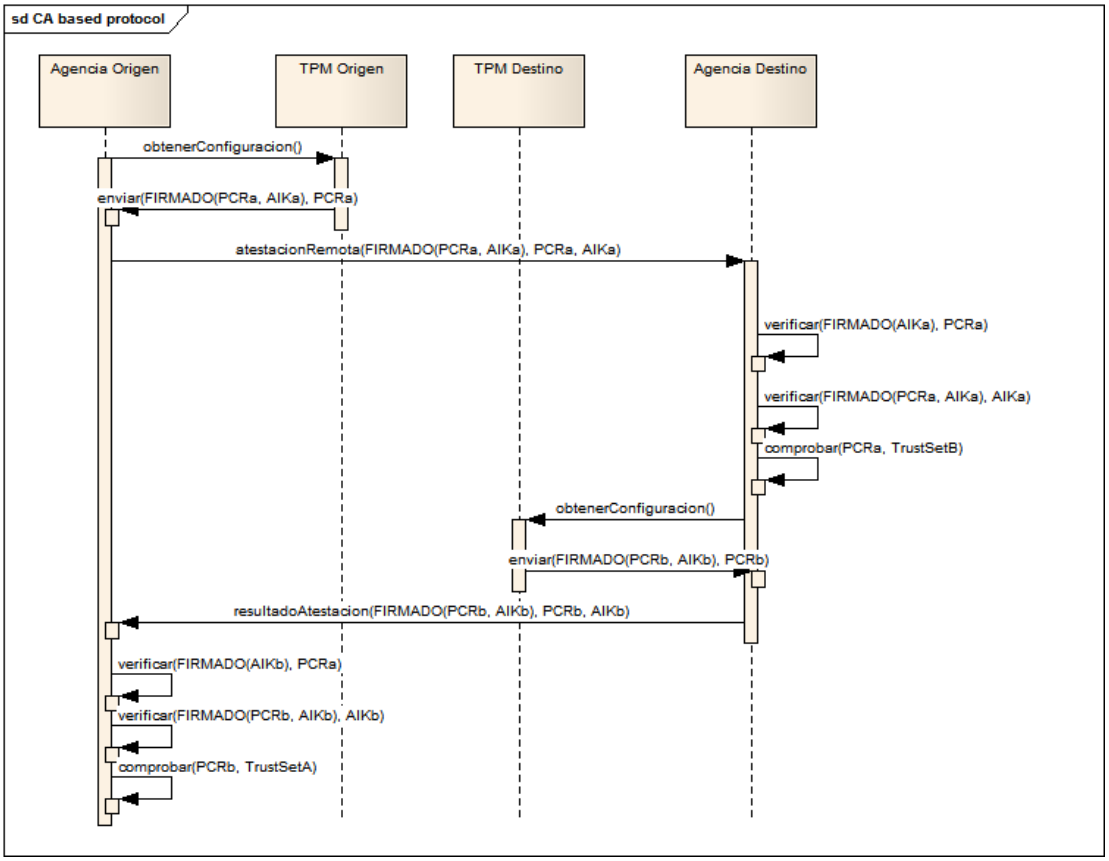

Figura 3.8: Descripción del protocolo de migración segura basado en el uso de una CA.

9. La agencia destino verifica que los valores de los PCR recibidos se encuentran dentro del conjunto de valores aceptados y que por lo tanto la configuración de la agencia origen es segura.

10. La agencia destino usa su TPM para obtener la firma de los valores de los PCR solicitados por la agencia origen junto con el valor de nonce.

11. La agencia destino obtiene las credenciales de la AIK de su repositorio de credenciales.

12. La agencia destino envía a la agencia origen la firma completa junto con las credenciales de la AIK, que contienen la clave pública correspondiente a la clave privada con la que se firmaron los datos.

13. La agencia origen valida la autenticidad de la clave recibida verificando las credenciales mediante la clave pública de la CA que generó dichas credenciales.

14. La agencia origen verifica la firma de los valores de los PCR y el nonce que ha recibido usando la clave pública de la AIK.

15. La agencia origen verifica que los valores de los PCR recibidos se encuentran dentro del conjunto de valores aceptados y que por lo tanto la configuración de la agencia destino es segura. En este momento existe confianza mutua entre ambas agencias.

16. La agencia origen concede al agente la migración hacia la agencia destino solicitada.

Este protocolo cumple con los requisitos que mencionábamos al inicio del presente capítulo. Por esta razón basamos nuestra solución en la seguridad proporcionada por este protocolo.

Antes de continuar esta investigación hicimos un alto en el camino y nos replanteamos las bases sobre las que habíamos creado nuestra solución, con idea de identificar posibles puntos débiles y así poder reforzarla, en caso de que fuese posible, o plantear una nueva alternativa si fuera necesario. Para ello decidimos someter a nuestro protocolo a un proceso de estudio y validación formal que hemos descrito en este capítulo.

3.2. Consideraciones sobre el método de migración

En primer lugar, vamos a definir una serie de modelos de atacantes contra nuestro sistema. El propósito principal de esto consiste en la justificación de los pasos del protocolo así como de los mecanismos de seguridad empleados en el mismo. Vamos a describir cada uno de los modelos de atacante en detalle, concluyendo que nuestro protocolo es efectivamente resistente todos ellos, mostrando así la validez del mismo.

3.2.1. El atacante no dispone de una agencia en su poder.

En este caso, el protocolo para llevar a cabo la migración de forma segura resulta bastante efectivo, ya que el agente conoce a la agencia destino, a la que pretende migrar,

con lo cual este ataque queda frustrado. Es evidente que el hecho de disponer de una agencia en la que se pueda ejecutar el agente es un requisito indispensable, precisamente por su obviedad consideramos trivial este caso.

3.2.2. El atacante dispone de una agencia pero esta no es confiable.

La prueba de que nuestro protocolo es resistente a los agentes realizados por este atacante es bastante sencilla. En el segundo paso del protocolo se realiza la petición de atestación desde la agencia origen a la destino, en el instante siguiente la agencia destino devuelve el nonce y los índices de PCRs que desea conocer, con lo cual en el cuarto paso la agencia origen solicita esos valores junto con el nonce firmado. En el séptimo paso del protocolo se envía la información que ha de validarse en la agencia destino y puesto que esta viene firmada es imposible y por tanto no puede continuar. Con lo cual el ataque es frustrado y queda demostrada la validez del diseño de nuestro protocolo, ya que este tipo de protección fue el que se planteo inicialmente.

3.2.3. El atacante conoce un destino adecuado

En este modelo de ataque el propio atacante conoce un destino adecuado, la intención del agente de migrar y la agencia destino a la que pretende hacerlo. Además de tener acceso al canal de comunicación entre ambos. El ataque consiste en realizar una suplantación consistente el completar todos los pasos de forma correcta, hasta el punto en que se alcanza la migración a la agencia destino, es decir, en el protocolo que hemos comentado anteriormente, justo en el paso en que se solicita la migración del agente. El atacante intercepta el mensaje, de forma que el agente no migra a la agencia solicitada sino a una tercera que no es confiable y que está dentro de los dominios del atacante.

Para resolver este problema, nos planteamos el uso de ciertos mecanismos criptográficos adicionales. La solución que proponemos se basa en que el agente cifre parte de su código haciendo uso de una clave generada en el TPM destino previamente a la migración, esta clave a su vez está ligada, tal y como hemos descrito en la sección anterior. De esta forma el problema queda resuelto.

3.2.4. El atacante dispone de una agencia confiable y con su TPM funcionando correctamente, además de disponer de otra agencia manipulada.

Nos planteamos el caso de que el atacante tenga en su domino la agencia destino funcionando de forma confiable con su TPM correspondiente. El ataque se basa en que puesto que desde el paso 14 no se hacen más comprobaciones con el TPM ni con la agencia confiable, se puede reemplazar la agencia confiable por otra que no lo sea. Esta nueva agencia que no es confiable podría estar manipulada y se puede haber modificado el comportamiento del agente. A este problema se le conoce como el de "tiempo-de-chequeo tiempo-de-uso".

Asumimos que estamos en entornos muy restringidos de operación, de forma que vamos a acotar la ejecución para evitar que se pueda dar el caso de que se ejecute otra agencia con propósitos ilícitos en el mismo momento evitamos la ejecución simultánea de otro software que no sea el de la agencia confiable junto con el Sistema Operativo. Para ello nuestro Sistema Operativo interviene en la cadena de confianza, de forma que realiza una operación *PCRQuote* en la cual se toma la medida de los PCRs que va a ser válida para la posterior ejecución. Esta funcionalidad proporcionada por el TPM resulta de gran utilidad puesto que permite almacenar el estado actual del sistema en estos registros internos del TPM, conocidos como PCRs. El estado del sistema viene caracterizado tanto por el hardware como por el software del sistema, considerando relevante el orden secuencial de instalación de cada uno de los componentes software.

En realidad, esto supone una restricción muy importante en el uso, y más aún para sistemas tan versátiles como son los basados en agentes. Puesto que ello implica que cada una de las agencias debe tener el mismo software y en el mismo orden secuencial de instalación. En ciertos casos se puede proporcionar un conjunto muy reducido de posibilidades, por lo que nos planteamos otras formas para proporcionar un mayor grado de flexibilidad.

La siguiente estrategia permite incrementar el grado de flexibilidad de la atestación, esta consiste en establecer una lista de software "confiable" que podamos ejecutar de forma simultánea. Obviamente, esto acarrea una serie de inconvenientes, entre los que destacamos el hecho de que tendremos muchas combinaciones distintas posibles de software. Este número se ve ingentemente incrementado si se considera relevante la secuencia de los mismos, puesto que esto queda reflejado a la hora de tomar las medidas de los PCRs y la producción de la firma con el uso del TPM. No obstante, esto nos permite tener un conjunto de valores de QUOTEs válidos. Sin embargo, esta solución también es bastante limitada, puesto que el software a usar está muy restringido, lo cual no es conveniente para situaciones reales. Por ello nos planteamos un paso más avanzado consistente en tomar las medidas de PCRs necesarias para evitar que dos agencias puedan ejecutarse de forma simultánea en la misma máquina, lo cual puede resultar una restricción muy fuerte, especialmente si una de ellas no es confiable. Tomamos medidas de PCRs y averiguamos cuáles son los valores de la agencia confiable, y cuáles los de la agencia no confiable. Se mantiene un conjunto de valores posibles de PCRs que se ha demostrado confiable y lo único que habrá que realizarse es una comprobación del valor a medir en este conjunto.

Aunque se están estudiando otras alternativas para dar solución a este problema. La más avanzada se basa en el uso de entornos virtualizados, entre los cuales las características del software y su secuencialidad pueden ser idénticos.

Otra iniciativa que nos planteamos está relacionada con un posible cambio de estado del sistema que implique una pérdida en la confiabilidad de la agencia destino. Para ilustrar este problema proponemos el ejemplo dado por la migración segura entre agentes que hemos diseñado. Se realiza la comprobación de la migración posible, a continuación se chequea que efectivamente todo funciona de forma correcta y se informa desde la agencia destino a la agencia origen que la migración se puede iniciar (paso 14 de nuestro protocolo). El ataque viene producido por que en ese preciso instante, un usuario con intenciones maliciosas suplante la agencia con código confiable y se pone en marcha la ejecución de otra agencia con intenciones maliciosas. Esta nueva agencia, con carácter malicioso, podría pretender manipular al agente y obtener así algún tipo de beneficio de

esta práctica. Hasta este punto es el mismo problema descrito anteriormente. Sin embargo, para solucionar este problema nos planteamos una solución más elegante, mediante la introducción de una nueva estrategia de protección. Esta nueva estrategia consiste en la protección de los propios agentes haciendo uso de una funcionalidad específica que nos proporciona el TPM. La idea radica en el cifrado de parte del código del agente con una clave generada en el TPM de la plataforma destino, de manera que esta quede ligada a cierto valor de PCR, así en el supuesto de que las medidas tomadas no coincidan con los valores de PCRs no se permitirá la ejecución del agente, puesto que el agente no está en claro y no se puede lanzar.

El proceso se describe de la siguiente manera. Tenemos el agente ag que migra de la agencia origen (O) a la agencia destino (D) y tenemos el TPM de la Agencia destino (T).

1. ag ->A1 : solicitud de migración
2. A1 ->A2 : solicitud de migración
3. A1 ->A2 : envío (nonce, PCRids,ValorPCR)
4. A2 ->tpm2 : generar(KBindPCR), la agencia destino utiliza la funcionalidad ofrecida por el TPM
5. A2 ->tpm2 : generar certificado KBindPCR
6. A2 ->tpm2 : generar certificado AIK
7. tpm2 ->A2 : enviar(KBindPCR)
8. tpm2 ->A2 : enviarCertificado KBindPCR
9. A2 ->A1: enviarCertificado KBindPCR
10. A2 ->A1: enviarCertificado AIK

La funcionalidad que usamos se conoce como *sellado-ligado (sealed binding)*, viene recogida en el estándar del TPM descrito por el TCG que presentamos en el captítulo 2. Esta consiste en que los mensajes a los que se les aplica esta operación quedan ligados a un estado de la plataforma, de esta manera el mensaje sólo puede ser descifrado si la plataforma del receptor se encuentra en dicho estado. Esta operación asocia un mensaje con una serie de valores de los PCR y con una clave asimérica no migrable, de esta forma el receptor sólo puede descrifrar el mensaje si el TPM posee la clave de cifrado correspondiente, así como los valores de sus PCS son los determinados por el emisor. A pesar de ser una opción muy restrictiva actualmente se está trabajando en alternativas más versátiles tal y como se describen en la sección de trabajos futuros.

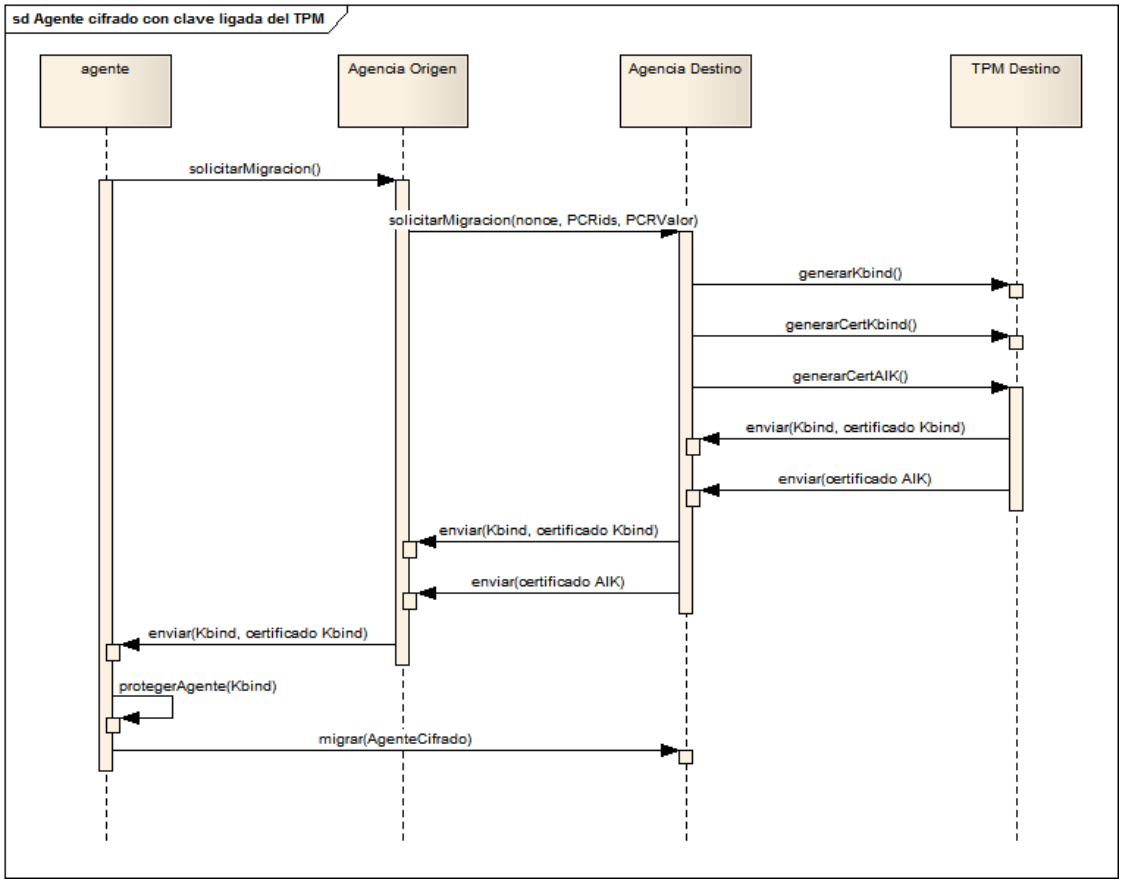

Figura 3.9: Descripción del protocolo que usa clave ligada.

3.3. Evaluación

En esta sección presentamos la validación del trabajo realizado hasta el momento. Este análisis lo hemos realizado a dos niveles distintos. En primer lugar, hemos llevado a cabo un análisis formal de los protocolos que intervienen en nuestra solución, que se han modelado mediante una herramienta de verificación formal. En segundo lugar, hemos analizado la implementación de la librería en sí, mediante un estudio del código y se han probado en escenarios reales.

3.3.1. Validación de los protocolos con AVISPA

Para la validación de los protocolos hemos hecho uso de un conjunto de herramientas caracterizadas como "model checking". Estas herramientas consisten en un método automático de verificación de un sistema formal, en la mayoría de las ocasiones derivado del hardware o del software de un sistema informático. La especificación consta de dos partes: el modelo del sistema y los objetivos de la prueba. El sistema a analizar se describe mediante un modelo que suele estar expresado como un sistema de transiciones, es decir, un grafo dirigido, que consta de un conjunto de vértices y arcos. Se asocia a cada nodo un conjunto de proposiciones atómicas. Los nodos representan los estados posibles de un sistema, los arcos representan las posibles evoluciones del mismo. Mediante las ejecuciones permitidas, se puede alterar el estado, mientras que las proposiciones representan las propiedades básicas que se satisfacen en cada punto de la ejecución.

Existen herramientas automáticas para realizar el Model checking, basadas en técnicas combinatorias, explorando el espacio de estados posibles. La consecuencia directa de esta estrategia es que para sistemas grandes se produce el problema de una explosión combinatoria de espacio de estados. Para evitarlo, diversos investigadores han desarrollado técnicas basadas en algoritmos simbólicos, abstracción, reducción de orden parcial y model checking on-the-fly.

En nuestro caso hacemos uso de un conjunto de herramientas concretas denominado AVISPA. Son las siglas del inglés Automated Validation of Internet Security Protocols and Applications. Estas herramientas permiten la validación automatizada de protocolos de seguridad mediante la formalización de los mismos en un lenguaje propio HLPSL (high-level protocol specification language). Antes de comenzar con la validación formal vamos a describir de forma sucinta en qué consiste AVISPA y su propósito.

Como hemos subrayado AVISPA es una herramienta de validación formal de protocolos de seguridad desarrollada en el marco de un proyecto, que ha sido financiado por la Comisión Europea. Este conjunto de herramientas cubre todo tipo de protocolos de seguridad en las cinco capas de OSI para más de veinte servicios y mecanismos distintos. Además de ello, esta herramienta se ha usado para validar más de 85 especificaciones de seguridad de la IETF. La librería de AVISPA está disponible en la red y se ha utilizado para resolver cientos de problemas en un gran número de protocolos de seguridad.

Como antes mencionábamos AVISPA utiliza un lenguaje específico HLPSL, del inglés High Level Protocol Specification Language, es decir lenguage de alto nivel de especificación de protocolos. HLPSL es un lenguaje de alta expresividad y muy intuitivo que facilita el modelado de los protocolos. Su semántica operacional se basa en la lógica temporal de

Lamport.

Todas las comunicaciones que se realizan en AVISPA son síncronas y el mecanismo de funcionamiento es el siguiente: El protocolo expresado en HLPSL es traducido a un formato intermedio denominado IF (Intermediate Format). En este paso intermedio se reescriben las reglas aplicadas para procesarlas en un paso posterior del proceso, de forma que un protocolo escrito en este lenguaje intermedio se ejecuta mediante un número de iteraciones finito, a menos que se encuentre algún bucle que se deba a algún tipo de ataque.

En el caso de no existir tal bucle, se considera que el protocolo es seguro, al menos en el número de iteraciones indicadas. El comportamiento del sistema en HLPSL se modela como un estado. Cada estado tiene variables que son las responsables de las transiciones. Así cuando una de estas variables cambia, el estado toma una nueva forma. Por otro lado, las entidades que se comunican son denominadas roles, y poseen variables propias. Estas variables pueden ser de ámbito local o global. Aparte de la entidad iniciadora y la receptora, el entorno y la sesión del protocolo de ejecución son modelados como roles en HLPSL. Los roles pueden ser básicos o compuestos, en función de si están formados por un agente o por más. Cada participante honesto o parte principal tiene un rol, este puede ser paralelo, secuencial o compuesto. Toda comunicación entre roles e intruso son síncronas, como antes se comentaba. Los canales de comunicación se representan por variables que contienen propiedades de un entorno en concreto.

El lenguaje usado en AVISPA es muy expresivo y permite describir detalles de un grano muy fino. Una de las ventajas principales de este conjunto de herramientas es que los resultados en AVISPA están detallados y se obtienen tras un número alcanzable de estados [53]. Por lo tanto, la interpretación de estos resultados no requiere de una destreza ni un aprendizaje profundo de la herramienta en sí, a diferencia de otras herramientas similares existentes, como es el caso de HERMES[54], en las que se requiere una destreza contrastada para la obtención de conclusiones.

El conjunto de herramientas de AVISPA nos proporciona cuatro motores de ejecución distintos. El primero de ellos se conoce como On-the-fly Model-Checker (OFMC) [55] que lleva a cabo una falsificación del protocolo y una verificación ligada mediante la exploración del sistema de transición descrito por la especificación IF, es importante destacar que se realiza de una forma dirigida por demanda. OFMC implementa cierto número simbólico de técnicas correctas y completas. Además soporta la especificación de las propiedades algebraicas de operadores criptográficos y modelos tanto tipados como no-tipados. El segundo de ellos se conoce como el Constraint-Logic-based Attack Searcher (CL-AtSe) que se basa en aplicar restricciones tal y como se muestra en [53], es interesante la capacidad de reducir la redundancia y la simplificación heurística. CL-AtSe se construye de forma modular y está abierta a extensiones para el manejo de propiedades algebraicas de operadores criptográficos. El tercer motor se conoce como el SAT-based Model-Checker (SATMC) [56] que se basa en la construcción de una fórmula proposicional que codifica una relación de transición especificada por el IF, se representa el estado inicial y el conjunto de estados que representan una violación a las propiedades de seguridad. Por último el TA4SP (Tree Automata based on Automatic Approximations for the Analysis of Security Protocols) [57] que realiza una aproximación del conocimiento del intruso mediante la construcción de un árbol.

En nuestra validación hemos utilizado el modelo OFMC dado que es el único que

utiliza valores nuevos para la generación de los nonce's. Un nonce, es una cadena de bits que se genera una única vez, es un mecanismo usado en algunas técnicas criptográficas para evitar ciertos ataques basados en repetición.

A continuación vemos un ejemplo del código en HLPSL utilizado para la verificación del protocolo de migración seguro basado en el uso de una CA que hemos desarrollado para nuestro propósito, además de una explicación de los resultados obtenidos [58].

En el modelo HLPSL de nuestro protocolo hemos definido varios roles, agent, agency_A, agency_B, TPM_agency_A, TPM_agency_B y environment. El rol agent tiene dos estados distintos; 0 para dar comienzo a la migración, y 1 para realizar la migración en sí. El modelo del agente es como sigue.

```
role agent(O,Ag:agent,SND,RCV:channel(dy)) played_by Ag
def=
local
State : nat
const
migrate : protocol_id
init
State := 0
transition
1.State = 0 /\ RCV(start) =|> SND(migrate.O) /\ State' := 1
2.State = 1 /\ RCV(migrate.Ag) =|> State' := 2
end role
```

El código muestra cómo el conjunto de transiciones está formado por tres únicos estados. El primero de ellos, caracterizado como estado 0 recibe la señal de comienzo y envía migrar al rol origen (O) y pasa al segundo estado de este rol (estado 1). El segundo estado espera a recibir el mensaje de migración del agente y pasa al tercer estado (estado 2).

El siguiente rol modela la Agencia origen de nuestro protocolo de migración segura. Este tiene como parámetros a la agencia origen (O), la agencia destino (D) el agente (Ag), el TPM de la plataforma de origen y los canales de envío y recepción. Consta de cuatro estados que comentamos a continuación.

```
role agency_O(O,D,Ag,TPM_O:agent,SND,
RCV:channel(dy)) played_by O
def=
local
State : nat,
Nonced,Nonceo : message,
Value : message,
Pcrid : nat set,
AIKod,AIKdo : public_key,
TrustsetO : message set
const
migrate,remoteattestrequest,remoteattestagree,
getconfig : protocol_id,
getCredentials : public_key -> message,
hashf : hash_func,
pcria : message,
pcra,pcrb : message->message
init
TrustsetO := {pcrb(pcrio)} /\ State := 0
transition
1.State = 0 /\ RCV(migrate.O) =|>
SND(remoteattestrequest.D) /\ State' := 1
```

Cuando se alcance el estado cero y se reciba el mensaje de migrar, procedente de la agencia origen (migrate.O) se envía el mensaje (remoteattestrequest.D) a la agencia destino y se cambia al nuevo estado 1, mediante la orden State' := 1.

```
2.State = 1 /\ RCV(remoteattestagree.Nonced'.
Pcrid'.O) =|>
SND(getconfig.Nonced'.Pcrid'.TPM_O) /\ State' := 2
```

Cuando estemos en el estado número 1, se solicitará la respuesta de aceptación de atestación remota de la agencia destino junto con el valor aleatorio del destino (nonced) y el conjunto de valores de *pcrs* produciendo la firma en el origen. Se enviará el mensaje para obtener la configuración con el nonce del destino y el conjunto de valores de Pcrs del destino y seguidamente se cambiará el estado al valor 2.

```
3.State = 2 /\ RCV(pcro(Pcrid).Nonced.{hashf(pcro(Pcrid).
Nonced)}_inv(AIKod').AIKod'.TPM_O)=|>
Nonceo':=new() /\SND(pcro(Pcrid).Nonced.
{hashf(pcro(Pcrid).Nonced)}_inv(AIKod').
AIKod'.getCredentials(AIKod').Nonceo'.pcria.D)/\
State':=3
```

En el siguiente paso, nos situamos en el tercer estado caracterizado como estado 2, se recibirá el valor del conjunto de pcrs del origen firmados junto con el nonce del destino y la clave pública del par de claves agencia origen-agencia destino. Y se creará un valor nonce en el origen, para evitar ataques por repetición mediante algún tipo de inundación y se enviarán igualmente los valores de pcrs firmados junto con el nonce del destino.

```
4.State = 3 /\RCV(Value'.Nonceo.{hashf(Value'.
Nonceo)}_inv(AIKdo').
AIKdo'.getCredentials(AIKdo').D) /\in(Value',
trustsetO) =|> SND(migrate.Ag)
end role
```

Para finalizar, cuando el flujo del programa esté en el estado 3 y se reciba la información correspondiente, comprobando que los pares de claves son correctos se procederá a enviar el mensaje de migrar el agente (migrate.Ag). La agencia destino del protocolo se modela por medio del rol agency_D, que como vemos es simétrico al anterior y no vamos a entrar en redundancia con más repetición del mismo mecanismo. Únicamente cabe resaltar el hecho de que el último paso solamente tendrá sentido para el rol correspondiente a la agencia origen, puesto que la agencia destino no tiene la capacidad de conceder la migración al agente.

```
\scriptsize
\begin{verbatim}
role agency_B(O,D,TPM_D:agent,SND,
RCV:channel(dy)) played_by D
def=
local
State : nat,
Nonced,Nonceo : message,
Value : message,
Pcrio : nat set,
AIKod,AIKdo : public_key,
TrustsetD : message
const
```

```
migrate,remoteattendrequest,remoteattestagree,
getconfig:protocol_id,
getCredentials : public_key->message,
hashf : hash_func,
pcrid : message,
pcro,pcrd : message->message
init
TrustsetD := {pcro(pcrid)} /\ State := 0
transition
1.State = 0 /\ RCV(remoteattendrequest.
O) =|> Nonced’:=
new() /\ SND(remoteattestagree.Nonced’. pcrib.O) /\
State’ := 1
2.State = 1 /\ RCV(Value’.Nonced.
{hashf(Value’)}_inv(AIKod’).
AIKod’.getCredentials(AIKod’).
Nonceo’.Pcrid’.D)/\
in(Value’,TrustsetD) =|>
SND(getconfig.Nonceo’.
Pcrio’.TPM_D) /\ State’:=2
3.State = 2 /\ RCV(pcrd(Pcrio).
Nonceo.{hashf(pcrd(Pcrio).
Nonceo)}_inv(AIKdo’).AIKdo’.
TPM_D) =|>
SND(pcrd(Pcrio).Nonceo.
{hashf(pcrd(Pcrio).
Nonceo)}_inv(AIKdo).AIKdo’.
getCredentials(AIKdo’).O)
end role
```

Hemos modelado todo el funcionamiento de la parte del protocolo correspondiente a la generación de las claves, la firma y la función quote del TPM dentro de un rol al que hemos denominado rol del TPM instalado en la agencia origen, TPM_O. La función quote es un comando de TPM que nos comunica el valor cifrado almacenado en los *pcrs* como hemos descrito en secciones previas. Aunque en realidad esto corresponde al software que realiza estas funciones no al dispositivo TPM propiamente dicho. Este rol tiene como parámetros la agencia origen (O), el dispositivo TPM instalado en la agencia origen (TPM_O) y los canales de envío y recepción de mensajes, que al igual que todos los modelados en este trabajo siguen el modelo de ataques de DolevYao [59].

```
role tpm_Agent_O(O,TPM_O:agent,SND,
RCV:channel(dy)) played_by TPM_O
def=
local
State : nat,
Nonceb : message,
Pcrid : nat set
const
getconfig : protocol_id,
pcro : message -> message,
aikod : public_key,
hashf : hash_func
init
State := 0
transition
1.State = 0 /\ RCV(getconfig.
Nonced’.Pcrid’) =|> SND(pcro(Pcrid’).Nonced’.
{hashf(pcro(Pcrid’).Nonced’)}_inv(aikod)
.aikod.O) /\ State’ := 1
end role
```

Este rol únicamente consta de un estado, y su funcionamiento consiste en producir la firma con el nonced (el valor nonce generado por la agencia destino) y la clave apropiada cuando

se reciba una petición por el canal de entrada. A continuación tenemos el modelado del rol del TPM de la agencia destino, que es análogo al anterior, por lo que no entramos en más detalle.

```
role tpm_Agent_D(D,TPM_D:agent,
SND,RCV:channel(dy)) played_by TPM_D
def=
local
State : nat,
Noncea : message,
Pcria : nat set
const
getconfig : protocol_id,
pcrb : message -> message,
aikba : public_key,
hashf : hash_func
init
State := 0
transition
1.State = 0 /\ RCV(getconfig.
Nonceo'.Pcrio') =|> SND(pcrd(Pcrio').
Nonceo'.{hashf(pcrd(Pcrio').Nonceo')}_inv(aikdo).
aikdo.O) /\ State' := 1
end role
```

Hemos modelado un rol correspondiente a la vida de una sesión, lo hemos denominado así puesto que un agente puede migrar multitud de veces en su vida, con idea de plasmar esto en el modelado lo hemos diseñado de esta forma.

```
role session(O,D,TPM_O,TPM_D,Ag:agent)
def=
local SND1,SND2,SND3,SND4,SND5,RCV1,
RCV2,RCV3,RCV4,RCV5:channel(dy)
composition
agent(O,Ag,RCV1,SND1)/\ agencyO(O,D,Ag,
TPM_A,SND2,RCV3)
/\ agency_B(O,D,TPM_D,SND3,RCV3)
/\ tpm_Agent_A(O,TPM_O,SND4,RCV4)
/\ tpm_Agent_B(D,TPM_D,SND5,RCV5)
end role
```

Por último, tenemos la codificación del rol que describe el proceso completo. Hemos llamado a este rol "environment", encargado de orquestrar todos los roles previamente descritos. Este rol es indispensable para realizar la validación usando AVISPA.

```
role environment()
def=
const
o,d,tpm_o,tpm_d,ag:agent
intruder_knowledge = {o,d,tpm_o,tpm_d,ag,hashf}
composition
session(o,d,tpm_o,tpm_d,ag)
end role
environment()
```

En el modelo de atacante que hemos incluido, este no conocerá las claves generadas en los propios TPMs ni tampoco es capaz de obtener información de las operaciones de cifrado y descifrado realizadas por el dispositivo. Una vez ejecutamos este código con las herramientas proporcionadas por AVISPA obtenemos los siguientes resultados.

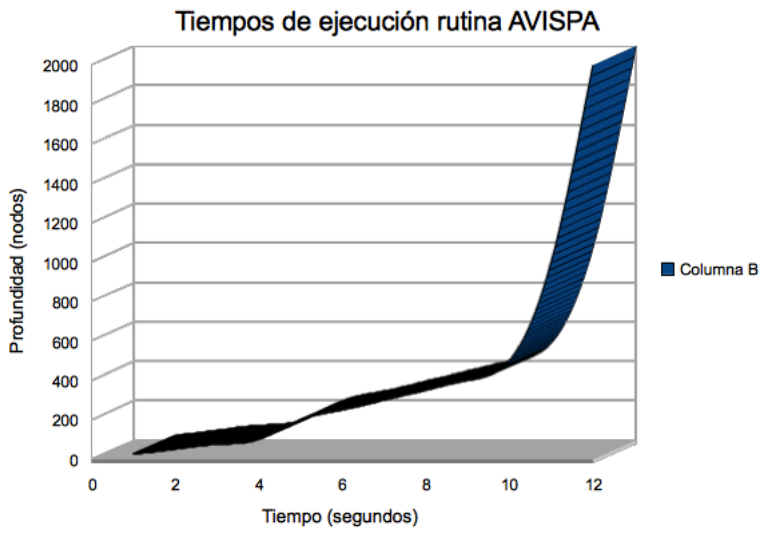

Figura 3.10: Ejecución de la rutina de AVISPA.

```
SUMMARY
SAFE
DETAILS
BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
/home/anto/avispa/testsuite/protocolo_2.txt.if
GOAL
as_specified
BACKEND
OFMC
COMMENTS
STATISTICS
parseTime: 0.00s
searchTime: 3.23s
visitedNodes: 4 nodes
depth: 200 plies
environment()
```

Como antes mencionábamos, una de las potenciales ventajas en el uso de AVISPA radica en la facilidad que encontramos en la interpretación de los resultados de salida. Podemos apreciar que los resultados nos muestran que el protocolo de migración queda validado conforme al modelo de atacante propuesto, que es precisamente el que toma como base las premisas indicadas, que a su vez es la razón principal por la que se hace uso del TPM. Además de ello, nos aparecen una serie de datos estadísticos, entre los que nos encontramos el número de nodos visitados en el parámetro depth. Hemos insertado diferentes valores para estas pruebas entre ellos 250, 300, 400, 500, 1000 y hasta 2000 obteniendo valores similares, evidentemente con un mayor retardo para cada caso. En la figura 3.10 se muestra una gráfica con los tiempos invertidos en ejecutar nuestro modelo en AVISPA, en función del límite en el grado de profundidad fijado.

3.4. Aplicación en sistemas reales

Un aspecto muy importante de cualquier tecnología nueva que se precie es su aplicación práctica para la resolución de problemas reales. Por este motivo hemos añadido esta sección en la que vamos a describir una aplicación de nuestra solución para solventar un problema real.

En primer lugar vamos a comentar la situación en que se encontraba el entorno antes de aplicar un sistema basado en agentes móviles seguros utilizando nuestra librería. Se nos plantea un sistema crítico como el de una planta nuclear, en la que nos encontramos una serie de medidas de seguridad muy importantes en gran parte del sistema. Observamos que existían ciertos puntos débiles que incrementaban en su número a medida que profundizábamos en el estudio. Algunos ejemplos de estos eran tan triviales como que la conexión a Internet del equipo que controlaba el sistema de la central era a través de una conexión wifi abierta.

Puesto que nuestro objetivo era proporcionar un alto nivel de seguridad alto al sistema, propusimos sacar provecho de las ventajas que los agentes móviles pueden aportar. Lo ideal sería diseñar un sistema de monitorización. Con el objetivo de apreciar cualquier anomalía por insignificante que parezca y poder reaccionar. Por supuesto, para poder realizar este sistema hemos de basarnos en un sistema seguro, puesto que no podemos construir una base fiable sobre algo que no tiene unos cimientos firmes.

En la ilustración 3.11 se muestra un diagrama que describe el funcionamiento global del sistema. Dentro de cada componente de la figura incluimos los agentes necesarios para realizar la monitorización del sistema. Por un lado tenemos el "Reactor Nuclear" que está conectado con las "Turbinas" al "Transformador" y este, a su vez, al "Tendido Eléctrico" que suministra energía eléctrica al sistema. Los "Calentadores" están conectados al "Reactor Nuclear". A su vez los "Calentadores" contienen "Condensadores". Por último, la "Torre de Refrigeración" está conectada al suministro de "Agua Fría".

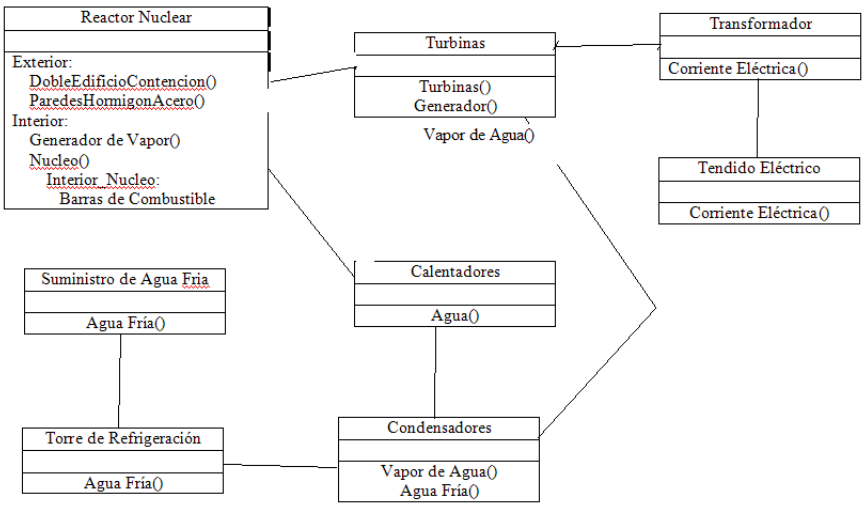

Figura 3.11: Funcionamiento global del sistema.

En la figura 3.12 se muestran los agentes involucrados en la monitorización de los

niveles de temperatura del sistema, “Refrigeración”, “Sistema de Circulación”, “Torres de Refrigeración”, “Canal de RecogidaAguas” y “Bombas Impulsión”.

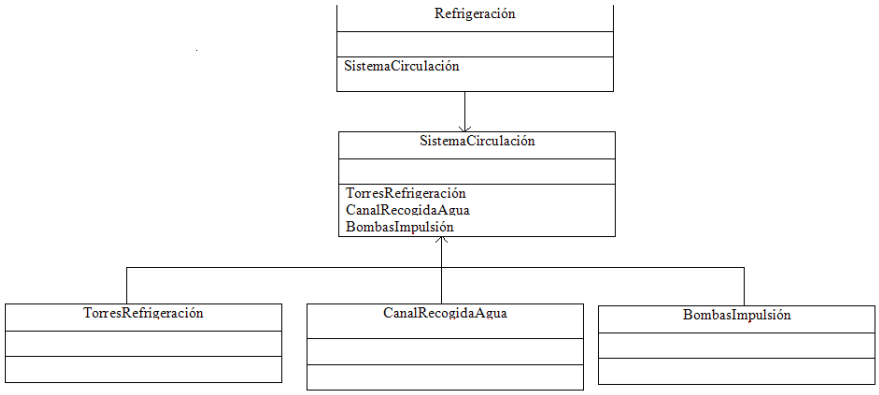

Figura 3.12: Diagrama UML del circuito de refrigeración del sistema de monitorización.

En la ilustración 3.13 aparece un diagrama en UML con las clases que componen la parte de seguridad del sistema completo “Seguridad” y “Gestión de Residuos Radiactivos” son las clases principales. La primera de ellas contiene “ProtecciondelReactos”, “BarrerasdeContencion” y “Software”.

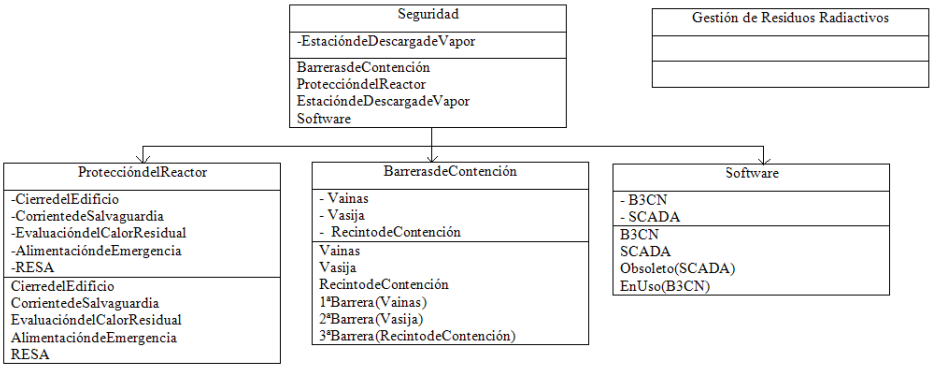

Figura 3.13: Diagrama UML sistema de seguridad.

Todo este sistema está desarrollado sobre la base de SecMiLiA descrita a lo largo de este capítulo. Defendemos la teoría de que para este tipos de sistemas críticos el uso de hardware criptográfico específico aporta un nivel de seguridad elemental que consideramos que es imprescindible para un sistema de estas características. Obviamente, las pruebas realizadas con este sistema las he llevado a cabo mediante el desarrollo de un simulador, dadas las condiciones específicas de este tipo de sistemas.

La aplicación de SecMiLiA en la protección de este sistema es directa. Para cada uno de los agentes descritos hemos creado instancias de la clase *SecureAgent* en lugar de los tradicionales de la clase agent *Agent*. La otra modificación que se ha tenido que realizar es cambiar la invocación a *migrate()* por *securemigration()*. Evidentemente, antes de realizar estos pasos se ha tenido que comprobar, en primer lugar la instalación correcta de JADE, seguidamente la del TPM y por último la de SecMiLiA.

3.5. Conclusiones y futuras mejoras

A lo largo de este capítulo hemos presentado SecMiLiA, que es una solución basada en JADE para la protección de sistemas de agentes. Hemos descrito los fundamentos teóricos de SecMiLiA, tanto a nivel de protocolos como de infraestructura básica, ya que la seguridad de esta solución radica en el uso de la tecnología de base TPM. Por último, hacemos una valoración de la solución tanto de la fundamentación teórica mediante el uso de técnicas formales de evaluación de protocolos, como desde el plano del uso práctico, a través de la evaluación en un caso real, más concretamente en un sistema de monitorización basado en agentes.

Entre las futuras mejoras proponemos las siguientes por considerarlas de mayor relevancia para nuestros objetivos.

Recuperación de estados anteriores. Como antes se mencionaba se puede dar el caso de que por cualquier circunstancia la máquina en la que está ejecutándose una agencia vea alterado su funcionamiento dejando de funcionar por cualquier razón. Por ello nos planteamos que sería interesante tomar medidas al respecto. Es decir, diseñar mecanismos de recuperación y de espera de estados previos.

Sistema de reputaciones. Una mejora podría consistir en la integración de un sistema de reputaciones. Se podría mantener un sistema de reputaciones con el objetivo de localizar las posibles agencias maliciosas por parte de los agentes. De forma que cuando ocurra algo no deseable, esto pueda ser monitorizado y notificado. Esto nos plantea otra serie de incógnitas, en primer lugar hasta qué punto es esto útil, ya que se puede dar la situación de que el sistema sea muy estricto y en el momento en que se detecte cualquier tipo de anomalía se notifique al sistema de reputaciones. Pudiendo penalizar a agencias por causas externas sin propósito malicioso alguno. Otra de las incogńitas que se nos plantean radica en la seguridad del propio sistema de reputaciones. Es importante plantearse la cuestión de quién controla este sistema de reputaciones. Puesto que el que lo controle puede obtener beneficios claros penalizando a ciertas agencias, e incluso favoreciendo otras que no fueran confiables.

Aplicar una autoridad de revocación de host (HoRA.) Otra propuesta que nos planteamos consiste en aplicar una autoridad de revocación de host. Esta propuesta consiste en la penalización de Agencias, una vez detectado su uso indebido en el sistema. Esta propuesta tiene una serie de limitaciones. Ya que se puede dar el caso de que ocurra un error no intencionado durante la ejecución de un agente, lo cual puede provocar que una agencia sea revocado. Aunque en múltiples ocasiones esto puede ser un castigo desproporcionado. HoRA podría aplicar políticas de castigo más relajadas ya que es dicha entidad la que controla quien debe ser revocado y quien no, e incluso revocar a los hosts en función de la gravedad de los ataques o el beneficio obtenido por los hosts maliciosos. Otra limitación radica en que la base de datos interna de HoRA crece indefinidamente, ya que no se eliminan nunca los hosts revocados, lo cual es necesario para que HoRA pueda controlar todo el historial de hosts revocados. En particular, se asume que la capacidad de

almacenamiento de la HoRA no debe ser una limitación para el sistema, ya que se trata de un servidor cuya tarea principal es el alamcenamiento y gestión de la base de datos. Otro aspecto es el problema planteado por la confidencialidad de los datos en ejecución. A pesar de que HoRA debe considerarse una entidad de confianza, en caso de revocación se le deben enviar los datos confidenciales. La dificultad que puede tener HoRA para realizar alguna escucha de dicha información depende en gran medida del mecanismo de detección utilizado.

Implementación de entornos virtualizados. La solución presentada en este capítulo puede resultar muy restrictiva. Especialmente el uso del TPM instalado en cada una de las máquinas en las que tenemos agencias de un sistema puede resultar algo estricto. Aunque viendo los avances del hardware en los últimos años, podemos observar que los nuevos PCs domésticos y los nuevos portátiles vienen cada vez con un mayor equipamiento. De hecho, una de las tendencias actuales aboga por incluir el TPM dentro de este equipamiento. Sin embargo, lo que sí representa una restricción extremadamente estricta es el hecho de que todas las agencias han de tener la misma medida de PCRs. Lo cual se traduce en que cada uno de los elementos software instalados en todas las agencias ha de ser idéntico tanto en versión como en la secuencia de instalación del mismo. Para dar solución a este problema estamos estudiando una iniciativa basada en el uso de entornos virtualizados, dentro de los cuales podemos tener alojadas a las agencias. De forma que el tener medidas idénticas de PCRs para cada una de las agencias sea una restricción asequible para cualquier sistema.

Capítulo 4

Protección basada en Computación Protegida

En el capítulo 2 definimos el concepto de agente cooperativo, así como también mostramos que la autonomía y la cooperación con otros agentes son los aspectos más importantes y representativos de este tipo de agentes. Los agentes cooperativos se usan para resolver diversos problemas, pero en general, por el simple hecho de encontrarse distribuidos y de que la solución que implementan dependa de la ejecución correcta de esos agentes, podemos ver que los aspectos de seguridad son esenciales y presentan varios retos de interés en el panorama actual. En nuestro caso vamos a centrarnos en el problema de la seguridad del sistema global, considerando requisitos tanto por parte del agente como de la agencia. Como apuntábamos en el capítulo 2, Roth [25] presenta la primera idea de agentes cooperativos para un propósito en común. La idea se centra en asumir que en un sistema abierto como es Internet la probabilidad de que varias agencias estén confabuladas para actuar en contra de un agente es muy reducida, puesto que es complicado que entre ellas haya una relación de confianza. La propuesta radica en la compartición de secretos y decisiones entre varios agentes que actúan en cooperación, con la consideración de que una parte individual del secreto no proporciona información sobre el total. Precisamente esta es la base de la computación protegida, y sobre esos antecedentes basamos la solución que presentamos a continuación.

4.1. Introducción

El objetivo principal de este capítulo es el de describir el problema de la seguridad de los sistemas multiagentes (compuestos por agentes cooperativos), además de presentar una solución al mismo. En nuestro caso, proporcionamos una solución basada en la protección mutua entre todos los agentes del sistema, utilizando para ello ciertos elementos software que describimos en las siguientes líneas.

Nuestra propuesta consiste en forzar que cada agente del sistema actúe como protector de otro. Entrando en más detalle en el mecanismo de protección, podemos decir que cada agente se encarga de ejecutar parte del código de otro (agente al que protege). De esta forma, un ataque al sistema sólo sería efectivo mediante un ataque a todos los

agentes que componen el mismo, es decir, una confabulación entre todas las agencias que intervienen contra el sistema global. En la práctica, este tipo de ataques no se producen en un sistema real, ya que en la mayoría de los casos los intereses de cada agencia son diferentes (frecuentemente las agencias compiten entre sí) y por tanto carece de sentido que tales agencias se asocien para atacar al sistema.

La figura 4.1 ilustra la idea de la protección mutua entre agentes, en ella se muestra cómo cada agente protege a otro y es protegido a su vez por un tercer agente. La idea es que en este esquema subyace un grafo euleriano. Recordamos que la definición de grafo euleriano corresponde con un grafo dirigido en el cual se recorren todas y cada una de las aristas del grafo una única vez, existiendo un camino o ciclo euleriano, que a su vez se define como un ciclo que contiene todas las aristas de un grafo una única vez, es decir, cada uno de los agentes del sistema ha de proteger a otro agente y este ha de ser protegido, a su vez, por otro agente del mismo y distinto del anterior, a excepción de tratarse de un sistema con dos únicos agentes que evidentemente es un caso trivial.

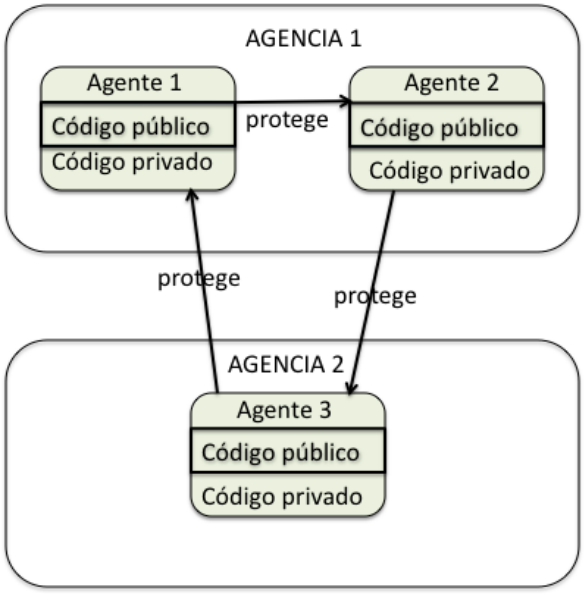

Figura 4.1: Esquema de protección mutua entre agentes colaborantes.

Así el esquema de protección que se establece entre tres agentes se realiza de tal forma que el agente1 es protegido por el agente3, y a su vez, el agente1 protege al agente2. Así ocurre sucesivamente con los demás agentes del sistema, siendo este esquema escalable a cualquier número de agentes para cualquier sistema sin que ello implique una mayor complejidad. Destacamos la escalabilidad de este modelo, puesto que nuestro sistema funcionaría de forma correcta sin depender del número de agentes del sistema. Toda la comunicación se realiza mediante el paso de mensajes por la red.

Obviamente, el hecho de que un agente proteja a otro y a su vez sea protegido por otro distinto conlleva una serie de consideraciones. En nuestra solución hemos hecho uso de la técnica conocida como "computación protegida". Esta técnica tiene como fundamento la división del código en dos o más partes, de forma que las partes son ejecutadas en procesadores distintos. Evidentemente, la división del código ha de realizarse con ciertos requisitos que describimos a continuación. La parte que consideremos como código privado, que se ejecutará en un procesador seguro (en el sentido de que no es accesible al atacante), que no está bajo el dominio del usuario, no puede ser derivada a partir del código que se ejecuta en la parte pública. Por tanto, el código etiquetado como público no podrá aportar información de la parte de código privado.

Otra característica de esta solución es que la ejecución total del código necesita la finalización de la ejecución de ambas partes. En nuestro caso particular, hemos aplicado esta técnica para dividir el código de los agentes móviles que componen nuestro sistema. Una vez aplicada la técnica obtendremos una nueva versión de los agentes que estarán compuestos por la parte pública de su propio código y por una parte privada de código perteneciente a otro agente al que protegen. De la misma forma, otro agente tendrá la parte privada de este agente, constituyéndose así como agente protector de este.

En la figura 4.2 se muestra la transformación que sufre el código mediante el uso de la computación protegida. Durante esta etapa el código se transforma con objeto de protegerlo frente a un posible ataque por ingeniería inversa. Hacemos especial énfasis en que la "computación protegida" no depende de los detalles específicos de esta etapa, en el sentido de que cualquier esquema que ofrezca como resultado una aplicación con la estructura mostrada en la figura 4.2, que sea equivalente funcionalmente a la original y con una serie de secciones traducidas al código ejecutable por el procesador confiable, podría ser usado aplicando esta técnica. De hecho, esta técnica se diseñó con independencia del tipo de procesador seguro usado. Por lo que el esquema es equivalente tanto si hacemos uso de un TPM, de una tarjeta inteligente de cualquier tipo, e incluso, como ocurre en nuestro caso, si hacemos uso de los propios agentes software.

Continuamos con la descripción de esta técnica. La etapa de transformación se realiza una única vez para cada agente protegido. Sin pérdida de generalidad, y con objeto de facilitar la descripción del proceso, consideraremos que un código de un agente se compone de una serie de secciones, que incluyen tanto instrucciones como datos.

Profundizando en la etapa de transformación. El primer paso de esta fase consiste en la identificación y selección de ciertas secciones del software original que se van a proteger. En esta fase se identifican las secciones protegidas que son las más adecuadas para ser ejecutadas en el procesador seguro. Esta identificación se realiza en función de varios criterios, como son el tipo de instrucciones que contiene, la entropía que poseen, etc. A continuación, en el esquema original, las secciones seleccionadas son traducidas al lenguaje ejecutable por el procesador seguro. Sin embargo, en nuestro caso esto no es necesario ya que todas las agencias son iguales, aunque es necesario un proceso de carga adaptativo debido a que el código se carga directamente cifrado en el agente. Durante el proceso de traducción también se realizan transformaciones de ofuscación. Estas son variantes de algunas de las transformaciones de ofuscación más relevantes. Entre las que destacamos:

- Identificación de dependencias entre las secciones protegidas. Estas dependencias se

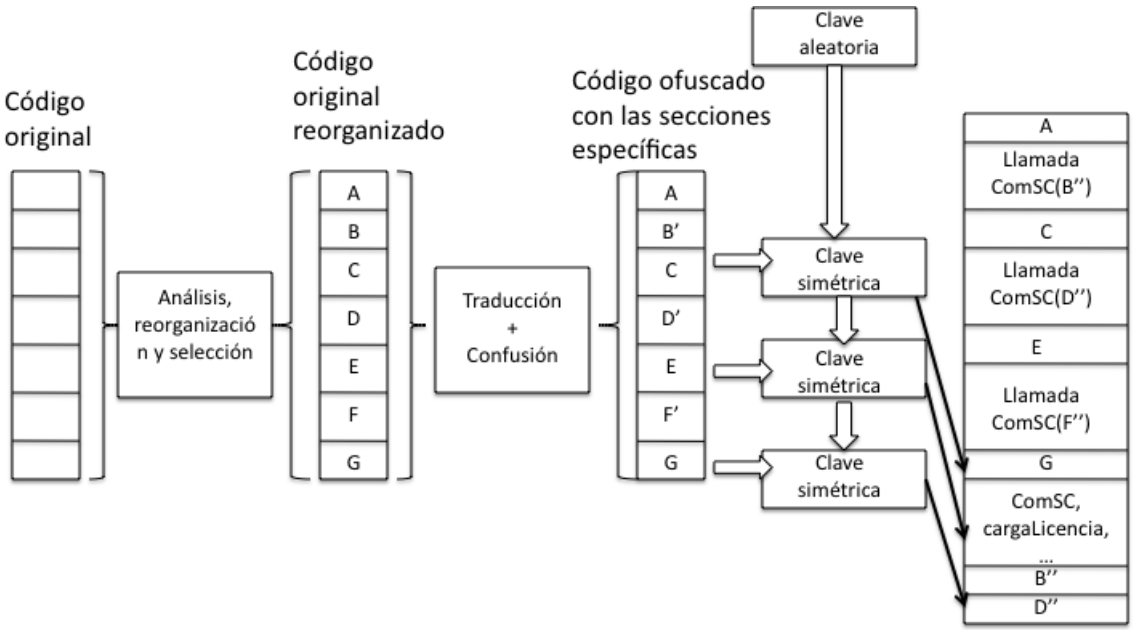

Figura 4.2: Transformación código en computación protegida.

utilizan para determinar qué resultados parciales, dentro de una sección se pueden mantener en el procesador seguro para ser utilizados por la siguiente sección.

- La reorganización del código es muchas veces necesaria para conseguir secciones más robustas o para evitar sobrecargar la ejecución en el procesador seguro.

- La introducción de código falso. Con objeto de confundir al atacante, se añaden datos e instrucciones que no son útiles para la aplicación real. Algunos de estos datos se procesan en el mismo procesador confiable produciendo, a su vez resultados innecesarios para la aplicación original, desde el punto de vista funcional, pero que contribuyen a su seguridad.

Una vez realizados los procesos de ofuscación y traducción, las secciones a proteger están preparadas para ser cifradas. Para ello utilizamos un criptosistema simétrico, que usa una clave generada de forma aleatoria que debe ser almacenada de forma segura por el productor de software. En el esquema original esta clave será utilizada con posterioridad para producir la licencia durante la fase de autorización. Es importante mencionar el hecho de que esto no será necesario en nuestro esquema. Las secciones protegidas se añaden al código original del agente como datos. Como veremos más adelante, es posible incluso prescindir del cifrado en algunas de nuestras soluciones, simplificando la aplicación de la metodología y obteniendo excelentes resultados.

El último paso de esta fase consiste en la sustitución del código de las secciones que han sido protegidas por llamadas a una función que se encarga de solicitar la ejecución de la correspondiente sección en el agente protector. En la computación protegida, el software resultante del proceso de protección realizado en esta fase puede ser distribuido y copiado libremente sin necesidad de ninguna medida adicional de protección. Esta característica

nos ha servido como inspiración en el uso de esta técnica para la protección de agentes móviles, puesto que estos viajan de agencia en agencia quedando expuestos a sus ataques.

Sin embargo, como antes mencionamos, para usar esta técnica es imprescindible disponer de un procesador seguro. En nuestro caso este papel de procesador seguro lo desempeñará otro agente del sistema el que actúe como tal.

En el estudio de esta tesis se ha determinado una clara diferencia entre dos soluciones distintas. La primera de ellas, es un método que denominamos de protección mutua estática. En este esquema se asume que el número de agentes en el sistema es fijo y está preestablecido antes de la ejecución del mismo. En el caso de que algún agente abandonase el sistema, se abortaría su ejecución. Por tanto, los agentes conocen tanto a sus agentes protectores como sus respectivos códigos de ejecución en tiempo de compilación. Este hecho limita la adaptabilidad práctica del sistema resultante. La adición o eliminación de un agente provocaría la necesidad de cambiar el código de toda la aplicación. Este método se ha desarrollado completamente y es el núcleo de este capítulo y a pesar de sus restricciones inherentes existen escenarios en los que obtenemos excelentes resultados en su aplicación.

El segundo método es una evolución del primero, con las mejoras suficientes para aportar un alto grado de dinamismo y más acorde con las necesidades reales. Lo hemos denominado método de protección mutua dinámica. Esta solución es más compleja que la anterior, pero proporciona más escalabilidad y cierta capacidad de reacción dinámica. El grafo de protección entre agentes se puede modificar en tiempo de ejecución, de la misma forma que también se puede modificar el código que ejecutan los mismos. Debido a las restricciones de tiempo fundamentalmente, hay que establecer ciertos límites en la investigación, por lo que hemos decidido desarrollar esta estrategia únicamente a nivel teórico, dejando su implementación como trabajos futuros de esta tesis. En definitiva, hemos diseñado el método y los protocolos de protección dejando como tarea futura la implementación de la misma.

En la implementación de estos mecanismos hemos utilizando la plataforma de agentes JADE, principalmente por el hecho de que esta plataforma sigue el estándar FIPA (véase la sección 2.3.4), además de ser la más extendida en el momento actual en la comunidad de programadores de sistemas de agentes.

En la figura 4.3 se muestra el proceso completo de la generación de agentes protegidos usando la metodología propuesta, a la que denominamos de forma genérica "técnica de protección mutua". En ella se aprecia cómo aparecen los dos esquemas descritos. En la parte superior aparece un cuadro que engloba el esquema de protección estática y en la inferior encontramos otro cuadro con el esquema de protección dinámica. También se muestra el caso de uso en que un implementador programe un sistema de agentes sin ningún mécanismo de seguridad.

El usuario (en este caso el desarrollador del sistema) hace uso de la herramienta de protección de código "code partition tool"(CPT), junto con un "perfil de protección". Como se explica a continuación esto permite una mayor flexibilidad en el proceso completo. Ambas herramientas se han diseñado e implementado como parte de esta tesis doctoral. En las secciones siguientes encontramos una descripción de los aspectos más relevantes de las mismas. De hecho, el propósito de estas herramientas es exclusivamente el apoyo en la tarea de partición del código para los usuarios, de forma que estos no necesiten un

conocimiento experto en el área de la seguridad.

Mediante la configuración del perfil de protección los usuarios pueden indicar diversos parámetros, tales como el porcentaje de instrucciones y/o datos a proteger. Además este perfil permite seleccionar alguno de estos parámetros en concreto por considerarlos de interés o cualquier otra razón. Puesto que cada sistema tiene unas características concretas, los agentes han de protegerse en concordancia con las mismas y no de forma general, puesto que ello podría provocar la aparición de serios problemas de seguridad.

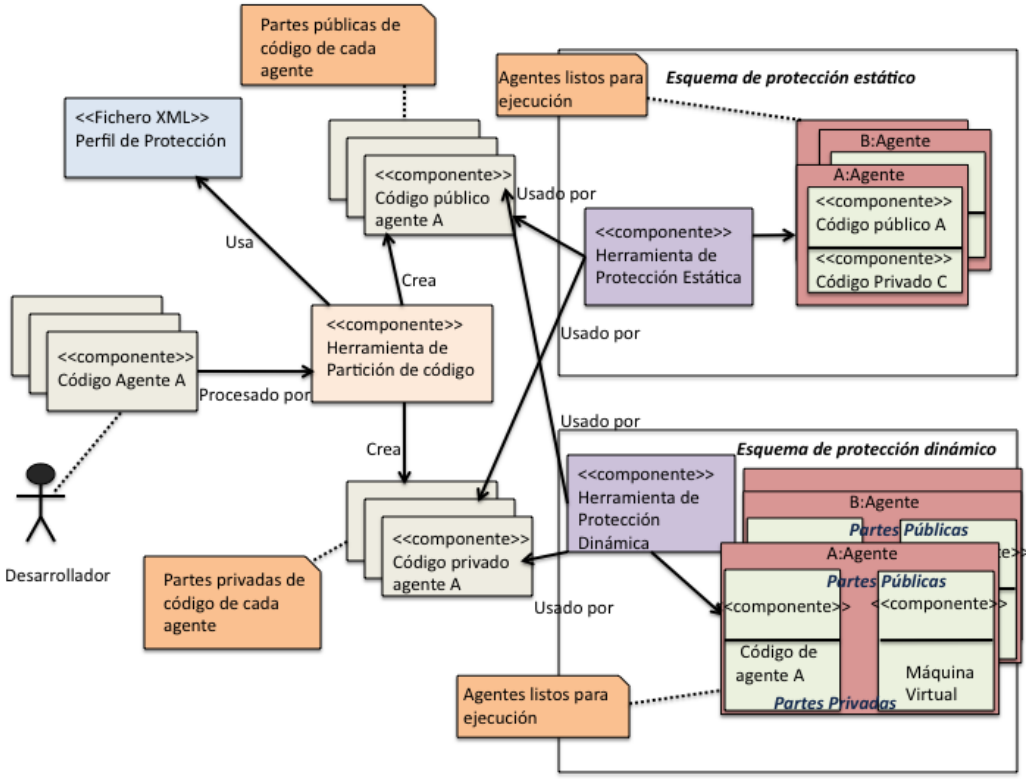

Figura 4.3: Protección de agentes usando la técnica de "protección mutua".

Como hemos comentado hemos definimos dos estrategias claramente diferenciadas, la primera de ellas la hemos denominado como "protección mutua estática" y por otro lado tenemos la estrategia de "protección mutua dinámica", que permite que cualquiera de los agentes que colabore en el sistema pueda ejercer como procesador seguro para los otros agentes. Por lo tanto, en este último caso la interacción de los agentes no está predeterminada y ello conlleva una dificultad añadida.

Protección mutua estática. En este esquema de protección tendremos una sociedad de agentes colaborando entre ellos para proporcionar seguridad al sistema. Los agentes actuarán como procesadores seguros unos de otros. La figura 4.4 nos muestra de forma esquemática esta estrategia.

En la figura 4.4 vemos que cada agente protege a otro agente del sistema y a su vez es protegido por otro distinto. También se muestra que cada uno de los agentes se ejecuta en una agencia independiente. Aunque evidentemente, varios agentes podrían ejecutarse

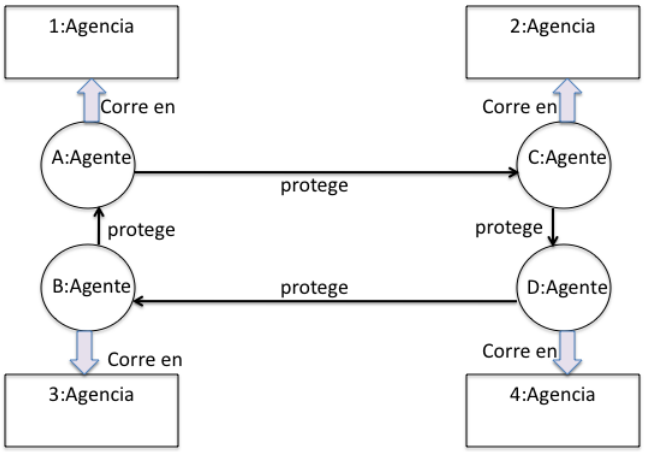

Figura 4.4: Protección mutua estática de una sociedad de agentes.

de forma simultánea en la misma agencia sin alterar el esquema de protección. Como hemos mencionado, esta estrategia nos permite que la parte protegida de un agente esté directamente incluida en otro agente. Es importante mencionar que sólo es aplicable en los casos en los que el conjunto de agentes a proteger sea estático y pueda determinarse en número de agentes del sistema antes de su ejecución. Con los calificativos estático y predeterminado nos referimos a que tanto el número de agentes del sistema como la selección de los mismos y la repartición del código se establece a priori, antes de que el sistema comience su ejecución.

Protección mutua dinámica. Existen casos en los que la protección mutua estática tiene un abanico de aplicación bastante reducido, ya que fue diseñada e implementada como un prototipo meramente académico. Sin embargo, para poder explotar esta idea en sistemas reales de agentes se desarrolló la estrategia de protección mutua dinámica. En esta alternativa cada agente es capaz de ejecutar arbitrariamente secciones de código de

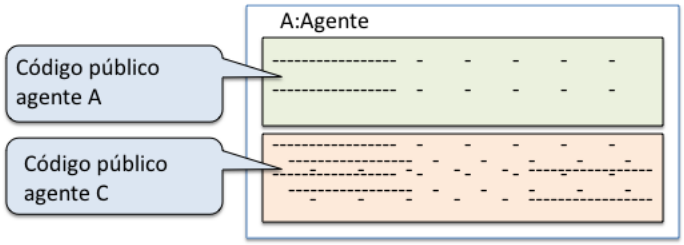

Figura 4.5: Partición del código.

otros agentes de la sociedad. Cada agente incluye una parte pública, una parte privada, que permanece cifrada, y una máquina virtual especifica. La máquina virtual permite a los agentes ejecutar secciones de código (en concreto nos referimos a las partes privadas) recibidas por otros agentes al instante.

4.1.1. Esquema de protección estático

Hasta ahora hemos descrito la base conceptual de ambos esquemas de protección. Aunque la puesta en marcha de estos mecanismos no resulta trivial. Llevar esto a la práctica es bastante más complejo de lo que se pueda presuponer. La elaboración de un sistema completo de agentes con un alto número de agentes en el mismo puede tener una complejidad muy elevada, más aún si el desarrollador se tuviese que preocupar de seleccionar las partes del código que se van a tratar como públicas y cuales como privadas. Y si además hay que hacer las llamadas oportunas para ejecutarlas en cada momento en concreto, se puede llegar a formar una amalgama difícilmente controlable de código. Para ello nos hemos planteado el diseño de herramientas de soporte para este tipo de tareas, con idea de facilitar la implementación de ambos esquemas de protección mutua entre agentes abstrayendo al desarrollador de las tareas de separación y repartición del código, como introducíamos anteriormente.

La elaboración de estas herramientas es bastante más compleja de lo que se prevee. Para el caso de la protección mutua estática nos encontramos con que hay que hacer un chequeo del código a nivel de bytecodes, para lo cual hay que realizar un análisis sintáctico y semántico del contenido tras una interpretación del mismo. Se construye un árbol jerárquico sobre el que se va a trabajar. Sin embargo, en el caso de la protección mutua dinámica es considerablemente más complejo, puesto que las partes de código etiquetadas como privadas se envían entre agentes en tiempo de ejecución. Hemos adoptado una solución basada en la integración de un tipo de máquina virtual en cada uno de los agentes, capaz de interpretar ciertas instrucciones de código. La librería desarrollada para implementar la protección estática nos ofrece las siguientes funcionalidades:

- Cada agente será protegido por otro del sistema.
- Cada agente protegerá a un agente del sistema.

Ambas funcionalidades de forma paralela sirven para configurar un sistema usando la protección en anillo descrita, formando un ciclo euleriano en el que no habrá ningún agente sin protección. La protección se establece en tiempo de compilación. Además estas no pueden ser modificadas durante el tiempo de ejecución del sistema. Únicamente se permitirán cambios en la ubicación de los agentes, protector y protegido, ya que estos migran para continuar su ejecución y conseguir sus objetivos. Cada agente esta formado por una parte de código propio que será público y una parte de código privado que pertenecerá al agente. Estas partes se definen en tiempo de compilación, por lo que, se determinarán las partes del código de cada agente que son críticas para la seguridad del mismo y se distribuyen entre los correspondientes agentes protectores. Esto permite que los agentes puedan recibir peticiones de ejecución remota de código por parte de sus agentes protegidos correspondientes. En respuesta a la petición anterior, el agente será

capaz de confirmar al agente solicitante la disponibilidad para la ejecución y proceder a la misma.

Cada agente tiene la capacidad de enviar los argumentos con los que quiere que se ejecute su código privado, en caso de no necesitar argumentos para su ejecución, se enviará un mensaje "null". Este mecanismo permite a los agentes notificar, tanto al agente que lo protege como al agente que protege su nueva ubicación que es esencial para poder establecer, en cualquier momento, la comunicación con ellos. Todo esto permitirá al agente protegido recibir el resultado de la ejecución realizada por el agente protector.

El agente seguro se modela con el uso de la clase SecureAgent, clase que hereda de Agent. Esto nos permite diseñar el sistema construyendo un tipo de agente particular con las funciones de seguridad apropiadas. Para construir un sistema multiagente protegido haciendo uso de la metodología de protección mutua estática, basta con que cada uno de los agentes sean instancias de la clase SecureAgent en lugar de la estándar Agent.

Los agentes se programan mediante el uso de comportamientos. Estos son los encargados de realizar las tareas correspondientes para conseguir los objetivos propuestos de los agentes. Para utilizar nuestra librería no es necesario conocer la función de cada uno de los comportamientos. En cambio, sí que es imprescindible realizar ciertas aclaraciones en el uso tanto de la clase SecureAgent como de la clase PrivateCode. La primera de ellas representa al agente seguro y realiza una especialización del agente genérico de Jade. Esta clase proporciona como funcionalidad principal el método "setup", que es imprescindible para cualquier implementación de agente. Este método se invoca automáticamente por el propio sistema en el momento de carga del agente en el sistema. En esta sección se definen las acciones que puede realizar el agente durante su ejecución. A pesar de que este método es implementado por el usuario de la clase, se deben seguir unas pautas y acciones necesarias en la implementación haciendo uso de nuestro esquema.

```
protected  void setup() {

      /* Establecer las protecciones del agente
       * tanto el protector como el protegido
       */
      /* Llamada al método inicializar del padre */
      super.inicializar(this);
      /* Crear una instancia del código privado del
      agente al que protege */
      this.privc = new PrivateCodeB();
      …
}
```

Figura 4.6: Método setup.

Para realizar una llamada de ejecución remota de código, es decir, ejecución del código protegido se han de usar las instrucciones que aparecen en la figura 4.7.

Para la creación y configuración de los mensajes ACL se hará uso de las tres primeras instrucciones. La cuarta instrucción sirve para asignar al campo myArgs los argumentos que pretendemos enviar al agente protector. Y por último, en la quinta instrucción damos comienzo a la comunicación. El proceso de comunicación entre dos agentes queda perfectamente descrito en el protocolo de la figura 4.9.

```
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
msg.setContent("execute-my-private-code");
msg.addReceiver(this.protectedBy);
myArgs = argumentos;
send(msg);
```

Figura 4.7: Llamada ejecución remota.

Consideramos de especial interés la interfaz PrivateCode, con la particularidad de que únicamente se define un método en la misma, de forma que el fragmento de código privado de cada agente irá en el método "*execute()*" de la clase específica que implemente la interfaz PrivateCode. El agente no necesita conocer cómo funciona el código para poder ejecutarlo únicamente tiene que saber que llamando al método "*execute()*", con los parámetros correspondientes, se producirá la ejecución. Otra condición necesaria de esta clase PrivateCode, es que esta ha de implementar la interfaz *Serializable*. Para ello, todos los componentes de la clase deben serlo a su vez. Tanto los argumentos como el objeto devuelto por "*execute()*" son de tipo *Object*, con idea de aportar el mayor grado posible de genéricidad. El usuario que implemente la interfaz determinará los tipos cada uno de los objetos para su sistema concreto.

```
public  class MyPrivateCode implements PrivateCode {
   public Object execute(Object o){
      /* Código protegido */
      return null;
   }
}
```

Figura 4.8: Clase que implementa PrivateCode.

4.2. Resultados obtenidos

En esta sección vamos a hacer un repaso de los resultados más significativos de la aplicación de esta metodología. Para proporcionar una perspectiva real haremos uso de un escenario de uso en el que hemos aplicado nuestra técnica. Una vez presentado el escenario hacemos una descripción del conjunto de mecanismos que hemos utilizado para la implementación de las herramientas de apoyo para la aplicación de nuestra estrategia de protección de agentes. Por último hacemos un repaso de las herramientas resultantes.

El escenario está situado en un centro comercial, en el mismo tenemos a un usuario que quiere consultar rápidamente en qué tienda puede encontrar un determinado artículo con unas características particulares (precio, color, talla). Se plantea modelar un servicio que permita a los usuarios hacer este tipo de consultas desde sus dispositivos móviles tanto sus PDA's, sus teléfonos móviles, etc.

El centro comercial contará con una agencia central que procesará las peticiones de los usuarios y repartirá agentes por las tiendas para recopilar la información que entregará

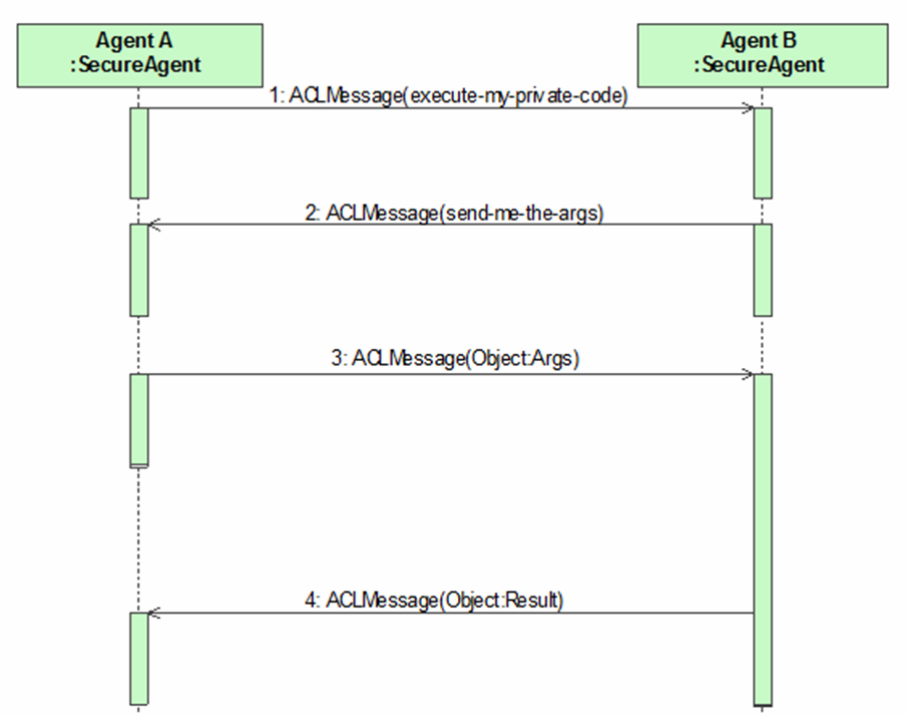

Figura 4.9: Intercambio de mensajes para ejecución segura.

de nuevo al cliente. El uso de agentes móviles en un sistema multiagente conllevan ciertos riesgos de seguridad, especialmente cuando estos visitan las diferentes agencias migrando entre ellas. En el escenario propuesto podría ocurrir que una agencia, con carácter malicioso, pretenda manipular al agente móvil encargado de recoger la información del producto, con el objetivo de que este modificase los resultados que se van a entregar al usuario una vez haya alcanzado la agencia central. Por lo que la protección del agente es un requisito evidente.

Planteamos un esquema de seguridad basado en el esquema de protección mutua estática entre agentes. Haciendo práctica de esta metodología, cada agente es el encargado de ejecutar parte del código de otro agente del sistema (agente al que protege). Partimos de la base de que una confabulación entre todas las agencias del sistema podría romper el sistema. Evidentemente este ataque no tiene sentido y por lo tanto no lo consideramos de interés. En cualquier estudio de la seguridad de un sistema es importante hacer un balance entre los posibles beneficios que puede obtener el atacante y los esfuerzos invertidos para perpetrar ese ataque, aunque existen otras razones como las motivaciones personales difícilmente cuantificables.

Hemos aplicado esta estrategia para la protección del sistema ya que este tipo de sistemas en los que el número de agentes en el mismo es estático se adapta perfectamente al modelo de nuestra estrategia. Para facilitar la aplicación de esta estrategia hemos creado una herramienta de apoyo capaz de generar un sistema de agentes móviles con la aplicaión de la estrategia de protección mutua estática partiendo de un sistema de agentes sin protección mediante un interfaz amigable. Esta herramienta permite leer los archivos precompilados de los diferentes agentes, en formato bytecode, y generar, de acuerdo a una

serie de parámetros de configuración, el código de los agentes protegidos.

Recordamos que la protección mutua consiste en la división del código de un agente en partes pública y privada. Así, la parte pública será ejecutada por el propio agente, mientras que la parte privada la ejecutará el agente protector. Cada vez que se menciona "código de un agente" nos referimos tanto a instrucciones como a datos. En el proceso de configuración dos de los parámetros que tendremos que precisar son el porcentaje de instrucciones y el de datos a proteger, por considerarlos de especial interés. Es necesario considerar que estos parámetros influyen en que el tiempo de ejecución de los nuevos agentes y en el número de solicitudes de ejecución de código privado que los agentes envían. Esta herramienta de apoyo nos permitirá, mediante una configuración sencilla, realizar las comprobaciones necesarias de forma que de forma eficaz se podrá localizar cuál es la configuración más adecuada para cada sistema concreto en consonancia con sus requisitos individuales.

4.2.1. BCEL: byte code engineering library

Anteriormente comentábamos que nuestro objetivo final era crear una aplicación software capaz de generar de forma automática un sistema multiagente protegido mediante el esquema de protección mutua estática. A lo largo de esta tesis hemos presentado cada una de las tecnologías de las que hemos hecho uso para poder completar con éxito nuestro objetivo. JADE como plataforma para la gestión de sistemas multiagentes y la protección mutua estática como esquema de seguridad.

Pero para llevar a cabo esta transformación necesitamos una herramienta para escudriñar y analizar los bytecodes [60] de los respectivos agentes sin protección y hacer las modificaciones pertinentes para realizar la protección adecuada. Existen varias API's disponibles que nos facilitan el trabajo para analizar el código precompilado de Java (como ASM [61] y BCEL [62]).

Hemos elegido BCEL por ser la más extendida entre la comunidad de programadores. Otra razón interesante es que proporciona un interfaz de uso considerablemente más sencilla que sus alternativas, lo cual abarca un conjunto mayor de posibles usuarios. Vamos a describir brevemente en qué consiste esta librería. BCEL es el acrónimo de Byte Code Engineering Library. Consiste en una API que proporciona al usuario la posibilidad de analizar, crear y manipular ficheros precompilados de Java (.class). Las clases se representan mediante objetos que contienen toda la información simbólica que posee una clase: métodos, campos, instrucciones, etc. La API básicamente consta de tres paquetes.

1. Un paquete que contiene clases que describen las limitaciones "estáticas" de los ficheros de clase. Las clases deben ser utilizadas para leer y escribir ficheros de clase desde o hacia un archivo. Esto es especialmente útil para analizar clases Java sin tener el código fuente a mano. La clase principal de este paquete es JavaClass.

2. Un paquete para generar o modificar dinámicamente ficheros de clase que puede ser utilizado para añadir o analizar código de los ficheros de clase, etc.

3. Además de estos dos paquetes, BCEL contiene varios ejemplos de código y utilidades como un visor de ficheros de clase, una herramienta para convertir fichero de clase

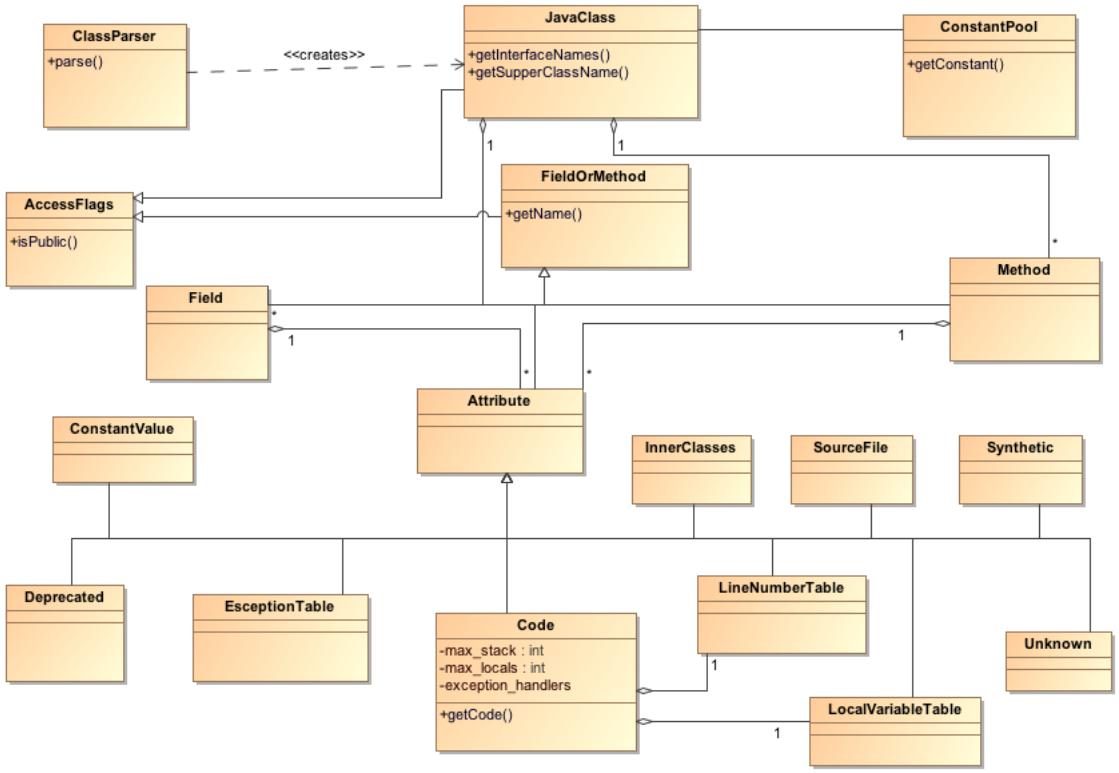

Figura 4.10: Diagrama UML de las clases de JavaClass.

a HTML y al lenguaje ensamblador Jasmin que pueden resultar de utilidad.

4.2.2. Componente estático de la API de BCEL

JavaClass es el componente estático de la API. Este componente lo podemos encontrar en el paquete org.apache.bcel.classfile y representa ficheros de clase. La figura 4.10 muestra un diagrama UML de este componente estático JavaClass.

En el nivel superior de la jerarquía encontramos la clase *JavaClass*, que en la mayoría de los casos es creada por un objeto *ClassParse*. Simplificando podemos decir que un objeto *JavaClass* consta básicamente de campos, métodos, referencias simbólicas a la clase superior (super) y las interfaces implementadas.

La clase *ConstantPool* sirve como un repositorio central y es de especial importancia para todos los componentes. Esta clase contiene una lista de constantes (Constants) accesibles mediante la utilización del método *getConstant()* que toma un índice entero como argumento. Estos índices pueden estar contenidos en instrucciones así como en otros componentes de un fichero de clase y en otras constantes de la ConstantPool. Los métodos y campos contienen una signatura, que simbólicamente define sus tipos. Los indicadores de acceso (*access_flags*) como *public*, *static*... aparecen en varios lugares y se codifican con una máscara entera de bits. Podemos ver también en la figura como las clases que representan los atributos son específicas para alguna estructura de datos y en ellas están marcados los componentes a los que pertenecen.

La clase *Repository* nos proporciona funciones para facilitarnos la lectura de ficheros de clase y cargarlos en clases *JavaClass* con una simple llamada:

JavaClass clase = Repository.lookupClass('miClase');

El acceso a los datos del fichero de clase se realiza como se detalla a continuación. La información de los componentes de los ficheros de clase se puede acceder *Java Beans* mediante intuitivos métodos *set/get*. Además todos definen un método *toString()* que permiten implementar un visor de clases de manera muy sencilla.

El análisis de los datos de la clase se realiza usando un componente bastante útil de BCEL, este soporta el patrón de diseño *Visitor*. Este patrón nos permite recorrer de una forma elegante toda la jerarquía de clases y analizar, en cada caso, el componente en cuestión.

La clase *ClassGen* es una parte de la API (paquete *org.apache.bcel.generic*) que ofrece un nivel de abstracción para la creación y/o transformación de ficheros de clase de forma dinámica. La clase *ConstantPool* está implementada mediante la clase ConstantPoolGen, esta ofrece métodos para añadir diferentes tipos de constantes. Por tanto, ClassGen ofrece una interfaz para añadir métodos, campos y atributos a nuestras clases.

4.2.3. Generación automática de agentes protegidos

El propósito principal de esta sección es explicar de forma detallada el proceso completo de la generación automática de agentes protegidos. Partimos de un sistema multiagente perfectamente configurado para trabajar en una plataforma *JADE*. Cada uno de los agentes del sistema está contenido en un fichero de clase *(.class)*. Otra opción para aplicar esta técnica es la aplicación directa, es decir, realizando manualmente la selección de las partes de código público y privado y la repartición del mismo, ya que es posible analizar cada uno de los agentes e ir aplicando la protección que nos ofrece *SecureAgent*. No obstante, esto es un trabajo bastante tedioso y difícil en muchos casos, especialmente en aquellos en los que contemos con un sistema de entrada compuesto por un número muy elevado de agentes. Por ello, nuestro objetivo al crear la herramienta de generación de agentes protegidos es la de automatizar esta tarea. De forma que partiendo de un conjunto de agentes, nuestra herramienta es capaz de crear un sistema seguro de una manera rápida y cómoda.

Como resultado tenemos que mediante una sencilla interfaz gráfica de usuario (GUI) tenemos la posibilidad de cargar un indeterminado número de agentes JADE (.class) para aplicarles una configuración de seguridad, mediante el elemento denominado perfil de protección, y poder crear así un sistema multiagente protegido con la estrategia mutua estática.

Esta solución ofrece una serie de funcionalidades al usuario, entre las que destacamos la carga de ficheros de clase que representen a agentes JADE, ya que en definitiva los usuarios de nuestra solución van a ser usuarios de JADE en su mayor parte.

Para conseguir una solución robusta de utilidad hemos realizado un análisis de los requisitos funcionales básicos que debe satisfacer cualquier solución que se precie. Entre los posibles requisitos funcionales aparecen la posibilidad de acceder a la información de los

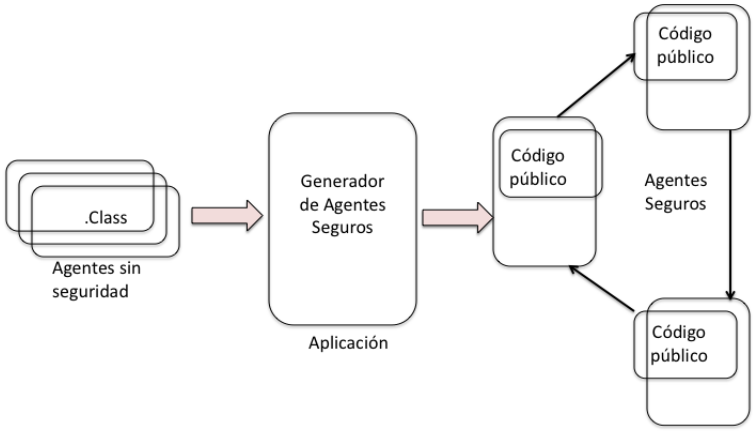

Figura 4.11: Esquema de comportamiento

agentes cargados en el sistema (entre esta información encontraremos toda la información de los agentes, es decir, los métodos, campos e instrucciones de los mismos). La protección de los campos de los agentes se realiza de forma individual (esto provoca que los campos seleccionados se introduzcan como parte del código privado del agente protegido). La selección del nivel de protección de las instrucciones del agente. La aplicación de perfiles de seguridad desde un archivo XML (este perfil se aplicará a todos los agentes cargados en el sistema), la configuración del sistema de protecciones en anillo, donde no habrá ningún agente seguro sin protección (este grafo de protección podrá configurarse a mano o automáticamente) y la selección del directorio donde guardar los nuevos agentes seguros, así como elegir los nombres para los nuevos agentes.

En lo que respecta al conjunto de requisitos no funcionales nos encontramos con que la interfaz gráfica ha de estar desacoplada del modelo de datos, en la medida de lo posible. La implementación de este modelo se ha realizado con la idea de la "facilidad de uso" en mente en todo momento. Obviamente se ha considerado que ciertos conocimientos básicos de seguridad son necesarios, pero no resulta imprescindible el hecho de ser un experto en esta materia para poder obtener un sistema seguro. Este hecho proporciona un grado que consideramos muy valioso para nuestra estrategia, ya que el abanico de posibles usuarios se abre considerablemente.

Una vez identificados los requisitos de nuestra herramienta nos planteamos que nuestra solución los cumpla, a pesar de que ello implique una serie de restricciones de partida. En primer lugar, es indispensable para su utilización tener instalada tanto la maquina virtual de Java como las librerías BCEL. Además de una plataforma JADE, ya que todo se realizará en la misma (aunque se podría usar cualquier otra con simples modificaciones).

A continuación mostramos un diagrama de casos de uso, en el cual se presentan los diferentes escenarios de éxito de estos casos de uso, satisfaciendo los requisitos mencionados anteriormente. En la ilustración 4.12 mostramos el diagrama de casos de uso general, en el que se puede apreciar cómo el sistema debe ofrecerle al usuario las cuatro opciones bá-

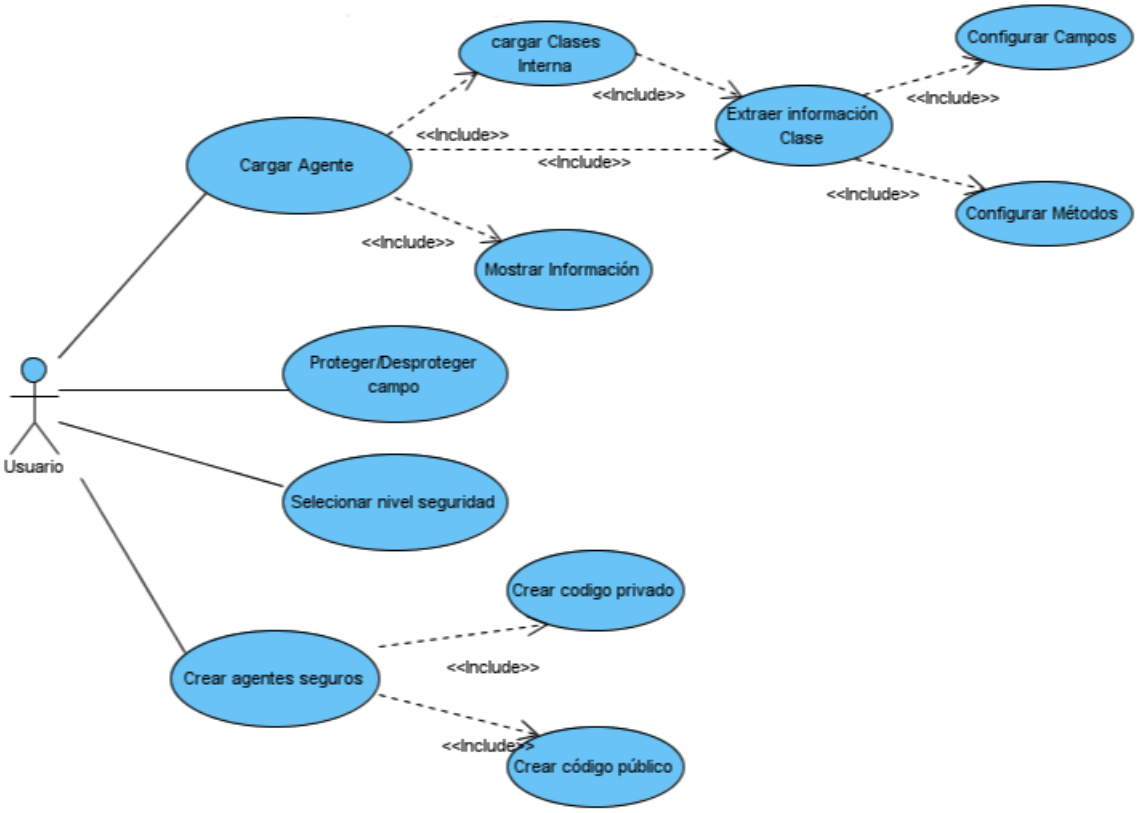

Figura 4.12: Diagrama de casos de uso

sicas, es decir, la carga del agente, la protección/desprotección de un campo, la selección del nivel de seguridad y la creación de agentes seguros.

En el caso de uso correspondiente a la carga del agente, el usuario carga un fichero de clase de un agente *JADE* con el fin de añadirlo al sistema multiagente que se desea configurar. El actor principal es el propio usuario de la aplicación. Los ficheros de clase estarán precompilados, además las clases han de heredar de *jade.core.Agent*. Los nombres de las clases no pueden contener el símbolo "$", ya que este es identificativo de las clases internas. Además el agente se une al sistema para su posterior configuración. El escenario de éxito principal viene descrito de forma que el usuario selecciona un archivo ".class" para cargarlo en el sistema. El sistema localiza los ficheros de clase de las clases internas y procede a su carga, a continuación se recoge la información de las clases internas.

El segundo caso de uso que hemos identificado es el que realiza la protección y/o desprotección de un campo. En el que el usuario selecciona uno de los campos del agente o de las clases internas para protegerlos o desprotegerlos. El actor principal es el usuario de la aplicación. El conjunto de precondiciones está formado por varias premisas, los campos estáticos no pueden ser protegidos y por lo tanto tampoco desprotegidos. Si el tipo de un campo es el de una clase interna, este campo no podrá ser protegido. En lo referente a las post-condiciones, el campo pasa del estado protegido al desprotegido o viceversa, lo cual indicará la localización final de este campo, si está protegido estará ubicado en la

parte privada del agente seguro y si no está protegido estará en la parte pública. Una vez descritas tanto las precondiciones como las postcondiciones vamos a presentar el escenario de éxito principal para este caso de uso. En primera instancia el usuario selecciona un campo a proteger/desproteger, en segundo lugar el campo modifica su estado y se cambia la configuración de seguridad del agente al que pertenece.

En el siguiente caso de uso se presenta la selección del nivel de seguridad. El proceso es el siguiente, el usuario selecciona un agente de los cargados en el sistema y establece su porcentaje de seguridad (el porcentaje de instrucciones y de datos que formarán parte del código privado). El actor principal es el usuario de la aplicación. La única precondición para este caso de uso es que el dato introducido deberá estar entre cero y cien. También presenta una única postcondición que consiste en que se establecerá el porcentaje indicado por el usuario en la configuración de seguridad del agente seleccionado. El escenario de éxito principal viene descrito por que el usuario seleccione un agente, previamente cargado en el sistema, establece su porcentaje de seguridad, en relación datos e instrucciones, y se guarda en la configuración de seguridad del agente seleccionado.

El último caso de uso es el de la creación de agentes protegidos. En este se comprueba que todos los agentes que componen el sistema están correctamente configurados y se procede a la creación de los nuevos agentes seguros. Al igual que ocurre en los casos de uso anteriores el actor principal es el usuario de la aplicación. Únicamente hay una precondición diferente que consiste en que el número de agentes cargados en el sistema debe ser mayor que 1, puesto que en caso contrario no tendría sentido el uso de esta técnica. Así como también sólo hay una postcondición que vendrá descrita por que habremos creado un sistema multiagente con el esquema de la seguridad mutua estática integrado. Para finalizar, el escenario de éxito principal viene descrito por que el usuario selecciona la acción "Crear agentes seguros", entonces el sistema comprueba que la configuración de seguridad de cada agente es correcta. Seguidamente, se crearán dos nuevas clases por cada uno de los agentes, una contendrá su código público y la otra su código privado. Se reparten los campos entre el código público y privado dependiendo de su configuración de seguridad, de forma similar con las instrucciones y se añaden las que sean necesarias para la comunicación. Y por último se guardan los nuevos agentes protegidos.

4.2.4. Diagrama de clases y comportamiento

Una vez vistos los casos de uso vamos a hacer una descripción de los aspectos más relevantes del diseño y de la implementación de esta solución. Como se describe anteriormente nuestro sistema consta de dos módulos (paquetes), en uno de ellos hemos introducido las clases relacionadas con la interfaz gráfica y en el otro el modelo de datos. De esta forma, conseguimos desacoplar los dos módulos de forma que si aplicamos algún tipo de cambio a cualquiera se convertirá en una tarea menos laboriosa. Respecto al modelo de datos, empezaremos describiendo su funcionamiento básico, que consta de tres etapas lineales que son la carga de agentes, la configuración de seguridad y por último la creación de agentes seguros, sin que haya posibilidad de solapamiento entre estas.

Vamos a mostrar el diagrama de clases por partes haciendo un modelado distinto para cada una de las partes de una etapa del funcionamiento. Es necesario cargar un agente JADE y extraer del fichero de clase toda la información necesaria.

Utilizamos envoltorios que modelan cada uno de los componentes principales de un fichero de clase, y se añade cierta información extra que es necesaria para los procesos posteriores. A continuación mostramos una breve descripción de cada una de estas clases:

1. ClassReady: Modela un fichero .class. Tendrá asociado una lista de métodos y de campos. Dos subclases heredan directamente de esta clase abstracta:

   AgentClass: Hereda de ClassReady y modela un fichero de clase concreto, el de un agente JADE.

   AgentInnerClass: Hereda de ClassReady y modela otro fichero de clase específico, el de una clase interna de un agente JADE.

2. FieldReady: Encapsula un campo de una clase. También cuenta con dos subclases:

   AgentFields: Representa un campo de un agente JADE.

   AgentInnerField: Representa un campo de una clase interna de un agente.

3. MethodReady: Representa un método de una clase. Es uno de los elementos que poseen más información ya que estos cuentan con una lista de instrucciones (InstructionReady).

4. IntructionReady: Modela a una instrucción, que a su vez, representa a la unidad mínima que puede ser protegida.

5. InformationVisitor: Hace uso del patrón de diseño Visitor para recolectar la información de los diferentes métodos de una clase.

   InformationAgentVisitor: Utilizado para los métodos de los agentes.

   InformationInnerVisitor: Utilizado para los métodos de las clases internas.

Para la creación de este modelo nos hemos basado en el modelo que ofrece BCEL, heredando algunas de nuestras clases directamente de sus clases para beneficiarnos de sus características. Veamos a continuación mediante un diagrama de secuencia como se produce la carga de una agente. Nótese que a pesar de no haber mencionado todavía las clases de la interfaz gráfica, éstas aparecen en el diagrama y serán descritas en mayor profundidad a lo largo de las siguientes líneas.

Podemos observar que a la hora de configurar las clases (tanto AgentClass como AgentInnerClass) aplazamos la configuración de los métodos hasta la última acción. Esto es debido a que, en el proceso de extracción de información de los métodos, es necesario tener perfectamente configurados los campos.

La Configuración de seguridad es la fase más simple del comportamiento de nuestra aplicación. En ella se distribuye el código de los diferentes agentes JADE cargados en el sistema para proporcionarles el nivel de seguridad que el usuario crea conveniente. Toda la información relativa a la configuración de seguridad se modela con una clase llamada SecureAgentConfiguration, esta clase es el puente entre la primera fase, la correspondiente a la carga de agentes y la tercera.

La clase SecureAgentConfiguration contiene tanto el porcentaje de seguridad de las instrucciones, como las listas con los campos que van a ser protegidos y los que no. El

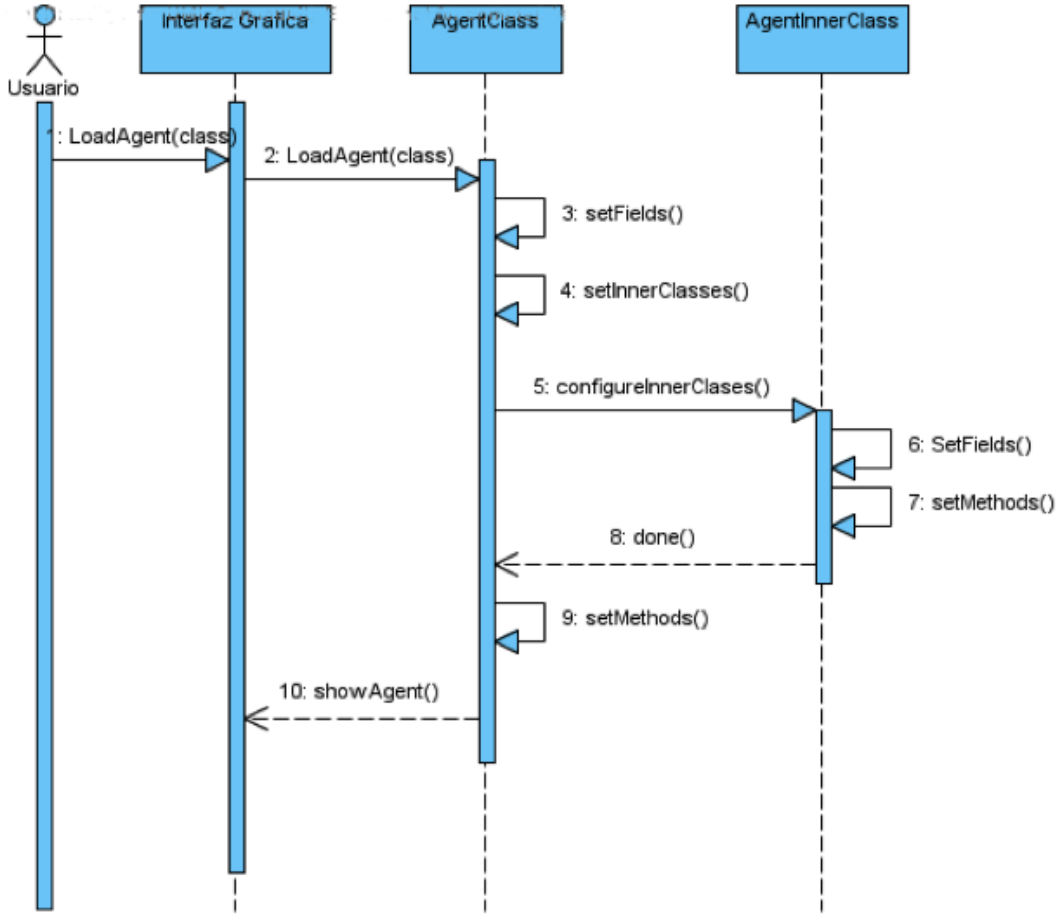

Figura 4.13: Diagrama de secuencia de carga de agentes.

uso de este elemento es relativamente sencillo, en la figura 4.14 mostramos los pasos que se realizan. En esta figura podemos ver cómo el usuario selecciona el nombre del campo a proteger, haciendo uso de la interfaz gráfica. Se envía el nombre del campo a proteger a la clase SecureAgentConfiguration. Esta clase obtiene el identificador de este campo y seguidamente invoca al método *protect* de la clase *FieldReady* y devuelve si la operación ha resultado exitosa.

Las acciones "desproteger un campo" y "seleccionar nivel de seguridad" tienen un comportamiento muy parecido al anterior. Esta clase también nos permite seleccionar el nivel de seguridad de un sistema de agentes a partir de un archivo XML. Esto nos proporciona la capacidad de dotar a todos los agentes del mismo nivel de seguridad, además de facilitarnos la tarea de protección de agentes por lotes, puesto que permite una automatización del proceso. Lo hemos etiquetado como "perfil de protección". En la figura 4.15 se muestra una versión reducida como ejemplo de perfil de protección. A pesar de que este ejemplo es bastante simple, podemos desarrollar un perfil de protección con multitud de clausulas específicas para cada caso concreto.

Una vez realizada la carga de los agentes en el sistema y la configuración de seguridad

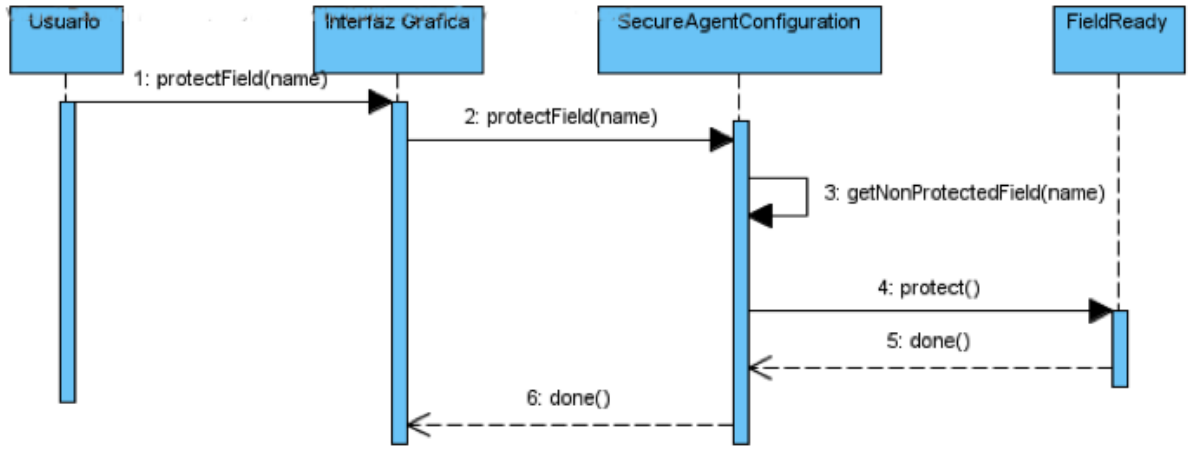

Figura 4.14: Diagrama de secuencia de la protección de un campo.

```
<?xml version="1.0" encoding="windows-1250"?>

<settings>
  <instructions>100</instructions>
  <data>100</data>
</settings>
```

Figura 4.15: Fichero XML congifguración de la seguridad.

procedemos a la protección de los agentes. Por cada agente cargado en el sistema debemos crear al menos dos ficheros de clase:

- El correspondiente al código público, una clase que herede de "SecureAgent".
- El correspondiente al código protegido, una clase que implemente la interfaz "PrivateCode".

A parte de estas dos clases y dependiendo de los agentes originales, crearemos un fichero de clase para cada una de las clases internas de los agentes originales. Con el objetivo de crear los ficheros de clase hemos diseñado el modelo que podemos ver en la figura 4.16. Aquí observamos varios elementos de unión con los diagramas anteriores:

- Tenemos la clase SecureAgentConfiguration que hemos utilizado para modelar la seguridad de un agente JADE.
- Además tenemos los envoltorios de los elementos básicos de un fichero de clase (ClassReady, MethodReady...) que utilizamos para obtener la información propia del agente al que queremos aplicar la seguridad.

Las clases más relevantes que vamos a comentar de este diagrama son:

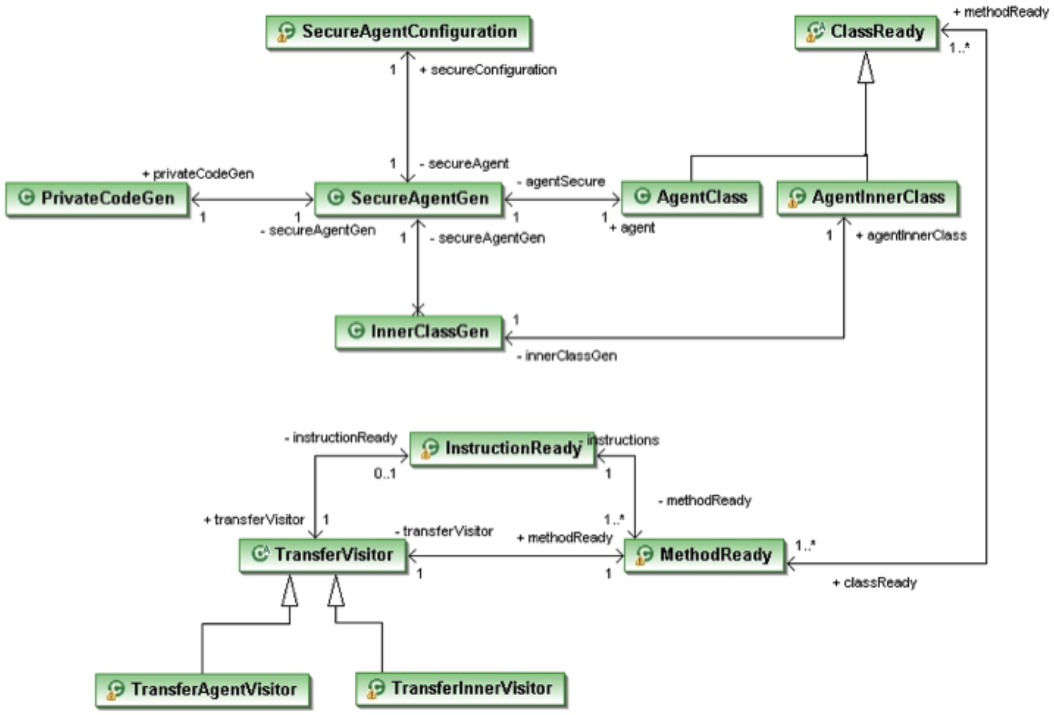

Figura 4.16: Diagrama de clases para crear agentes seguros.

1. SecureAgentGen: es la clase principal, usada para crear el nuevo agente protegido y está íntimamente ligada tanto a la configuración de seguridad como al agente original, del que consigue la información relativa a los métodos.

2. InnerClassGen: esta clase es la encargada de crear, a partir de la clase interna de un agente, una nueva clase interna al nuevo agente protegido.

3. PrivateCodeGen: es la dedicada a la creación del código protegido, es decir, crea un nuevo fichero de clase que implementa la interfaz PrivateCode. Este contiene tanto los campos protegidos como el conjunto de instrucciones protegidas.

4. TransferVisitor: del mismo modo que para extraer la información de los ficheros de clase utilizamos el patrón Visitor, ahora lo utilizamos para analizar las instrucciones y crear las nuevas clases adecuadamente.

El comportamiento de esta sección es más complejo que los anteriores. Una vez cargados y configurados los agentes en el sistema podemos crear los agentes ya protegidos. A simple vista podemos comprobar en la figura 4.17 que la clase "SecureAgentGen" es el elemento principal, es decir, el encargado de distribuir el trabajo y de crear el nuevo sistema de agentes con protección mutua. En el diagrama de secuencia se describen los pasos a seguir para la creación de agentes protegidos. El usuario selecciona la opción "crear agente seguro" de la interfaz gráfica. Esta funcionalidad abstrae del proceso interno al usuario.

La clase "SecureAgentGen" hace uso de la función de configuración "AgentClass", que a su vez llama a "configureMethods" con los valores de configuración del perfil de protección. Tras este paso se configura el "AgentTimeClass" de forma similar al anterior. Una

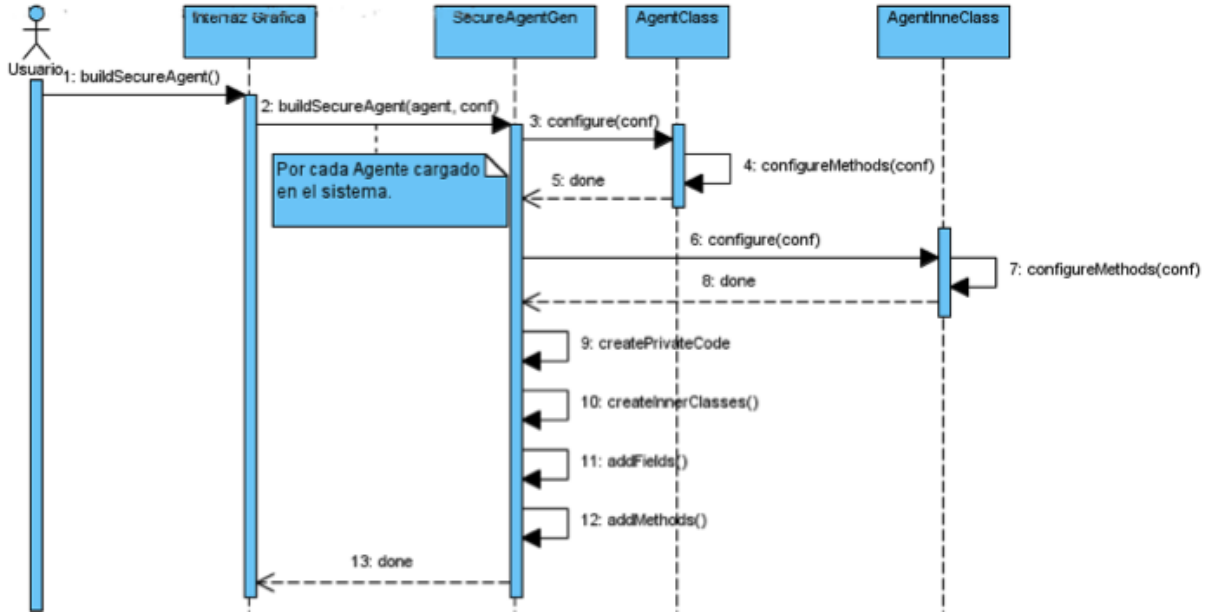

Figura 4.17: Diagrama de secuencia para crear agentes seguros.

vez finalizado este paso, se crea el código privado, haciendo uso de la función "createPrivateCode". A continuación se llama a "createinnerClasses", se añaden los campos, con la función "addFields" y por último se añaden los métodos "addFields" y se devuelve el mensaje que muestra si el proceso se completó satisfactoriamente.

Una vez visto todo lo referente a la especificación del modelo de datos, comentaremos la estructura de la interfaz gráfica, sin adentrarnos en los detalles de diseño que no consideramos de especial interés, únicamente resaltar el hecho de que se ha pretendido utilizar un diseño muy sencillo, con idea de minimizar el tiempo de aprendizaje en el uso de la herramienta.

La interfaz gráfica realiza la función de puente de conexión entre el usuario de nuestra aplicación y el conjunto de funciones que ofrece nuestro modelo de datos (formado por cargar, configurar y crear agentes seguros). Para lograr cierta sencillez, hemos utilizado el patrón de diseño Modelo-Vista-Controlador, donde el modelo corresponde al modelo de datos previamente presentado, la vista es la GUI y el controlador es el encargado de proporcionar la lógica de control para la interacción entre el modelo y la vista. Trasladando este patrón de diseño a nuestro modelo concreto obtenemos el diagrama de clases que se muestra en la parte derecha de la figura 4.18.

4.2.5. Implementación

El objetivo de este apartado es mostrar los aspectos más relevantes de la implementación de nuestra aplicación. Sin entrar en un nivel detallado de la misma, puesto que queda fuera del objetivo de esta tesis. Aunque consideramos importante la inclusión de ciertos aspectos derivados de la implementación que han enriquecido, mediante un proceso iterativo, los resultados de la investigación.

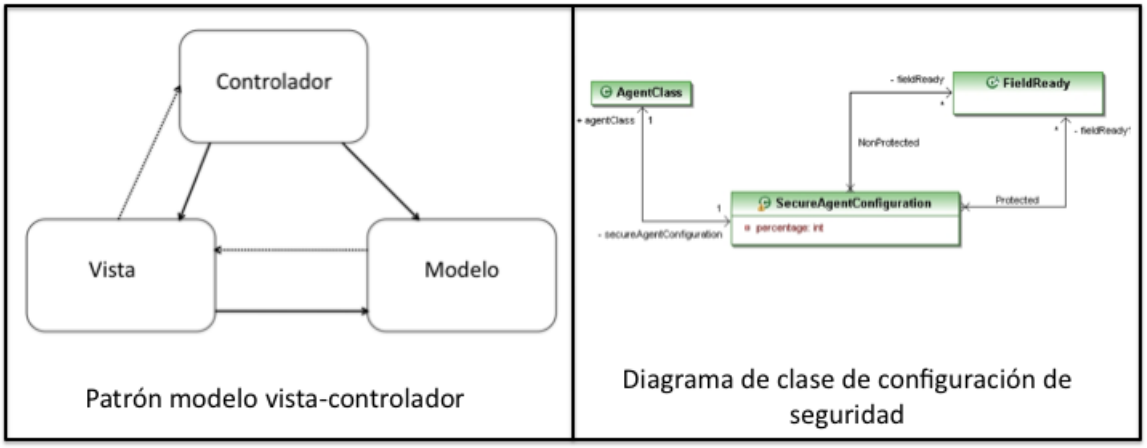

Figura 4.18: Patrón modelo vista-controlador y diagrama de clase de la aplicación de configuración de la seguridad.

De forma paralela al apartado anterior hemos divido esta sección de acuerdo a la implementación en las diferentes etapas de funcionamiento, dejando la interfaz gráfica para el final. En lo referente al modelo de datos, a pesar de que la implementación de las diferentes acciones (cargar, configurar, crear...) se ha realizado de forma paralela, vamos a describirlas por separado para conseguir una presentación más nítida al lector.

La carga de agentes se ha realizado usando la librería BCEL, que dada su relevancia en esta solución, se le ha dedicado una sección en este capítulo. Gracias al uso de esta librería se ha simplificado la carga de una clase desde un fichero, así como el acceso a esta información. En la clase "ClassReady" hemos introducido algunos métodos estáticos que nos faciliten la búsqueda y carga de ficheros de clase.

Las clases "ClassReady", "MethodReady" y "FieldReady" heredan directamente de clases de BCEL, obteniendo así, la mayoría de la información relativa a los ficheros de clase.

En este punto aún no disponemos de la información relativa a las instrucciones, ésta ha sido una de las tareas que ha requerido mayor esfuerzo. Exceptuando algunos casos particulares, las instrucciones en bytecode de forma independiente no realizan ninguna acción útil. Por ello, debíamos empaquetarlas en unidades mínimas de código que puedan ser protegidas.

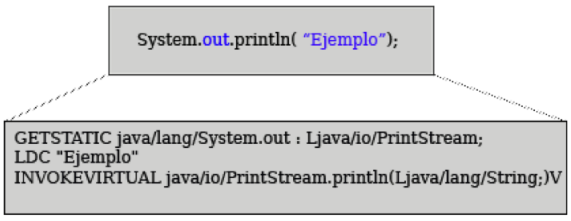

Figura 4.19: Código entre Java y bytecode.

La clase "InstructionReady" representa esta unidad mínima que puede ser protegida y corresponde a una instrucción Java terminada en ";". Para acotar el conjunto de instruc-

```
private void prepareNormalMethod(InformationVisitor visitor){
   // Cogemos las instrucciones del método
   InstructionHandle instructions[] = getInstructionList();
   // Inicializamos todas las variables a 0
   ...
   // Por cada instrucciones comprobamos si la pila esta valanceada
   for(int i = 0; i < instructions.length; i++){
      // Vemos lo que produce y consume la instrucción actual en la pila
      Instruction ins = instructions[i].getInstruction();
      consum = ins.consumeStack(cp);
      prod = ins.produceStack(cp);
      balance = (balance + prod) - consum;
      // Si la variable balance es igual o menor que 0
      if(balance <= 0 || (ins instanceof ATHROW)){
        InstructionReady newInstruction = new InstructionReady(startIndex,
                                                               i,
                                                               this,
                                                               visitor);
        // Añadimos la nueva instrucción al campo instructions
        this.instructions.add(newInstruction);
        balance = 0;
        startIndex = i + 1;
       }
   }
}
```

Figura 4.20: Algoritmo method ready.

ciones en bytecode que corresponde a un instrucción Java aplicamos el algoritmo de la figura 4.20.

La llamada a este método se realiza por medio de cada constructor del objeto MethodReady. Al finalizar el método el objeto MethodReady contendrá, en su campo instructions, una lista con todas los objetos InstructionReady que a su vez contendrán toda la información necesaria para la creación de los agentes seguros. Debido a las exigencias de nuestra solución, nos planteamos la forma de extraer la información de cada una de las instrucciones en bytecode. Podemos observar en los parámetros del constructor InstructionReady la existencia de uno en particular. El objeto visitor es el encargado, de obtener la información en función del tipo de instrucción bytecode. Este objeto visitor se comporta como una enorme estructura "if...else" y "if...else if..." que va escogiendo en cada caso el tipo de instrucción que queremos analizar. Este patrón de diseño nos permite recorrer y organizar la información fácilmente y de una manera muy elegante.

En lo que a la creación de los propios agentes seguros respecta, a diferencia del apartado anterior, aquí nos encontramos con que hay que tratar con la transferencia de instrucciones de unos métodos a otros. Hasta ahora hemos utilizado la funcionalidad estática de BCEL, con la que hemos podido captar toda la información de los ficheros de clase de los agentes originales. A partir de ahora vamos a hacer uso de la parte dinámica, donde crearemos las nuevas clases basándonos en las originales. La generación de agentes seguros sigue

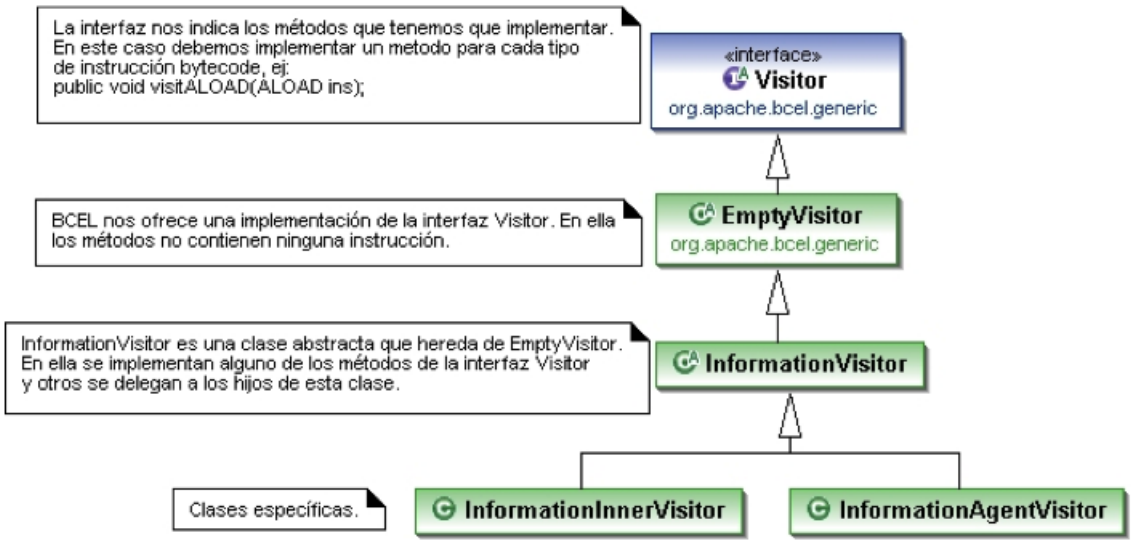

Figura 4.21: Diagrama jerarquía patrón visitor.

exactamente los mismos pasos que especificamos en el diseño.

- Crear el objeto (ClassGen) que represente al fichero de clase del agente seguro e ir añadiéndole los campos e instrucciones que no estén protegidos.
- Crear un objeto, por cada clase interna del agente original, que represente al fichero de clase de la nueva clase interna y añadirle sus campos e instrucciones.
- El funcionamiento de la clase InnerClassGen es muy parecido al de SecureAgentGen por lo que no vamos a extendernos en este punto.
- Crear el objeto que represente al fichero de clase del código privado e introducirle su estructura básica (ver figura 4.22).
- Hemos forzado a que sea la propia clase la que implemente la interfaz PrivateCode y hemos añadido el método "*execute()*" que implemente la interfaz.

```
this.secureAgent = new ClassGen(this.secureAgentName,
                        "jade.core.secureAgent._static.SecureAgent",
                        null,
                        Constants.ACC_PUBLIC,
                        originalAgent.getInterfaceNames());
```

Figura 4.22: Creación classgen para secureagent.

En este punto, hemos conseguido los objetos contenedores que representan a los ficheros de clases, pero estos están vacíos. El siguiente paso consiste en introducirles el código y los campos de la forma adecuada. Repartir los campos del agente original es una tarea bastante sencilla. Dependiendo si están o no protegidos los introducimos en el ClassGen

```
private void addInnerClasses() throws BuildSecureAgentException {
   int size = originalAgent.getInnerClasses().size();
   innerClasses = new ArrayList<InnerClassGen>();

   for (int i = 0; i < size; i++) {

     // Creamos una nueva clase interna por cada original
     AgentInnerClass innerClass = (AgentInnerClass) originalAgent
                                .getInnerClasses().get(i);

     InnerClassGen innerClassGen = new InnerClassGen(this,
                                                     innerClass,
                                                     secureAgentConfiguration);
     innerClasses.add(innerClassGen);
   }
}
```

Figura 4.23: Creación nuevas clases internas.

```
privateCode = new ClassGen(privateCodeName,
                "java.lang.Object",
                null,
                Constants.ACC_PUBLIC,
                new String[] {"jade.core.secureAgent._static.PrivateCode"});

...
executeMethod = new MethodGen(Constants.ACC_PUBLIC,
                                          Type.OBJECT,
                                          new Type[] {Type.OBJECT},
                                          new String[] {"arg0"},
                                          "execute",
                                          privateCodeName,
                                          iList,
                                          consPool);
```

Figura 4.24: Creación del ClassGen para el PrivateCode y el método *execute(arg0)*.

del PrivateCodeGen o del SecureAgenteGen respectivamente. Esta información la tenemos en el objeto SecureAgentConfiguration. En el código de la figura 4.25 podemos observar la creación de los nuevos campos, mostramos únicamente el caso de los campos no protegidos.

Aquí mostramos la instrucción que hemos enfatizado que es la encargada de introducir el nuevo campo *FieldGen* en el objeto *secureAgent*, el cual representa al fichero de clase del nuevo agente protegido. Los métodos tienen un proceso similar, por cada uno de los métodos pertenecientes al agente original creamos un nuevo método con el código y los atributos necesarios. A continuación describimos los pasos a seguir, para un mayor nivel de detalle de la implementación consultar la figura 4.26.

- En el constructor del *MethodGen* indicamos el nombre del método, el nombre de la clase a la que va a pertenecer, tipo del valor de retorno y demás atributos relevantes para el método.

- Transferimos del método original al nuevo método la tabla de excepciones, esto no es más que el conjunto de excepciones declarados en la sentencia throws del método.

```
private void addNonProtectedFields() {

    // Cogemos los campos no protegidos
    ArrayList<FieldReady> fields = secureAgentConfiguration.getNonProtectFields(
                                                    originalAgent.getClassName());
    ...
    // Por cada campo no protegido
    while (iterFields.hasNext()){

      FieldReady originalField = iterFields.next();

      FieldGen newField = new FieldGen(originalField.getField(),
                                       originalCP);

      newField.setConstantPool(consPool);
      newField.setType(originalField.getNewType());
      newField.setName(originalField.getNewName());
      secureAgent.addField(newField.getField());
    }
}
```

Figura 4.25: Creación de campos no protegidos

- Transferimos el código, donde podemos diferenciar:

  Variables locales, que son introducidas en el nuevo método.

  Instrucciones, tanto las protegidas como las no protegidas.

- Por último añadimos el método al objeto ClassGen secureAgent.

La transferencia de las instrucciones se realiza con la ayuda del patrón Visitor que, al igual que en la carga de agentes, nos ofrece una manera sutil de realizar dicha tarea. El diagrama de actividad siguiente 4.27 nos proporciona una idea de cómo se produce el traspaso de instrucciones. Si las instrucciones no están protegidas se introducen en el nuevo agente seguro, si por el contrario, las instrucciones están protegidas las emplazaremos en el método "*execute()*" del PrivateCode.

A modo de ejemplo mostramos algunos casos en los que el paso de instrucciones ha sido especialmente complejo. Instrucción protegida utiliza campos no protegidos. Este es uno de los casos más simples que nos podemos encontrar ya que, únicamente tenemos que cambiar los accesos a campos por accesos a variables locales. Vamos a centrarnos en el código de la figura 4.28 para realizar nuestro análisis. La instrucción marcada va a ser protegida y al no estar protegido el campo contador enviará su valor con los parámetros (Ver 6.1.5).

```
private void addOriginalMethods() {
    ...
    for (int i = 0; i < length; i++) {
      MethodReady method = originalAgent.getMethodList().get(i);

      ...
      // Creamos el nuevo metodo
      MethodGen newMethod = new MethodGen(method.getAccessFlags(),
                                          returnType,
                                      null,
                                      null,
                                          method.getName(),
                                      secureAgentName,
                                      ilist,
                                          consPool);
      transferExceptionTable(method, newMethod);
      ...
      transferCode(method, newMethod);
      ...
      // Añadimos el metodo al objeto ClassGen secureAgent
      secureAgent.addMethod(newMethod.getMethod());
    }
}
```

Figura 4.26: Creación de los métodos.

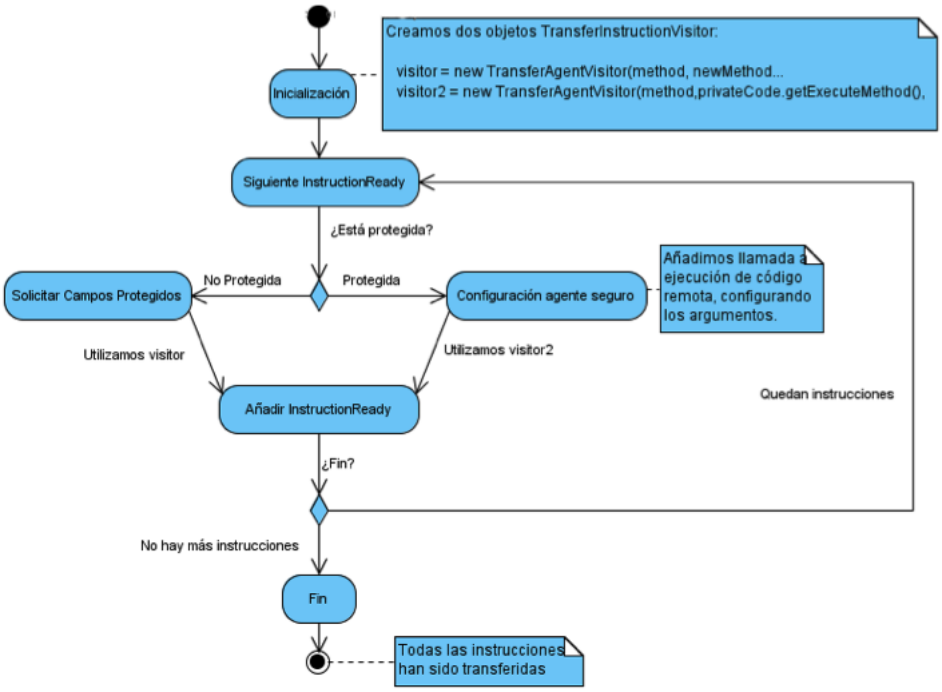

Figura 4.27: Diagrama de actividad.

Esto hace que en el código protegido tengamos que extraer los parámetros en una variable local e introducirlos en la nueva lista de instrucciónes. En la ilustración 4.29

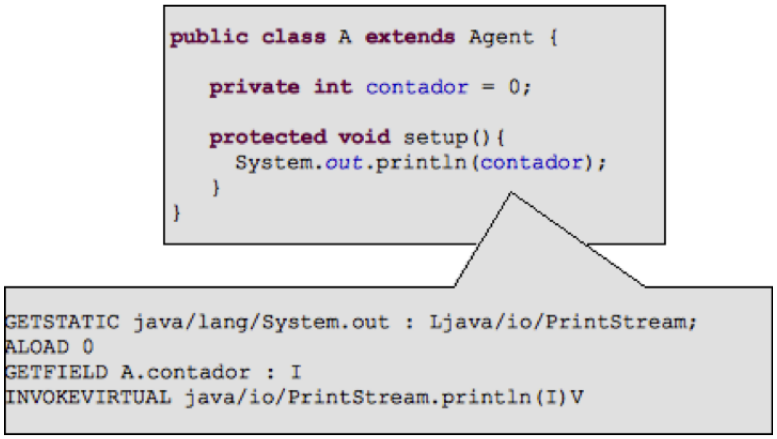

Figura 4.28: Código Java y bytecode del agente original.

mostramos el código privado resultado de esta operación.

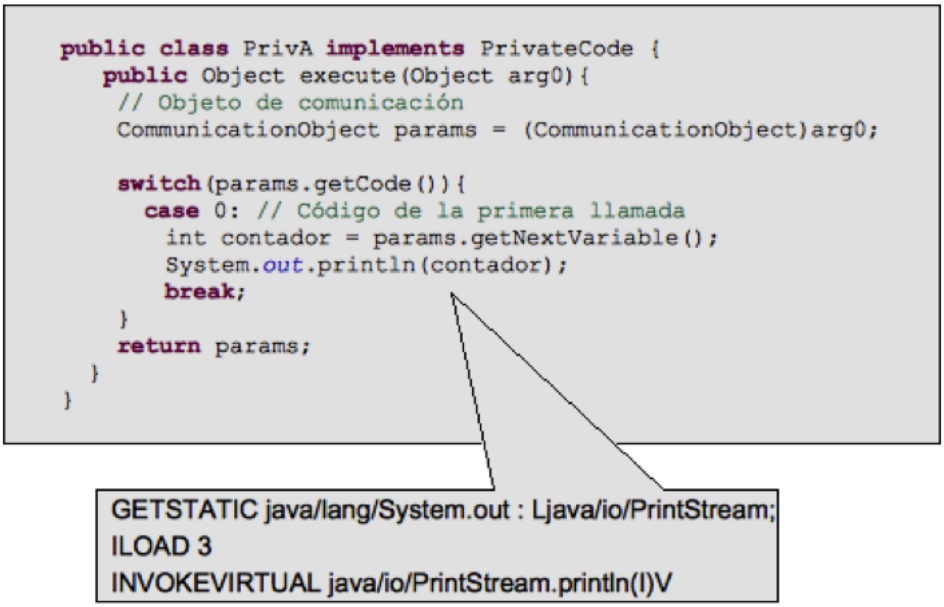

Figura 4.29: Código Java y bytecode del del PrivateCode generado.

En la ilustración 4.29 se aprecian los cambios introducidos en el bytecode para lograr el funcionamiento correcto. Gracias a la información del objeto InstructionReady podemos cambiar un acceso a un campo por la lectura de una variable local. Por último mostramos en el código de la figura 4.30 el aspecto del nuevo agente protegido.

```
public class AgentePrueba extends SecureAgent {
  public CommunicationObject communicationObject=
                                new CommunicationObject();
  public int contador = 0;

  protected void setup() {
    // Inicialización del Agente seguro
    ...
    new SetupThread().start();
  }

  public class SetupThread extends Thread{

    public void run(){

     communicationObject.setCommunicationCase(0);
     communicationObject.addVariableValue(contador);

     ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
     msg.setContent("execute-my-private-code");
     myArgs = communicationObject;
     msg.addReceiver(this.protectedBy);
     send(msg);
    }
  }
}
```

Figura 4.30: Ejemplo de agente seguro.

Cada instrucción no protegida utiliza campos protegidos. Nos basaremos en el mismo código de inicio de la figura 4.28. En el momento en que el algoritmo encargado de transferir las instrucciones tiene constancia de que la instrucción a transferir necesita un campo protegido, este debe introducir cierto conjunto de instrucciones complementarias para conseguir el campo deseado (ver 6.1.4).

```
communicationObject.setCommunicationCase(-2);
communicationObject.addVariableName("contador");

ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
msg.setContent("execute-my-private-code");
myArgs = communicationObject;
msg.addReceiver(this.protectedBy);
send(msg);
waitToResul();
int contador = communicationObject.getNextResult();

System.out.println(contador);
```

Figura 4.31: Ejemplo en caso de comunicación.

```
/* Llamada con unos argumentos*/
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
msg.setContent("execute-my-private-code");
myArgs = argumentos1;
msg.addReceiver(this.protectedBy);
send(msg);
system.out.println(result);
```

Figura 4.32: Ejecución de código privado.

```
while  (result == null){
     try {
       java.lang.Thread.sleep(time);
     } catch (InterruptedException e) {
            ...
     }
}
```

Figura 4.33: Espera del resultado.

Una vez obtenido el resultado, introducimos la instrucción, teniendo en cuenta que el acceso no será a un campo de la clase sino a una variable local.

En lo que respecta al desarrollo y la implementación de una interfaz gráfica, esta tarea puede consumir tanto tiempo como se desee, abierta en todo momento a reformas y nuevas maneras de mostrar los datos. En nuestro caso, no hemos prestado especial interés en este apartado, ya que no consideramos que este sea un objetivo primordial. No obstante, el trabajo se ha realizado para lograr la funcionalidad esperada. En la parte izquierda de la figura 4.18 podemos observar las partes de las que se compone nuestra interfaz gráfica que sigue un modelo vista-controlador. Para la clase "vista" *Java* nos ofrece la librería *Swing* para el manejo de ventanas. Esta librería dispone de muy buena documentación, bastante extensa y que cuenta con numerosas posibilidades.

4.3. Evaluación

Durante el desarrollo de cualquier solución o sistema van surgiendo diversos problemas y contratiempos. Es importante encontrar soluciones de fácil implementación, versátiles y robustas. Esto no siempre es posible, en muchas ocasiones al darle solución a un problema se están generando nuevos problemas que más tarde tendremos que resolver. Por ello, hay que analizar cada caso individualmente y elegir la solución más afín a nuestras necesidades. Debido a su importancia, hemos dedicado una secci ón para describir los problemas más relevantes que han ido apareciendo en el transcurso de la tesis. Los hemos dividido en secciones dependiendo del elemento software que provoca el problema. En el funcionamiento de "SecureAgent" encontramos ciertos inconvenientes que hemos tenido que solucionar que son inherentes a las tecnologías utilizadas.

Una vez aclarados estos aspectos de la solución, pasamos a la obtención de resultados. Para ello imaginemos que tenemos dos agentes protegidos con la estrategia de protección

mutua estática A y B. El agente A solicita la ejecución de su código protegido utilizando la siguiente secuencia de instrucciones:

- Las cuatro primeras instrucciones son de configuración, la quinta es la que marca el comienzo del paso de mensajes entre el agente A y el B (paso de parámetros, recepción del resultado, etc...), y la sexta instrucción debería mostrar por pantalla el resultado devuelto por el agente B.
- La primera vez que ejecutamos este código pudimos observar como el resultado que se mostraba por pantalla era "null" éste no era el comportamiento deseado.

Tras esta ejecución del código tenemos constancia de que surge un problema. La ejecución remota posee un retardo intrínseco que no podemos controlar, por lo que la variable "result" tarda cierto tiempo en capturar un valor adecuado y confiable. Hemos encontrado una solución elegante tras analizar algunas alternativas, esta consiste en dormir al hilo que ejecuta la llamada hasta que el resultado estuviese actualizado. Esto hizo que surgiese otro problema relacionado con los hilos y los comportamientos y que solucionaremos posteriormente. Este código lo introducimos dentro de un método al que llamaremos *waitToResult()*, el cual crearemos automáticamente al construir un agente seguro.

Existen otros problemas relacionados con los hilos de ejecución y los comportamientos. Al esperar el resultado llamando a la función *java.lang.Thread.sleep(time)* el hilo de ejecución del agente pasa al estado "dormido". Al dormirse el hilo también lo hacen los comportamientos que el agente posee, provocándose una situación de interbloqueo. Para que esto no ocurra, habría que separar el hilo que se encarga de la comunicación de los agentes seguros, del hilo normal de ejecución. JADE nos proporciona una clase capaz de ejecutar los comportamientos de un agente en un hilo independiente. La clase *ThreadedBehaviourFactory* encapsula un conjunto de comportamientos mediante el método *wrap*(comportamiento), y al añadirle los comportamientos al agente, éstos se ejecutan en un hilo independiente.

Así pues, para solucionar nuestro problema hemos tenido que realizar varios cambios:

- Modificar la inicialización de la clase *jade.core.secureAgent.static.SecureAgent* e introducir todos los comportamientos dedicados a la comunicación de los agentes seguros en un *ThreadedBehaviourFactory*.
- Añadir el código del método "setup" del agente original en el método *run()* de un nuevo Thread. En la figura 4.35 podemos ver cómo utilizamos este nuevo hilo. Su ejecución comienza en la última instrucción del método.

La justificación de esta última modificación se basa en el modo de funcionamiento de los agentes JADE. Los comportamientos no se empiezan a ejecutar hasta que el método "setup" finalice, aunque estos comportamientos hayan sido introducidos en un hilo de ejecución diferente mediante *ThreadedBehaviourFactory*.

Para realizar la ejecución en un solo método, la interfaz *PrivateCode* establece que las instrucciones protegidas de un agente seguro deben ir encluidas en el método *execute()*. Cuando se solicita la ejecución de código protegido, los comportamientos de los agentes realizan automáticamente el intercambio de información y llaman al método *execute()*.

```
public void inicializar(Agent ag){

   ThreadedBehaviourFactory tbf = new ThreadedBehaviourFactory();

   // Creamos los comportamientos en un nuevo thread
   Behaviour cer1 = tbf.wrap(cer);
   Behaviour rc1 = tbf.wrap(rc);
   ...

   //Añadimos los comportamientos
   addBehaviour(cer1);
   addBehaviour(rc1);
   ...

}
```

Figura 4.34: Inicialización SecureAgent.

```
public class A extends Agent {

  // Implementación metodo
  // Setup()
  protected void setup(){
     System.out.println("Hi!!");
  }
}
```

```
public class SecureA extends SecureAgent {

   // Método setup con el código de
   // inicialización para el agente seguro
   protected  void setup() {
      this.setProtectedBy(...);
      this.setProtectto(...);
      super.inicializar(this);
      this.privc = new PrivateCode1(this);
      new SetupThread().start();
   }

   // Clase interna para el thread que
   // ejecutará el código del método setup
   public class SetupThread extends Thread{
      public void run(){
         System.out.println("Hi!!");
      }
   }
}
```

Figura 4.35: Utilización del SetupThread.

```
public class A extends Agent {

   // Campo privado del agente A
   private int contador = 0;

   // Método setup()
   protected void setup(){

   // Primera seccion con un valor de contador
Sec1:   System.out.println("Valor inicial: " + contador);

   // Incrementamos el valor de contador
        contador++;

   // Segunda sección con el nuevo valor de contador
Sec2:   System.out.println("Valor final: " + contador);
        }
}
```

Figura 4.36: Ejemplo de diferentes secciones.

```
public class Privc implements PrivateCode {
  public Object execute(Object arg0){
    // Objeto de comunicación
    CommunicationObject params = (CommunicationObject)arg0;

    switch(params.getCode()){
      case 0: // Código de la primera llamada
              break;
      case 1: // Código de la segunda llamada
              break;
    }
    return params;
  }
}
```

Figura 4.37: Estructura método "execute".

Sin embargo, nos encontramos con que el código protegido no tiene que ser contiguo, este puede estar dividido en secciones, pertenecer a diferentes métodos e incluso a diferentes clases internas. Esto hace que tengamos que diferenciar qué parte de código queremos ejecutar en una llamada concreta y ser capaces de diferenciar estas secciones en el método "*execute()*".

Para solucionar este inconveniente hemos creado una estructura común para todos los métodos *execute()* con la forma que se muestra en la siguiente ilustración 4.37. El único problema que queda por solucionar es obtener el identificador de cada sección. Este deberá venir en los parámetros que se envían al solicitar la ejecución de código protegido.

El acceso de campos protegidos se realiza de la siguiente forma. A parte de las secciones de código que crean las respectivas llamadas de ejecución de código protegido, existen dos casos de especial interés:

- Consulta de campos protegidos por parte del agente seguro.
- Variación de campos protegidos por parte del agente seguro.

La descripción del problema es la siguiente. En ocasiones, es necesario proteger campos del agente original, esto requiere que dichos campos se introduzcan en la clase PrivateCode. Cuando el código del nuevo agente seguro acceda a alguno de estos campos lo hará solicitando la ejecución de código protegido, pues esta es la única manera que implementa la librería jade.core.secureAgent. Así pues, tenemos que diferenciar entre los casos en los que se solicita la ejecución de un trozo de código del agente original y los casos en los que se solicita un campo protegido. Para solucionarlo, introducimos una pequeña variación al código que aparece en la ilustración 4.37, en ella contemplamos dos casos extras. El acceso a campos protegidos de lectura.

Identificaremos este caso en la estructura switch como el "-2". Se podrán leer tantos campos protegidos como se desee. Toda la información necesaria se encuentra en el objeto *communicaionObject* que envía el agente que solicita el acceso a campos protegidos. Y el acceso a un campo protegido de escritura. Identificaremos este caso en la estructura switch como el "-1". De esta forma, la estructura del método *execute()* queda como se muestra en la ilustración 4.38.

Cuando hablamos de la librería SecureAgent en el capítulo 3 comentamos que el intercambio de parámetros entre los agentes seguros se realiza a través del campo *SecureAgent.myArgs. Este campo está definido de tipo* Object *con la restricción de que debe implementar la interfaz serializable.* En la mayoría de los casos, es necesario más de un parámetro para la ejecución correcta del código, ya que, como hemos dicho en el apartado anterior, en estos parámetros debe ir el identificador de la sección que queremos ejecutar. Además, es bastante probable que una sección contigua de código necesite más de una variable para su ejecución, ya sean variables locales, campos del agente, etc. Para dar solución a este inconveniente hemos decidido crear una entidad que encapsule toda la información que necesitemos y los métodos para acceder, modificar o eliminar esta información. Esta entidad viene modelada por la clase *CommunicationObject.* Sus elementos más relevantes son:

- El identificador del número de sección a ejecutar.
- Lista con los valores de los diferentes parámetros que necesita la sección de código.
- Lista para introducir los resultados.
- Conjunto de métodos para garantizar la coherencia a la hora de extraer e introducir elementos.

En lo que respecta al trabajo con la librería BCEL, hemos de mencionar que no ha sido una tarea trivial, los errores son difíciles de localizar y la escasa documentación me han provocado una gran confusión en ciertos aspectos. BCEL dispone de una tabla de variables locales, y en la especificación de la Maquina Virtual de Java existe un atributo asociado al código de los métodos de especial interés. *LocalVariableTable* es un atributo opcional, donde se encuentra la información relativa a las variables locales de los métodos, que nos han resultado de gran interés. Sin embargo, existen algunos casos extraordinarios en los que estas tablas están imcompletas o simplemente inexistentes.

Todo ello ha provocado que en cierto punto no pudiésemos identificar las variables que se utilizan por lo que en la primera versión generábamos errores en la creación del nuevo

```
public class Privc implements PrivateCode {
   // Campo protegido
   private int contador;

   public Object execute(Object arg0){
    // Objeto de comunicación
    CommunicationObject params = (CommunicationObject)arg0;

    switch(params.getCode()){

      case -2: // Código de acceso a un campo para lectura
          for (int i = 0; i < params.getVariablesNameSize(); i++){

           if (params.getNextVariableName().equals("contador")){

             params.addResult(contador);

           // Else if por cada campo protegido
           } else if (...){

           }
         }
          break;
      case -1:// Código de acceso a campos para escritura
        if (params.getNextVariableName().equals("contador")){

          contador = params.getNextVariable();

          // Else if por cada campo protegido
          } else if (...){

          }
          break;
      case 0: // Código de la primera llamada
              break;
      case 1:
           ...// Resto de llamadas
    }
    return params;
  }
}
```

Figura 4.38: Estructura método “execute”.

agente. Para lo cual, tomamos la iniciativa de identificar los casos en los que ocurría el problema e indicarlos como restricciones (como clases internas anónimas). No está permitido cargar un fichero de clase de un agente que contenga clases internas anónimas. Para resolver este problema diseñamos el siguiente código que aclara el concepto de clase interna anónima mediante un ejemplo práctico. El compilador crea una clase *Anonimous* que hereda de *Thread*. Los métodos de acceso son creados por el compilador cuando existe en la clase interna un campo o método privado y se realiza un acceso a ese campo o método desde la clase externa. Dichos métodos generados automáticamente no contienen la tabla de variables locales correctamente configurada por lo que es complejo trabajar con ellos. En consecuencia no están permitidos los campos y los métodos privados en las clases internas. A pesar de haber tenido más problemas a lo largo del desarrollo los consideramos menores siendo los anteriormente mencionados los más relevantes y con especial interés.

4.4. Conclusiones y mejoras futuras

En los últimos años, el afán del ser humano ha sido delegar las tareas más tediosas a entidades software o hardware para que las realicen, y si además, estas entidades pueden gozar de cierta inteligencia, el campo de actuación se multiplica.

Internet, además de ser la red de redes que canaliza gran cantidad de información y conecta casi todos los rincones del mundo, es un nido de inseguridades. Cuando hablamos de entidades software que se desplazan por multitud de host ubicados en distintas partes del mundo y que no tenemos la certeza de que actúan de buena fe, no podemos implementar un sistema que trabaje con información crítica sin tomar medidas. Existen multitud de desarrollos teóricos sobre sistemas de seguridad aplicados a agentes móviles, pero hay muy pocos implementados.

A lo largo de este capítulo se han presentado el análisis, diseño y posterior implementación de dos técnicas que hacen a una comunidad de agentes móviles más segura. La primera de ellas es la más simple, que además presenta una menor carga, tanto computacional como de red. Está pensada para sistemas muy poco cambiantes en cuanto a las tareas a realizar y donde los ataques potenciales sean más difíciles. La segunda de ellas, la técnica de protección mutua dinámica ofrece más versatilidad y seguridad, aplicable a sistemas muy cambiantes y con riesgos potenciales altos.

Al ser un sistema distribuido no conocemos en todo momento en qué situación se encuentran los agentes. Si algún agente ha dejado de existir, si la ejecución remota ha sido correcta, si la recepción de los parámetros ha sido satisfactoria, etc. Son tantos los posibles problemas que pueden suceder, que su gestión remota es complicada.

Como futuras mejoras, partiendo del desarrollo del sistema de protección mutua dinámica, que es el más completo y útil, podemos sugerir varias mejoras. En primer lugar la gestión de claves, que se realiza mediante el uso de una Autoridad de Certificación (CA) aunque hay un campo abierto de investigación. En un ejemplo más sencillo cada agente genera su par de claves pública y privada, cada vez que se inicia, sin embargo, desde un punto de vista crítico de la seguridad esto no es viable. Con lo cual es necesario buscar algún tipo de solución intermedia que nos solucione el problema y no nos fuerce a tener que montar las CAs y todo el manejo de claves y certificados asociados. Esto, por otro

lado, conlleva una sobrecarga considerable en el sistema.

Actualmente, el código privado de cada agente está estructurado en una única sección, de forma que sólo se puede solicitar la ejecución de la totalidad de la sección. Otro campo que queda abierto se basa en el estudio de la estructuración del código privado de cada agente en varias secciones y que se pudieran solicitar ejecuciones de secciones independientes. El resultado de la ejecución de cada sección podría ser una entrada en la ejecución de otra sección y así sucesivamente. De esta forma, la seguridad aumentaría considerablemente. Entre otras cosas se podrían diseñar esquemas de protección específicos para ciertos problemas pudiendo estar cada agente coprotegido por varios agentes del sistema.

El método "*execute()*" de la clase *PrivateCode*, recibiría un argumento que sería el número de sección a ejecutar. Dentro del método "*execute()*" se podría realizar una bifurcación y dependiendo del argumento recibido, entraría por ese lugar y ejecutaría el trozo de código correspondiente. Cada resultado parcial podría ser almacenado en la variable result de la clase *DynamicSecureAgent* y poder ser utilizado por sucesivas ejecuciones con sólo acceder a la variable.

Entre las posibles extensiones de esta solución destacamos el proporcionar una mayor versatilidad en la configuración por parte del usuario. El sistema no está configurado correctamente por el usuario ya que el propio sistema configura automáticamente a los nuevos agentes protegidos.

Aunque habría que valorar si este sistema de coprotección es eficiente en términos de consumo de recursos y sobre todo si merece la pena este consumo de recursos para conseguir cierto objetivo en concreto, consideramos que sería una buena práctica el realizar un estudio analítico en términos de consumo de recursos en función del tipo de sistema, es decir, si se trata de un sistema con un gran número de agentes o reducido, si los agentes necesitan un uso masivo de recursos, si necesitan un elevado grado de seguridad, etc.

Otra de las vías que hemos dejado abierta para posibles investigaciones está relacionada con el almacenamiento de código privado. En nuestra versión actual, cada vez que un agente solicita la ejecución de su código privado, se envía tanto el código privado como la clave de cifrado relacionada y correspondiente al agente protector. En el caso en que se realicen peticiones de ejecución del mismo código de forma muy continuada puede llegar a ser bastante ineficiente, puesto que se estaría enviando continuamente la misma información ya almacenada. Aunque algunas de las técnicas existentes de cacheo se podría trasladar a este área sin mayor complicación, considerando en todo momento que estamos trabajando en seguridad y que hemos de ser cautos y no dejar puertas traseras abiertas. Una solución sencilla consiste en que el agente protector almacene el código privado de su agente protegido la primera vez que solicita la ejecución del mismo. Mientras el agente protegido no cambie su código privado a ejecutar, no será necesario pasar dicho código por la red, simplemente ejecutarlo a petición del agente protegido. Los únicos datos a enviar serían los argumentos, que en volumen son menos pesados para la red. Para cambiar esto en el desarrollo se debería modificar el comportamiento que gestiona la recepción de peticiones de ejecución. De forma que si ya tenemos almacenado el código privado del agente protegido, de una ejecución anterior, se salte el paso de pedir el código privado y proceda directamente a la ejecución del mismo. Aunque esta alternativa es mejorable considerablemente, tanto en aspectos relacionados con la eficiencia como en la sofisticación

de la misma.

Otro campo en el que trabajar es el de las rutas de los agentes. Además de proporcionar un grado extra en la seguridad el control de las rutas, sería deseable que los agentes que colaboren en la realización de una misma tarea no pasen todos por la misma agencia.

Puede darse el caso de que todos los agentes pasen por la misma agencia y soliciten desde ella una ejecución remota de su código privado. De forma remota esto podría provocar que una agencia con caracter malicioso pudiera reconstruir parte del código de los agentes implicados y pudiera intuir qué tareas se están intentando realizar. Lo cual va totalmente en contra de la filosofía de esta técnica que pretende proteger el sistema. Hacemos referencia a que esto se hace de una forma muy remota puesto que las posibilidades de estudio de comportamiento de un software de caja negra no son triviales, aunque bien es cierto que si se repiten las mismas tareas con mucha frecuencia se podrían obtener cierta información.

Si logramos que los agentes tengan conocimiento de la ruta que siguen sus colaboradores, podrían desplazarse sólo por agencias por las que no han pasado ellos. Esto sería posible si diseñamos un sistema multiagente donde cada uno tiene una ruta inicial preasignada, de forma que podamos realizar rutas disjuntas. Aunque también hay que hacer un balance de hasta qué punto puede ser seguro que un agente tenga una ruta preasignada, ya que esto podría entablar problemas de seguridad aún más severos que el planteado.

Capítulo 5

Conclusiones y Líneas Futuras

En este capítulo ofrecemos una reflexión del trabajo realizado en esta tesis, además de presentar las conclusiones más relevantes y los trabajos futuros para continuar el fruto del trabajo de la tesis. En primer lugar, revisamos los objetivos iniciales de nuestro trabajo, describimos la forma en que se han conseguido y finalizamos con la identificación de los trabajos futuros más relevantes. La estructura de este capítulo es la siguiente, en la primera sección presentamos cómo se han conseguido los objetivos, la segunda sección presenta una visión global del trabajo de la tesis y las secciones 5.3, 5.4 y 5.5 están dedicadas a diferentes líneas futuras de trabajo.

5.1. Objetivos superados

En el capítulo 1 identificamos y describimos en detalle los principales objetivos de esta tesis. Estos objetivos son:

- El estudio de los requisitos, restricciones y puntos débiles relacionados con la seguridad de los sistemas basados en agentes. Realizamos este estudio tanto desde el punto de vista del agente como desde el de la agencia.
- Un análisis de los mecanismos que existen actualmente enfocados a proporcionar seguridad a los sistemas basados en agentes. Hemos evaluado estos mecanismos teniendo en cuenta tanto aspectos de seguridad como de usabilidad que consideramos esencial para la aceptación de los sistemas basados en agentes.
- Diseño de soluciones para la construcción de sistemas seguros basados en agentes, junto con las herramientas necesarias para facilitar el uso de estas soluciones por desarrolladores que no son especialistas en el campo de la seguridad.
- La validación de nuestra solución por medio de análisis formales de nuestras dos soluciones, así como mediante la aplicación de las mismas en la solución de problemas del mundo real usando sistemas basados en agentes.

Los dos primeros objetivos se han conseguido de forma simultánea ya que el estudio de los mecanismos existentes hasta el momento ha ayudado en la identificación de los puntos

débiles de estos mecanismos. En el capítulo 2 incluimos una descripción detallada de todos los mecanismos, así como una descripción de las debilidades de los mismos, de esta forma mostramos cómo se han conseguido ambos objetivos. Para conseguir superar el principal objetivo de esta tesis se han considerado dos alternativas en esta tesis. La primera de estas se basa en el uso de hardware específico para la seguridad y la segunda de ellas se basa en el modelo de "computación protegida". Creemos que estas dos soluciones complementarias cumplen nuestro objetivo. De hecho, ambas soluciones se han implementado de manera que se puedan utilizar en sistemas basados en ya en funcionamiento sin la exigencia de grandes cambios, tal y como se describe en los capítulos 2 y 4. El resultado tangible de la primera solución es una librería que implementa el protocolo de migración segura diseñado. La implementación de la segunda solución, que a priori presenta un mayor grado de dificultad, ha sido también realizada a través de un conjunto de herramientas diseñadas para facilitar la aplicación de esta técnica por desarrolladores de un nivel medio.

5.2. Conclusiones

Hemos llevado a cabo un estudio detallado del problema de la protección de sistemas basados en agentes centrándonos en diferentes aspectos, además de describir los objetivos a conseguir en cada uno de los diferentes casos. Las soluciones que se han desarrollado se centran en dos iniciativas diferentes, una basada en proporcionar una solución segura basada en el uso de un elemento externo y la otra propone una solución que hace uso de la capacidad de colaboración entre los agentes para la resolución de un problema como la base para asegurar un mínimo de seguridad, tal y como se describe en el capítulo 2. En esta tesis se muestra cómo ambos mecanismos se pueden utilizar en diferentes escenarios para la protección del sistema global.

También hemos llevado a cabo un análisis de los diferentes mecanismos de protección existentes, tanto a nivel académico como industrial, y hemos hecho una clasificación de estos en base a la naturaleza del propio mecanismo. Hemos mostrado cómo los mecanismos existentes son inadecuados, y en la mayoría de los casos están basados en aplicaciones directas de soluciones diseñadas para software en general y no para sistemas basados en agentes. En otros casos, hemos observado que la aplicación de soluciones existentes requiere que los desarrolladores tengan conocimientos avanzados de los mecanismos de seguridad. Como parte de la revisión de los trabajos relacionados, hemos resaltado que una alternativa para asegurar la protección de los sistemas basados en agentes se basa en el uso de la tecnología de *Trusted Computing*. Hemos trabajado con la propiesta más destacada dentro de este campo: la representada por el Trusted Computing Platform Alliance. Esta iniciativa está guiada por el diseño de la solución presentada en el capítulo 2.

Para la validación de esta solución hemos decidido tomar dos caminos alternativos, lo cual proporciona un nivel superior de robustez de la solución. En primer lugar, hemos realizado una validación formal del protocolo de migración segura de nuestra solución. Para ello, hemos hecho uso de un conjunto de herramientas conocido como AVISPA. Además hemos aplicado la solución a un caso de uso del mundo real y hemos diseñado un sistema que funciona como simulador de una planta nuclear en el que hemos construido

un sistema basado en agentes que realiza una monitorización del funcionamiento correcto de cada una de las partes del sistema. Obviamente, este escenario requiere un alto nivel de seguridad.

5.3. El papel de los agentes en los ambientes inteligentes

Tanto los portátiles, como las PDAs y los teléfonos móviles de tercera generación cuentan con algún tipo de conexión inalámbrica que nos permite el acceso a diferentes redes de datos en cualquier momento y prácticamente en cualquier lugar en que nos encontremos. La evolución tanto en tamaño como en capacidades de estos dispositivos, así como de las comunicaciones inalámbricas han permitido la conexión permanente de los usuarios a la red. Este incremento en la movilidad se puede asemejar al que hubo años atrás desde el clásico ordenador de sobremesa al portátil y va acompañada de una rápida adaptación por parte de los usuarios a este nuevo modelo de comunicación. El siguiente paso sería ayudar a la gente a ser capaces de apreciar la existencia de otras máquinas [63]. Las expectativas para el nuevo software son que será capaz de manejar conceptos como la localización y percepción del contexto, personalización, adaptabilidad, crecimiento orgánico, movilidad y otras muchas características que imponen la necesidad de un software más completo con métodos de ingeniería del software y nuevos e innovadores lenguajes de modelado [64].

La Inteligencia Ambiental (AmI) se centra en adaptar nuestro entorno sensitivo a nuestras necesidades y responder de forma inteligente a los usuarios y a los cambios de contexto del propio entorno [65]. Se espera que los objetos que nos rodean en nuestra vida cotidiana actúen de manera autónoma sin necesidad de la intervención humana. Este tipo de ambientes y la computación ubicua están íntimamente relacionados, ya que se pretende proporcionar a las personas servicios y utilidades de forma transparente, sin que explícitamente lo soliciten. El paradigma del agente es muy prometedor para la implementación de sistemas complejos como los de comercio electrónico, tráfico aéreo, planificación de recursos corporativos, etc. [66]. Inicialmente creado en el seno de la Inteligencia Artificial, la comunidad científica y los investigadores han encontrado otras áreas en las que explotar de forma fructífera el paradigma del agente. Como se defiende en [67], [68] este paradigma ha tenido un especial interés en la ingeniería del software, ya que cambia el paradigma de la orientación de objetos. Este cambio se basa en la apreciación del mundo como una sociedad de elementos inteligentes llamados agentes, que tiene percepción y pueden decidir. Esta manera de ver el mundo se diferencia del de la orientación a objetos, ya que ve conceptualmente al mundo como una colección de objetos sin autonomía.

Una de las debilidades de los entornos de Ambientes Inteligentes es la falta de modelos y de prácticas de ingeniería del software que ayuden en en análisis de requisitos, diseño, verificación, pruebas, etc. Hasta el momento, la investigación en este área está en su etapa más temprana, y la necesidad del desarrollo de estas metodologías goza de una extensa aceptación. Creemos que el paradigma del agente es de mucha utilidad, no sólo para la implementación de sistemas AmI, sino también para todas las fases del ciclo de desarrollo de la ingeniería. Somos conscientes de que las nuevas metodologías de la

ingeniería del software orientada a agentes debe ser revisada y extendida para la ingeniería de los sistemas de Ambientes Inteligentes.

Sin embargo, a pesar de la atención prestada por la comunidad investigadora, esta tecnología no ha tenido una gran acogida, sobre todo a nivel industrial. De hecho, sólo se ha aplicado en escenarios muy concretos y limitados del mundo real. Esto se debe, en gran parte, por la existencia de ciertos problemas de seguridad que, de hecho, se considera la dificultad más urgente a resolver en esta tecnología antes de estar lista para usarse en el marco industrial. En lo referente a la seguridad, los agentes presentan características muy interesantes y pueden cubrir requisitos de seguridad por parte de diferentes actores, con un modelo para lograr los objetivos de cada parte en colaboración. Evidentemente, esta tecnología no estará suficientemente madura para usarse de forma extendida hasta que los problemas de seguridad más importantes se hayan solucionado. Esto consiste tanto en encontrar los mecanismos apropiados como en la facilitación de la aplicación de estos mecanismos.

Tradicionalmente se han aplicado algunos de los mecanismos de protección genérica de software para la protección de sistemas de agentes. Sin embargo, las características especiales de esta tecnología obligan al uso de soluciones específicas. Algunos mecanismos de protección están orientados a la protección de los hosts del sistema frente a agentes con carácter malicioso. La solución más relevante para este problema pensamos que viene por la utilización del concepto del Sandbox, consistente en un contenedor que limita o reduce el nivel de acceso que los agentes tienen y proporcionan ciertos mecanismos para controlar la interacción entre ellos. Otra técnica que trata esta vertiente del problema se conoce como proof carrying code (o código portador de prueba) [69], esta técnica se basa en que cada fragmento de código incluye una prueba detallada que se puede usar para determinar si la política de seguridad del host se satisface por parte del agente. Los hosts sólo tienen que verificar que la prueba es correcta (lo que significa que corresponde con el código) y que es compatible con la política de seguridad local, aunque hemos de mencionar que la aplicación de esta técnica no resulta sencilla, de hecho en multitud de ocasiones no es fácil encontrar estas políticas de seguridad.

Otros mecanismos están orientados a la protección de los agentes frente a posibles agencia (hosts) maliciosos. La más representativa es la que se conoce como el uso de santuarios [40], que consisten en entornos de ejecución en los que el agente móvil puede ejecutarse de forma segura. La mayoría de estas propuestas se construyen asumiendo que la plataforma donde se implementa el santuario es segura. Desgraciadamente, asumir esto no funciona en la práctica para los sistemas basados en agentes, puesto que no podemos basarnos en que todas las plataformas sean confiables.

Por otro lado, se pueden aplicar varias técnicas para verificar la propia integridad. Esta idea resulta de utilidad para evitar que tanto el código como los datos del agente puedan ser manipulados de forma inadvertida. Dentro de este conjunto de técnicas anti-manipulación se incluyen otras como son las de cifrado, checksum, anti-depurado, anti-emulación, etc.[70], que en definitiva comparten el mismo objetivo, pero que también están orientadas a la prevención del análisis de la función que implementa el propio agente. Todas estas técnicas presentan un grado de dificultad bastante alto en su aplicación, especialmente para los programadores que utilicen esta tecnología, principalmente debido al hecho de que la aplicación de estas técnicas implica ciertos conocimientos mínimos de

seguridad.

Una vez expuestas las diferentes alternativas, defendemos el hecho de que todas las técnicas descritas proporcionan protección a corto plazo, además de requerir una base de conocimiento mínima en temas de seguridad, por lo tanto, no son útiles para nuestro propósito. En cambio, en ciertos escenarios, pueden representar una solución bastante factible, especialmente combinada con otras soluciones. Existen algunas soluciones teóricas al problema, de hecho se ha demostrado que la autoprotección de código no es viable, es decir, que no se puede asegurar la fiabilidad de una solución que sólo se base en software para la protección [71]. En algunos escenarios, la protección requerida está limitada a algunas partes del software (código o datos). De esta forma, la función implementada por el software o los datos han de estar ocultos para el host en el que se ejecuté el software. Algunas de estas técnicas requieren un paso de procesamiento adicional externo para obtener los resultados deseados. Entre estos esquemas, las técnicas de ocultación permiten la evaluación de funciones cifradas [33]. Esta técnica se basa en proteger los datos procesados para la protección del propio agente. Por esta razón, es una técnica apropiada para la protección de los agentes. Sin embargo, sólo es aplicable para la protección de funciones polinomiales.

El caso de los esquemas de colaboración en línea también es muy interesante. En estos esquemas, parte de la funcionalidad del software se ejecuta en uno o más computadores externos. La seguridad de esta solución depende de la imposibilidad de cada parte de identificar la función realizada por otros. Esta solución es muy apropiada para arquitecturas de computación distribuidas como son los sistemas basados en agentes o la computación grid.

Por último, existen técnicas que crean una protección bidireccional, protegen tanto al agente como a la agencia. Entre ellas algunas se basan en el uso de soluciones basadas en hardware, como algunas que usan el Trusted Computing Platform como elemento base [17]. De hecho, con la reciente aparición de la computación ubicua, la necesidad de una plataforma segura se ha consolidado como un hecho evidente. Esta solución añade un componente confiable a la plataforma de computación, que normalmente consiste en cierto hardware integrado en la placa madre del ordenador y que se utiliza para crear confianza en los procesos software [72]. Otras técnicas se basan únicamente en elementos software, como por ejemplo la que hace uso de la computación protegida [49]. Esta técnica se basa en la partición de los elementos software en dos o más partes mutuamente dependientes, de forma que el código será ejecutado de forma remota por diferentes agentes.

5.4. Aplicación del mecanismo de protección en clouds computing

La computación en nube, del inglés cloud computing, es un paradigma que permite ofrecer servicios de computación a través de Internet. Evidentemente, la nube es una metáfora de Internet con ciertas particularidades.

En este tipo de computación todo lo que puede ofrecer un sistema informático se ofrece como servicio, de tal forma que los usuarios puedan acceder a los servicios disponibles "en la nube de Internet" sin conocimientos (o, al menos sin ser expertos) en la gestión de los

recursos que usan. Según el IEEE Computer Society, es un paradigma en el que la información se almacena de manera permanente en servidores en Internet y se envía a cachés temporales de cliente, lo que incluye equipos de escritorio, centros de ocio, portátiles, etc. Esto se debe a que, pese a que las capacidades de los PC han mejorado sustancialmente, gran parte de su potencial está desaprovechado, al ser máquinas de propósito general. La computación en nube es un concepto que incorpora el software como servicio, tal como la Web 2.0 y otros recientes, también conocidos como tendencias tecnológicas, en los que el punto en común es la confianza en Internet para satisfacer las necesidades de cómputo de los usuarios.

Como ejemplos de Computación en Nube destacan Amazon EC2, Google Apps, eyeOS y Microsoft Azure que proveen aplicaciones comunes de negocios en línea accesibles desde un navegador web, mientras el software y los datos se almacenan en los servidores.

El concepto cómputo cloud empezó con proveedores de servicio de Internet de gran escala tales como Google y Amazon, así como otras que construyeron su infraestructura previa. Así surge una arquitectura consistente en un sistema de recursos horizontalmente distribuidos, introducidos como servicios virtuales. Este modelo arquitectónico fue inmortalizado por George Gilder en un artículo de 2006 publicado *Wired* y titulado *"Las Fábricas de Información"*. Las granjas de servidores acerca de las cuales Gilder escribió eran similares en su arquitectura al cómputo grid, pero mientras que los grids son utilizados para aplicaciones de cómputo técnico "loosely coupled" (o sea un sistema compuesto de subsistemas con cierta autonomía de acción a la par que mantienen una interrelación continua con los otros componentes) este nuevo modelo de nube se estaba aplicando a los servicios de Internet.

Tanto las nubes como los grids estan hechos para escalar horizontalmente muy eficientemente. Ambos están construidos para resistir fallos de los elementos o nodos individuales y se cargan "por-uso". Pero mientras que los grids típicamente procesan los trabajos en batch, con un punto definido de inicio y final, los servicios nube pueden ser continuos. Lo que es más, las nubes expanden los tipos de recursos disponibles como el almacenamiento de archivos, las bases de datos, y los servicios web y extienden la aplicabilidad a la Web y a las aplicaciones de la empresa.

Al mismo tiempo, el concepto de cómputo de programas utility llegó a ser el foco de diseño y operaciones de las tecnologías de la información. Nick Carr describe en su libro "The Big Switch" [73] que la infraestructura de los servicios de cómputo empieza a ser comparable con el desarrollo de la electricidad como utilidad. El autor afirma que sería grandioso si se pudieran comprar recursos de computación bajo demanda, sólo pagando lo que se necesite y en el momento adecuado.

Dado que la computación en nube no permite a los usuarios poseer físicamente los dispositivos de almacenamiento de sus datos (con la excepción de la posibilidad de copiar los datos a un dispositivo de almacenamiento externo, como una unidad flash USB o un disco duro), deja la responsabilidad del almacenamiento de datos y su control en manos del proveedor.

La computación en nube ha sido criticada por limitar la libertad de los usuarios y hacerlos dependientes del proveedor de servicios. Algunos han criticado este paradigma afirmando que sólo es posible usar las aplicaciones y servicios que el proveedor esté dispuesto a ofrecer. Así, el London Times compara la computación en nube con los sistemas

centralizados de los años 50 y 60, en los que los usuarios se conectaban a través de terminales con ordenadores centrales.

Generalmente, los usuarios no tenían libertad para instalar nuevas aplicaciones, y necesitaban la aprobación de administradores para desempeñar determinadas tareas. En definitiva, se limitaba tanto la libertad como la creatividad. El Times argumenta que la computación en nube es un retorno a esa época. De forma similar, Richard Stallman, fundador de la Free Software Foundation, cree que la computación en nube pone en peligro las libertades de los usuarios, porque éstos dejan su privacidad y datos personales en manos de terceros. Se ha afirmado que la computación en nube es simplemente una trampa destinada a obligar a más gente a adquirir sistemas propietarios, bloqueados, que les costar cada vez más caros.

Una vez claro el concepto de la computación en nube y las ventajas que proporciona este modelo nos planteamos un paso más allá en la evolución de esta tecnología. Hasta el momento la base de esta computación radicaba en el envío de una serie de datos a procesar por la nube, con la consiguiente delegación de confianza en la misma nube por parte del usuario. Sin embargo, este modelo queda restringido a las funcionalidades proporcionadas por el proveedor de la nube. Por lo que el usuario únicamente podrá ejecutar los programas que estén integrados en la nube, proponemos que el usuario pueda enviar a la nube además de los datos sus propios programas.

Estableciendo un paralelismo con la tecnología de agentes, que básicamente son entidades de computación que pueden englobar cualquier código. La principal ventaja que encontramos en verlo de esta forma es que con los avances obtenidos en este trabajo de tesis tendremos solucionados varios de los problemas de seguridad derivados de la ejecución del cierto código en un host ajeno.

5.5. Tecnologías alternativas y complementarias

A lo largo de toda esta tesis hemos presentado dos propuestas que a priori enfocaban el problema desde dos perspectivas diametralmente opuestas. La primera de ellas basa su seguridad en un hardware criptográfico de apoyo. En cambio, la segunda solución presentada en este trabajo de tesis consiste en la aplicación de la técnica de "computación protegida" para la protección software entre agentes, como se describe detalladamente en el capítulo 4. Al final de cada uno de los capítulos dedicados a la presentación de cada una de las soluciones presentamos unas líneas de las conclusiones y los posibles trabajos futuros para cada una de las soluciones. Sin embargo, tras realizar un análisis global del problema, nos planteamos el uso de ambas técnicas de manera conjunta para la obtención de una solución que aunque presente de forma evidente un mayor grado de complejidad también se espera un mayor nivel de seguridad en la solución.

La idea radica en la aplicación directa de la técnica propuesta por la computación protegida. En cambio, en vez de realizar un anillo como se propone en el capítulo 4 la idea es realizar una partición del código de los agentes de forma que cada agente quede desdoblado en dos partes, siendo una de estas partes ejecutada directamente por el propio TPM.

Evidentemente la aplicación de esta técnica implica una latencia mayor, además de

todos los inconvenientes de las dos técnicas desarrolladas puesto que los requisitos de uso de la misma son un conjunto formado por la unión de los conjuntos de requisitos de ambas técnicas.

No obstante, defendemos la idea de que resultaría de gran interés el estudio de una combinación de ambas técnicas para la obtención de un nivel superior de seguridad, especialmente para ciertos casos en los que la seguridad del sistema pueda verse cuestionada.

El principal argumento para la combinación de ambas técnicas en una misma solución radica en obtener una solución capaz de proteger tanto al agente como a la agencia, es decir, en conseguir una protección en ambos sentidos.

Apéndice A

Thesis summary

A.1. About this work

The present work has been carried out as a PhD student in the Gisum a research group of the Computer Science department of the University of Málaga. Contributions of this work have been partly funded by the following research projects:

- UBISEC: Ubiquitous Networks with a Secure Provision of Services, Access, and Content Delivery (IST-FP6-506926).
- SERENITY: System Engineering for Security and Dependability (IST-PF6-027587).
- Mistico-Mechanics: Proyecto de la Junta de Castilla la Mancha en colaboración con la Universidad de Castilla La Mancha.

A.2. Motivation and main goals

Mobile agents are software entities with the ability to migrate from node to node in computer networks. Agents act both autonomously and in cooperation with other agents to perform a set of tasks. Nowadays a large number of applications based on agent technology exist such as peer-to-peer computing, web crawlers, etc.

Multi-agent Systems (MAS) represent a promising architectural model to build web applications and distributed applications. MAS can contribute with relevant benefits, especially in highly distributed scenarios. Indeed, the autonomy and auto-organization features of mobile agents provide an excellent support for the development of flexible and dynamically adaptable systems, in which security and dependability are essential requirements. In this sense, we focus on the use of mobile agents in ubiquitous computing scenarios and ambient intelligence solutions because these are the basis for numerous applications in which dependability and security are essential features. Despite of the attention that the scientific community has paid in recent years to this field, its acceptance has not meet the initial expectations. However, this technology has been applied in several relevant scenarios if real world. We believe that this fact is motivated because security aspects play an essential role in multi-agent systems and are one of the main problems to

solve before this technology is mature to be used by the industry but this aspect is not currently appropriately solved.

Moreover, current security mechanisms for agents such as sandboxing [1], ciphering or encryption, are not enough to guarantee the necessary security, especially when we consider the scenarios of malicious hosts. Security must be considered in every stage of software life. In fact, several authors have proved that considering security as an additional or orthogonal aspect is an obsolete idea that produces systems with poor security. Other relevant aspect to consider is that the security of agent-based systems needs to be specifically adapted to these kind of systems and the technologies used to build them. The most relevant feature to be consider from our perspective is the mobility. The main goals of our work are:

- Studying the weakest security points in the current agent-based systems, both for agents and platforms.
- Analyzing current mechanisms used to provide security to agent-based systems, performing an evaluation of these, considering both security and usability aspects, which are essential for the real acceptance of new my solution.
- Designing a solution by means of which agent-based system developers can build secure agent-based systems. Moreover, the implementation of the necessary tools to facilitate the task to final users, a central are part of this objective. The analysis carried out as our second goal revealed the convenience of addressing this lines of protection: one based on software-only solutions, and another one that makes use of a hardware element called Trusted Platform Module (TPM), as these solutions complement each other and address needs of different application scenarios.
- Validating our solution. This will be performed by means of a study of the use of our solutions and tools both by means of formal analysis and by experience applying the results in real world systems.

A.3. Contribution of this work

In this section, we review the contribution of this thesis. As we previously mentioned in the main goals, which were fulfilled, we have developed two different and completely functional methodologies to protect agent-based systems.

First of these methodologies is based on the cryptographic hardware, concretely in TPM. We have developed a secure migration protocol that is the basis of this solution. This protocol makes usage of functionalities provided by this device. This protocol represents an advance to the state of the art in the agent protection. We have validated our protocol by means of "model checking" techniques, we have used AVISPA tool-suite that provides an interesting engine for protocol validation. We have proved that security aspects are not violated after a searching process of 2000 nodes in the tree generated by AVISPA. Thus, we demonstrated that protocol fulfills the expectation for which it was designed through different use cases. In order to facilitate the use of our solution we have developed the "secure migration library" integrated in the JADE platform. Our library was designed in

the way to facilitate the use of our migration protocol in the existent JADE framework with no security expertise required.

Second methodology is based in the use of "protected computing" methodology, which is completely different to the previous one and unique software based. The core of this methodology is the performance of an intelligent division of code and distribute among the agents in the system. We have developed assistant tools to implement this methodology at user level, to facilitate this task of applying security expert techniques in agent protection. Therefore, we have developed tools to split the code according to the needs in every case by means of a "protection profile" that helps in the automatic procedure. We have analyzed a wide set of possible requirements for code splitting and we have designed a set of tools to capture analyzed requirements.

We have disseminated the result of our work in the scientific community by means of certain number of publications. In the following we describe each part of the work this thesis and relate it to the publication. We present the most relevant publications that support the scientific value of this thesis. Thus, we structured this section in three parts. Firstly, we describe the most relevant contributions focused on the use of TPM. Secondly, we list the most relevant contributions related with the "protected computing" paradigm. Finally, we listed a set of contributions oriented on future applications of the results of this work.

A.3.1. Publications related with the TPM based solution

One of the key publications of this thesis is "The Role of Trusted Computing in the Secure Agent Migration", authored by Antonio Muñoz, Antonio Maña and Daniel Serrano, published in the"International Journal of Computer Science & Applications". ISSN 0972-9038. Likewise, the paper entitled, "TPM-based Protection for Mobile Agents", authored by Antonio Muñoz and Antonio Maña Gomez,published in the "International Journal of Security and Communications Networks" plays also a central role in the work presented here. This journal is especially interesting because it is indexed Thompson Reuters, in the Journal Citation Reference. These two works describe the main concepts in multi-agent based systems protection based on cryptographic hardware. The first of they is focused on enforcing security checks in the migration process. The second work is focused on the possible improvements applied to this migration method at different levels. Firstly, different alternatives of existing protocols are studied, such as the anonymous attestation protocol, which we consider very interesting in specific cases. Additionally, we present the design and validation of a new protocol based on a key-binding state that we consider the most important contribution of this work. We have added two relevant book chapters. The first of they is the "Model Cheking Ambient Intelligence with AVISPA", part of the book "Ambient Intelligent Perspectives", authored by Antonio Muñoz, Antonio Maña and Daniel Serrano, published by IO Press and edited by Peter Mikuleck, Tereza Liskov, Pavel Cech y Vladimir Bures in 2009 (ISSN 1875-4163). This chapter describes a set of mechanisms to model security solutions for its validation by means of a model checking tool. This is particularly interesting because it provides specific tools for modelling solutions in ambient intelligence scenarios. Additionally, we also highlight the chapter entitled "Verification of S&D solutions for workflows, network communications and devices", authored Carsten

Rudolph, Luca Compagna, Antonio Muñoz, and Jurgen Repp. This chapter is part of the book 'Security and Dependability for Ambient Intelligence", edited by Spanoudakis G., Maña A. and Kokolakis, and published by springer in Information Security Series (ISBN-978-0-387-88775-3), in June 2009. We consider both of them specially interesting contributions as they present tools used to perfrom a formal validation of the protocol we use as basis of our solution.

A.3.2. Publications related with the protected computing paradigm

This section presents the most relevant publications related with the solution based on "protected computing".

Among the works in this area, we highlight two main publications: the first of them is entitled "Protected Computing vs. Trusted Computing", authored by Antonio Maña and Antonio Muñoz, presented in the First International Conference on Communication System Software and Middleware, this work was published by IEEE COMSWARE'06. This work is the seed of the work in this line that is reported in this thesis. This paper poses the use of "protected computing" technique as an alternative and as a complement to the use of "trusted computing".

As we have mentioned along this thesis we raise two parallel ways to protect agents. The first of these alternatives is based in the first approach of this paper, concretely the use of "protected computing", and the second is based on the use of "trusted computing". In this paper, we described the most relevant pros and cons in the use of each of these technologies. In the same year we published the "Mutual Protection for Multiagent Systems", authored by Antonio Maña and Antonio Muñoz,included in the proceedings of the Third International Workshop on Safety and Security in Multiagent Systems (SASEMAS '06) in 2006. SASEMAS was part of the AAMAS conference celebrated in Hakodate (Japan) in May 2006. In this paper authors present the first proposal for the application of "protected computing" approach to protect agent-based systems. The attendance to this conference provided us with relevant feedback because AAMAS represents the most important conference on agents, especially considering this was a session dedicated to security aspects.

In 2007, Antonio Maña, Antonio Muñoz and Daniel Serrano published "Towards Secure Agent Computing for Ubiquitous Computing and Ambient Intelligence" in the fourth international conference of Ubiquitous Intelligence and Computing (UIC'07) celebrated in Hong Kong (China) in 2007. Our paper is part of the springer proceedings of this conference with published in the LNCS series and with (ISBN 978-3-540-73548-9). This paper presents at an abstract level the solution to protect agents based on smartprot, and we mention the division of two solutions: one solution for static systems with a predefined set of agents in the system and the second one with a dynamic number of agents in the system.

In 2009 two relevant works were published. The first of these was entitled "SecMiLiA: An Approach in the Agent Protection", authored by Antonio Muñoz, Antonio Maña and Daniel Serrano, included in the proceedings of the fourth international Availability, Reliability and Security conference (ARES), published by IEEE. The first version of the

Secure Migration Library for Mobile Agents based on cryptographic hardware is presented in this paper. In this paper we describe the most relevant libraries involved in this work using TPM functionalities and based on the JADE platform. We propose for the use of Trusted Computing technology as cornerstone for a secure solution. We actually consider this work relevant due to the fact that this is a very strong conference with an acceptance rate around of 30 %. The second paper highlighted in this year is authored by Antonio Muñoz, Antonio Maña, Rajesh Harjani and Marioli Montenegro under the title "Agent Protection based on the use of cryptographic hardware", this work is part of the proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference, COMPSAC09 celebrated in Seattle,Washington. This is one of the most significant conferences in security. Important constructive feedback was received, this enhancing results presented in this thesis.

A.3.3. Future work publication

In this section, we highlight one paper related with future works applications of our work. The paper, entitled "Agent Paradigm for Engineering AmI", authored by Amed Raian, Amir Sameh, Paolo Giorgini, Antonio Maña and Antonio Muñoz (in cooperation with University of Trento), presented in the second edition of the Ambient Intelligence Conference 2007, and included in the proceedings of this conference describes the role of mobile agent paradigm in Ambient Intelligence scenarios and shows the relevance of the security issues involved. In our opinion, this paper represents a promising start point for the practical application of our results in future computing scenarios. The proceedings were published by Springer-Verlag with ISBN 978-2-287-78543-6.

A.4. Summary of thesis work

Agent-systems can bring important benefits especially in applications scenarios where highly distributed, autonomous, intelligence, self-organizing and robust systems are required. Furthermore, high levels of autonomy and self-organization of agent systems provide excellent support for development of systems with high dependability requirements. Two main research areas that fall in this category are Ubiquitous Computing and Ambient Intelligence (AmI). Despite the attention given to agent-based systems by the research community the agent technology has failed to gain wide acceptance and has been applied only in few specific real world scenarios. Security issues play an important role in the development of multi-agent systems and are considered to be one of the main issues to solve before agent technology is ready to be widely used outside the research community. We show that solutions are available for most of these problems. There are promising technologies currently under development (in some cases in a quite advanced phase) for the remaining problems. Our view is that the main reason why agent-oriented approaches have not gained wider acceptance is the lack of appropriate application scenarios. We argue that AmI ecosystems provide one of the most appropriate application scenarios of agent-based technologies. Furthermore, agents present one of the most adequate security solutions of AmI because they facilitate concealing disparate security requirements from

different points in order to achieve each partś goals in an important collaborative setting. Of course, as mentioned above, we need to solve the most important security issues for general multi-agent systems.

Some of the general software protection mechanisms can be applied to agent protection. However, the specific characteristics of agents mandate the use of tailored solutions. First, agents are most frequently executed on potentially malicious pieces of software. Therefore, we cannot simplify the problem as it is done in other scenarios by assuming that some elements of the system are trusted. Then, the security of an agent system can be defined in terms of many different properties such as confidentiality, non-repudiation, etc. but it always depends on ensuring the correct execution of an agent on agent servers (a.k.a agencies) within the context of global environments provided by the servers. Some protection mechanisms are oriented to the protection of the host system against malicious agents, such as sandboxing and proof-carrying code. Other mechanisms are oriented towards protecting agents from malicious agencies, where one of the most recent approaches is the concept of sanctuary. Nevertheless, most of these proposals are built with the assumption that the platform where the sanctuary is implemented is secure. Unfortunately, for agent-based systems this assumption is not applicable.

Several techniques can be applied to an agent to verify self-integrity avoiding that the code or the data of an agent is inadvertently manipulated. Anti-tamper techniques, such as encryption, checksum, anti-debugging, anti-emulation and some others [70] share the same goal, but they also are oriented towards the prevention of analysis of the functions an agent implements. Some protection schemes are based on self-modifying code and code obfuscation. In agent systems, these techniques exploit the reduced execution time of the agent in each platform. Software watermarking techniques [70] are also interesting. In this case the purpose of protection is not to avoid the analysis or modification but to enable the detection of such modification. The relation between all these techniques is strong. Indeed, it has been demonstrated that neither perfect obfuscation nor perfect watermark exists [76]. In summary, all these techniques provide short-term protection; therefore, in general they are not applicable for our purposes. However, in some scenarios, they can represent a suitable solution, especially, when combined with other approaches. Theoretic approaches to the problem have been demonstrated that self-protection of the problem is unfeasible [71].

In some scenarios, the required protection is limited to some parts of software (code or data). In this way, a function performed by the software, or the data processed, must be hidden from the host where the software is running. Some of these techniques require an external offline processing step in order to obtain the desired results. Among these schemes, function hiding techniques allow the evaluation of encrypted functions [33]. This technique protects the data processed and the function performed. We consider that as an appropriate technique for protecting agents. However, it can only be applied to the protection of polynomial functions.

Another interesting approach is the case of online collaboration schemes where part of software functionality is executed on one or more external computers. The security of this approach depends on the impossibility of each party to identify the function performed by the others. This approach is very appropriate for distributed computing architectures such as agent-based systems or grid computing, but has the important disadvantage of

the impossibility of its application to off-line computing scenarios.

This doctorate thesis advocates techniques that provide a two-way protection. Some of these are hardware based, such as based on Trusted Computing Platform. The need for a secure platform has become even more evident with recent advances of ubiquitous computing. Therefore, this approach adds a trusted component to the computing platform, usually built-in hardware used to create a foundation of trust for software processes. Other techniques are software based, by means of Protected Computing [77]. The last is based on partitioning software into two or more dependent parts so that part of the code is remotely executed on a different agent.

In the rest, we will discuss in some details the two approaches, that is, based on the use of cryptographic hardware and based on the use of protected computing to address most relevant security issues in mobile agents. Cryptographic hardware based approach. A concrete cryptographic hardware is based on TPM. We designed and developed a Secure Migration Library (SecMiLiA) in order to provide the secure migration functionality with friendly use of its mechanisms. We based SecMiLiA on the JADE platform because of its widespread usage in the agent community and because of the well-defined inter-platform migration mechanism provided by JADE.

The most important objective of the design of this library is provision of a secure environment for secure agents execution and migration. An important design feature is the easy integration with JADE, that is, no modifications to JADE are necessary to be done. The aim is to facilitate software developers, who are not security experts, with the provision of a library that conforms with existing security solutions. We achieved to develop a library that provides security to software agents on JADE. We developed a mechanism enabling secure migration of agents. The mechanism is based on testing the trust of destination agency before the migration process actually takes place. This guarantees that an agent execution is always performed on a secure environment avoiding the problem of malicious (distrusted) hosts. Thus, an agent reaches a secure environment where its execution goes on, in a way that agents cannot modify the host agency.

Protected computing based approach. The basic idea is to divide an application code into two or more mutually dependent parts. Some of these parts (which we call private parts) are executed in a secure processor, while others (public parts) are executed in any processor even if it is not trusted. A detailed description of this technology is presented in [78]. We apply the protected computing scheme to protect a society of collaborating agents by making every agent collaborate with one or more remote agents running in different (distrusted) hosts. Because agents run in potentially malicious hosts, the goal in this scenario is to protect agents from potential attacks of hosts. The basic idea is to make agents collaborate, not only in the specific tasks they are designed to perform, but also in the protection of other agents. In this way each agent acts as a secure coprocessor for other agents.

Using the protected computing model, the code of each agent is divided into public and private parts. For the sake of simplicity, and without loss of generality, we will consider the simplest case where the code of each agent is divided in two parts: a public one and a protected one. From this description, it is easy to derive the possibilities code division into more parts. In particular, the division of multiple private parts executed in different coprocessors, is especially relevant for the scenarios that we target. The private part of

each agent has to be executed by another agent in another host. The scheme is suitable for protecting a set of several mutually dependent agents. A conspiracy of all hosts is necessary in order to attack the system.

To split the code into parts a developer uses an automatic Code Partitioning Tool (CPT). Since code partitioning is a difficult task and specific expertise is required for performing it, such a tool eases the code partitioning according to a set of rules that we call protection profiles. The outcome of the operation of this tool is a set of public parts and a set of private parts. These parts will be used in a different way depending on the mutual protection scheme applied (static or dynamic). One can protect code by means of data protection, i.e. by marking the data to be protected. For this purpose the tool allocate the instructions to protect taking into account java labels such as final, static, etc. One can protect code by means of instructions. In order to do that the tool has a classification of instructions and a developer marks the type of instructions to be protected (or a group of them). A combination of both techniques can be applied if the code requires such. A developer sets up the protection profile and adjusts it to get the more suitable configuration for his requirements. Another important approach is marking the code to be protected while it is being developed, rather than performing the configuration by means of a protection profile. One could even select parts of code to be protected. Then the Java compiler marks those byte codes, by developer command, adding some annotations. A developer according to the code and context restrictions must command data protection. However, instructions protection is suitable to be automated. For this purpose we grouped the byte code instruction set, currently consisting of 212 instructions. Strategies for securing agents. There are two distinguished strategies for securing agents. A Static Mutual Protection (SMP) strategy defines collaboration between agents in a predefined manner so that every agent has the private parts of code of one or more agents it is collaborating with. A Dynamic Mutual Protection (DMP) strategy defines any of the collaborating agents to play the role of a secure coprocessor to any other agent. In this case, interactions between agents are not predefined. This DMP strategy is more powerful and flexible than the SMP strategy, but it also adds more complexity and reduces performance of agents.

The SMP strategy increases the performance of agents by avoiding the transmission of protected code sections over the network. It is suitable in those scenarios where the set of agents to be protected is static and determined before their actual execution. An example of possible application of this scheme is that of a competitive bidding. In this scenario a client requests bids from several contractors to provide a good or a service. It is important that the bidding takes place simultaneously, so that none of the contractors can access the offer from the other contractors, because this would give it advantage over the others. The client can use several single-hop agents to collect the offers from the contractors. Every agent will protect each agent, using the SMP strategy.

We can also safely assume that a coalition of all contractors will not happen. In fact, no technological solution can prevent all contractors to reach an external agreement. Because each agent is protected by other agents running in the hosts of the competitors, and because the protected computing model ensures that it is neither possible to discover nor to alter the function that the agents perform and it is also impossible to impersonate the agents. We know that all agents will be able to safely collect the bids, guaranteeing

the fairness of the process. There are scenarios where it is not possible to foresee possible interactions between agents, or where agents are generated by different parts, or involvement of very dynamic multi-hop agents. In these cases the DMP strategy is more appropriate to apply where each agent will be able to execute arbitrary code sections on behalf of other agents in the society.

We have defined a Dynamic Protection Tool (DPT) that allocates into every agent a little virtual machine code. This virtual machine executes public and private code from other agents on the fly. Doing this, there will be no necessity of fixed assignations between agents because every agent is a potential secure processor for the rest of the agents in the system [49]. Ongoing work is focused on two directions:

- Prototyping automatic tools for code partitioning driven by a policy specification. The focus of this direction is on flexibility and adaptability of those tools to different parameters in policies.
- Expanding the dynamic mutual protection approach in the context of DPT. The main task here is to identify problems rose when put together concepts of dynamism, virtual machine and agent private parts. This is because possible deadlocks or other problems could rise from this paradigm.

A.5. Achievement of objectives

Main objectives of this thesis are described as follows:

- Study of the requirements, restrictions and weak points of agent-based systems related to the security. We perform this study from the points of view of both the agents and the agencies.
- An analysis of current mechanisms to provide security for agent-based systems. We evaluated these mechanisms taking into account security and usability aspects, which we consider essential for the practical acceptance of agent systems.
- Design of solutions for building secure agent-based systems, along with the necessary tools to facilitate the use of these solutions by non-specialist developers.
- Validation of our solution by means of formal analysis of our solutions, and by applying our solutions and tools in real world agent-based systems.

The first and the second objectives have been achieved in parallel since the study of current mechanisms have helped to identify the weak points of these mechanisms. Chapter 2 contains a detailed description of all existing mechanisms and a description of the weak points of these mechanisms, thus illustrating the achievement of both objectives.

Two alternatives have been considered in this work in order to achieve the main objective. The first alternative based on using TPM hardware, and the second one is based on the "protected computing" model. We believe that these two complementary solutions fulfill our objective. In fact, both solutions have been implemented to be used in real agent-based systems as described in chapters 3 and 4. The tangible result of the first

solution is a library that implements the secure migration protocol designed. The implementation of the second solution in which a priori presents more difficulties has also been realized by a set of tools designed to facilitate the application of this technique by average developers.

A.6. Conclusions

We conducted a detailed study of the problem of protecting agent-based systems, looking at different aspects, we have described the objectives to be covered in each case. The solutions developed focus on two different approaches: one for providing a secure solution based on the use of an external element and the other proposes a solution that uses of the capacity for collaboration between agents to solve a problem, as the basis for ensuring the security, as described in Chapter 2. In this thesis we show how both mechanisms can be used in different scenarios for the protection of the global system.

We have also conducted an analysis of different existing protection mechanisms at both academic and industrial levels, and we have classified them based on the nature of the mechanism itself. We have shown how existing solutions are inadequate, in most cases because they are based on the direct application of solutions designed for software in general and not designed for multi-agent systems. In other cases we have observed that the application of existing solutions requires that the developers have an exhaustive understanding of security mechanisms. As part of the review of relevant work, we have noted that an alternative to ensure the protection of multi-agent systems is based on trusted computing platforms. In particular, we have worked with the most relevant initiative in this field: the one represented by the Trusted Computer Platform Alliance. That approach has led to the design of the solution presented in Chapter 2.

For the validation of this solution we have decided to take two alternative paths, which provide a higher level of robustness of the solution. First we performed a formal validation of the secure migration protocol of our solution. We have done so using a powerful set of tools known as AVISPA. On the other hand, we have applied the solution to a real world use case and have designed a system that functions as a nuclear power plant simulator in which we have built an agent-based system that monitors the proper functioning of each of the parts of the system. Obviously, this scenario requires a high level of security.

A.7. The role of agents in Ambient Intelligence

Notebooks, PDAs and smart phones all have some sort of wireless connection that allows access to various data networks virtually anytime anywhere. The evolution in size and capabilities of these devices and wireless communications has enabled the permanent connection of users to the network. This increased mobility may mimic what happened years ago when users switched from classic desktop computers to laptops. The next step would be to help people to be able to sense and take advantage of the existence of other machines in the environment [63]. The expectations for the new software are to be able to handle concepts such as location and perception of context, personalization, adaptabi-

lity, organic growth, mobility and many other features which will require more complete software engineering methods and new software innovative modeling languages [74].

Ambient Intelligence (AmI) is focused on adapting our environment in a way that is sensitive to our needs and responds intelligently to users behaviour and to changing contexts [64]. It is expected that the objects that surround us in our daily lives act autonomously without human intervention. This type of ambient intelligence environments are intimately related to the concept of ubiquitous computing, as it is intended to provide people services and utilities in a transparent way, without explicit requests from users. The agent paradigm is very promising for the implementation of complex systems such as electronic commerce, air traffic, enterprise resource planning, etc. [66]. Initially developed within the field of Artificial Intelligence, the scientific community has found other areas in which fruitful exploitation of the paradigm of the agent is possible. As advocated in [67] and [68], this paradigm has received a special interest in software engineering because it provides advantages over the paradigm of object orientation in distributed and autonomic computing scenarios.

One of the weaknesses of Ambient Intelligence environments is the lack of models and software engineering practices that aid in the analysis of requirements, design, verification, testing, etc. So far research in this area is in its early stages, and the need for viable development methodologies is becoming widely recognized. We believe that the agent paradigm is useful not only for the implementation of AmI systems, but also in all phases of the engineering and development cycle. We are aware that the new methodologies of agent-oriented software engineering should be further reviewed and extended for engineering Ambient Intelligence systems.

However, despite the attention given to agents by the research community, this technology has failed to gain wide acceptance, especially in industry, and has only been applied in very specific and limited real world scenarios. This comes largely from the existence of certain security issues, which indeed are considered the most urgent difficulty to solve in this technology before it is ready to be used in the industrial landscape. Regarding security, the agents exhibit interesting features and can cover security requirements coming from different actors, with a natural model for achieving the objectives of each party in a collaborative way. Clearly, this technology will not be mature enough to be used until the most important security problems are solved. This consists both in finding the appropriate mechanisms, as well as in facilitating the application of these mechanisms.

Traditionally, some of the protection mechanisms for general software systems have been applied to the protection of agent-based systems. However, the special features of this technology requires the use of specific solutions. Some protection mechanisms are oriented to the protection of the hosts (agencies) against malicious agents. The most well-known solution to this problem is the use of the concept of Sandbox. A Sandbox is a container that limits or reduces the level of access that agents have and provides some mechanisms to control the interaction between them. Another technique that addresses this aspect of the problem is known as Proof Carrying Code (or PCC) [69], with this technique, each code fragment includes a detailed test that can be used to determine if the security policy the host is satisfied by the agent. The hosts need only to verify that the test is successful (which means it matches the code) and is compatible with the local security policy, but we must mention is that the application of this technique is not straightforward since

determining the necessary policy to enforce is frequently very difficult.

Other mechanisms are designed to protect the agency from potentially malicious agents. The most representative technique is known as Sanctuaries[40]. A Sanctuary is analogous to a Sandbox and consists in an execution environment in which mobile agents can be safely executed. This proposal bases its security on the assumption that the platform where the Sanctuary resides is safe. Unfortunately, assuming this does not work in practice for agent-based systems, since we can not control all platforms where the agents will run.

On the other hand, several techniques can be applied to agents for verifying their own integrity. This idea is useful to prevent that both the code and data of the agent can be manipulated inadvertently. This set of anti-manipulation techniques include others such as encryption, checksum, anti-debugging, anti-emulation, and so on [70], which ultimately share the same objective but are also aimed at preventing the analysis of the function that the agent implements. All these techniques have a fairly high degree of difficulty in their application, especially for average developers, mainly due to the fact that the application of these techniques involves some specialized knowledge of security.

Once we have presented several existing alternatives, we must highlight the fact that all the techniques described provide short term protection, in addition to requiring a detailed knowledge of security issues. Therefore, they do not meet our needs and can be considered useless for our purpose. However, in certain scenarios, they may represent a feasible solution, especially combined with other solutions. Theoretical studies of this problem, have proved that self-protection of code is not viable, ie can not ensure the security of a solution that is based only on software for protection [71]. In some scenarios, the protection required is limited to some parts of the software (code or data). Thus, the function implemented by the software or data must be hidden from the host on which the software is run. Some of these techniques require an additional external processing step to get the desired results. Among these schemes, some techniques allow the evaluation of encrypted functions [33], thus proving itself as an appropriate technique for the protection of agents. However, in practice the approach is only applicable for the protection of polynomial functions.

The case of online collaboration schemes is also very interesting. In these schemes, some of the functionality of the software runs on one or more external computers. The security of this solution depends on the inability of each party to identify the function performed by others. This solution is well suited for distributed computing architectures such as agent-based systems or grid computing.

Finally, there are techniques that create a bidirectional protection, protecting both the agent and the agency. Among them some are hardware-based solutions, including some that use the Trusted Computing Platform as a basis [17]. In fact, with the recent emergence of ubiquitous computing, the need for a secure platform has established itself as an obvious fact. This solution adds a reliable component of the computing platform, which normally consists of some hardware built into the motherboard of the computer and used to create trust in the software processes [72]. Other techniques are based solely on software features, such as making use of the protected computing paradigm [49], which is based on the partition of software elements into two or more parties that are mutually dependent, In this way part of the code will be executed remotely by different agents.

A.8. Aplication of the protection mechanism to the protection of software in *Cloud Computing*

Cloud computing is a paradigm that allows computation through the Internet. Evidently, the cloud is a metaphor of the Internet with some particularities.

In this kind of computation all that a computer system can offer is by means of a service, in such a way that users can access to available services in the "cloud", this access creates an abstract level in such a way that users do not need to know how the cloud is implemented is any kind of expertise required to manage the resources offered. According to IEEE Computer Society, the *cloud computing* is a paradigm in which the information is permanently stored in servers and is sent temporarily to clients caches, which includes desktops, entertainment platforms, mobile telephones, etc.

Cloud computing is a new concept that includes software as service, such as the Web 2.0 and other newer, also known as technological tendencies, which the common point is the trust in Internet to satisfy the user computation requirements. Among current *clouds computing* implementations we highlight the Amazon EC2, Google Apps, eyeOS and Microsoft Azure, which provides common online business applications accessible from the web browser while data and software are stored in servers.

Cloud computing concept started with highly scalable service providers such as Google, Amazon, and other that built their infrastructure. A new architecture arose, that is a horizontally distributed resource system as IT virtual services massively escalated and managed as continuously configured resources. This architectural model was described by George Gilder in "Information manufactures". Gilder wrote about these server farms comparing them to grids. However, while grids are used for technical computation applications (a system composed of several partially autonomous subsystems while continuously interlinked with other components) this cloud model is applied to Internet services.

Both clouds and grids are designed to be horizontally scalable efficiently. Both models are built to resist fails in individual nodes or elements. However, while grids process tasks in batch, with a starting and ending point, cloud services can be continuous. Moreover, clouds expand the kind of resources available, file storing, data base and web services to the web applicability and company particular applications.

One of the most relevant features of *cloud computing* is that does not allow users to physically keep storing devices of their data (only one exception exists consisting in copying data to an external storing device, such a flash USB or external hard disk), the user delegates data storing responsibilities and control the provider.

Cloud computing has been harshly criticized for limiting user liberty and establishing a dependency on the service provider [75]. Some critics state that only those applications and services that provider offer are available to users. London Times compares "clouds computing" with centralized systems from 50's and 60's, where users connected to from dump terminals to central computers. Generally, users were not allowed to install new applications, only under administrator approval. To summarise both liberty and creativity were highly limited. The Times states that "clouds computing" is a way back to that age. Richard Stallman, founder of Free Software Foundation, defends that "clouds computing" endangers user freedom since these leave their data and information in the hands of third

parties. Richard states that clouds is only a trap to force people to acquire proprietary blocked systems, which will be increasingly more and more expensive.

However, there are a lot of advantages to this new technology, especially if we plan a further step in the evolution of this technology. Hence "clouds computing" consists of executing a set of data remotely by the clouds with the consequent user trustworthy delegation in the cloud. Nevertheless, this model is restricted to functionalities offered by provider and the user can only can execute those pieces of software included in the cloud. We propose a step further allowing the user to send their own software establishing a parallelism with agent technology, which are computation entities able to encapsulate any code. Obviously, this new feature removes the limitations of current clouds criticized by Stallman but the most important appeal is that the approaches in this thesis are perfectly suitable to solve most of the security problems of this new model due to the fact that the problem is a piece of code that is executed remotely in an external host.

A.9. Alternative and complementary technologies

Throughout this thesis we have presented two approaches that face the initial problem from two different perspectives. The first approach is based in the use of a cryptographic hardware and the second one considers the application of "protected computing" technology as we describe in depth in chapters 3 and 4.

Concluding chapters 3 and 4, where both approaches are fully described, a section dedicated to conclusions and future work is included. However, once a global study of the problem has been performed we propose the use of both approaches simultaneously since they are complementary and we can achieve a higher level of security. The idea is based on the straight application of the "protected computing" approach, but instead of creating a trusted ring with all agents of the system as we describe in chapter 4, the idea is to establish a code partitioning of agents in two different parts public and private. The public part is executed regularly but the private one must be executed in the trusted hardware, in addition our case the *tpm*.

Evidently the application of this combined technique implies further disadvantages because when combining the two approaches we also import and combine the worst traits of each. Nevertheless, we think that a deeper study of the combination of both techniques is highly interesting, especially for those systems that require a high level of security but this is out of the scope of the present thesis.

Apéndice B

Método de Trabajo

En ciencias tradicionales como son la física, la biología o la medicina, se cuenta ya con una serie de metodologías de investigación. Sin embargo, en otras disciplinas más recientes como es el caso de la computación, la ingeniería del software y la ingeniería de la seguridad, en las que el desarrollo de la investigación data de pocas décadas [2] no ocurre lo mismo. Aunque hemos de mencionar que se está trabajando de forma intensa en esta línea. En este capítulo vamos a describir el uso de dos métodos de trabajo para lograr los objetivos planteados en el inicio de esta tesis. El primero de ellos es el Método Investigación-Acción (Actions-Research) por ser éste uno de los principales métodos de investigación cualitativa en el campo de los sistemas de información en la ingeniería del software [3] y la propuesta de Kitchenham para la investigación sistemática de la literatura [4].

B.1. Investigación-Acción

En el año 1946 el investigador Kurt Lewin publica su trabajo titulado "Action Research and minority problem" en el cual cita por primera vez el concepto de la Investigación-Acción (IA). Lewin describe este método mediante un proceso de espiral de varios pasos cada uno de los cuales está compuesto de los ciclos planificación, acción y búsqueda de hechos acerca de los resultados de la acción [5]. Este método ha evolucionado considerablemente en los últimos años, en gran medida por la calurosa aceptación por parte de la comunidad científica. La Investigación-Acción consiste en una clase de métodos que tienen en común las siguientes características:

1. Orientación a la acción y al cambio.
2. Identificación de un problema.
3. Un modelo de proceso orgánico que engloba etapas sistemáticas y algunas veces iterativas.
4. Colaboración entre los participantes.

La Investigación-Acción está orientada a la generación de cierto conocimiento de investigación relevante [6]. Es una forma colaborativa de investigación que pretende unir

teoría y práctica entre investigadores y profesionales mediante un proceso de naturaleza cíclica.

En el campo de los Sistemas de Información, el cliente de una investigación suele ser una organización para la que trabaja el mismo investigador [6]. El investigador que utiliza la metodología de la Investigación-Acción en el contexto de Sistemas de Información (IA-SI) cumple un doble servicio, para el cliente de la investigación y para la comunidad científica de Sistemas de Información. En múltiples ocasiones ocurre que las necesidades de ambos suelen ser muy diferentes, e incluso opuestas entre sí. El objetivo primordial de cualquier investigador es intentar satisfacer ambas demandas.

B.1.1. Etapas de la investigación-acción

La Investigación-Acción viene caracterizada por una serie de etapas, tal y como describe Padak en [7]:

- La planificación consiste en identificar la metodología adecuada que servirá como guía para la investigación. Estas deben estar directamente relacionadas con el objetivo de la investigación. Evidentemente en muchas ocasiones, en esta actividad se buscan caminos alternativos, o líneas a seguir para reforzar lo existente. El resultado es que se definen claramente otros problemas o situaciones a tratar.

- La acción es la variación de la práctica cuidadosa, deliberada y controlada. Se efectúa una simulación o prueba de la solución. Concretamente se refiere a la intervención del investigador en la realidad.

- La observación se describe como la recolección de la información. Esta puede ser mediante la toma de datos, o bien mediante la documentación de los sucesos, etc. La información puede proceder de cualquier sitio (bibliografía, medidas, resultados de las pruebas, observaciones, entrevistas, documentos, etc).

- La reflexión consiste en compartir y analizar los resultados con el resto de interesados, de tal manera que se invite al planteamiento de nuevas cuestiones relevantes. En algunas variantes de IA la reflexión no es una etapa realmente, sino un proceso continuo que dura todo el periodo, como ha ocurrido en nuestro trabajo de tesis.

El hecho de que la Investigación-Acción sea un proceso iterativo permite que se vayan obteniendo soluciones cada vez más refinadas al final de cada ciclo, además de proporcionar medios para poner en práctica nuevas ideas que son comprobadas en el ciclo siguiente. Esta característica permite la evaluación de los caminos seguidos y el estudio de nuevos caminos a tomar.

A pesar de las ventajas de esta metodología. La Investigación-Acción presenta ciertas limitaciones. Se han identificado varios problemas en su aplicación que tienen tres causas fundamentales:

- El principal es la falta de método con que los investigadores y profesionales utilizan y conciben la IA-SI, que evidentemente no se aplica de forma adecuada.

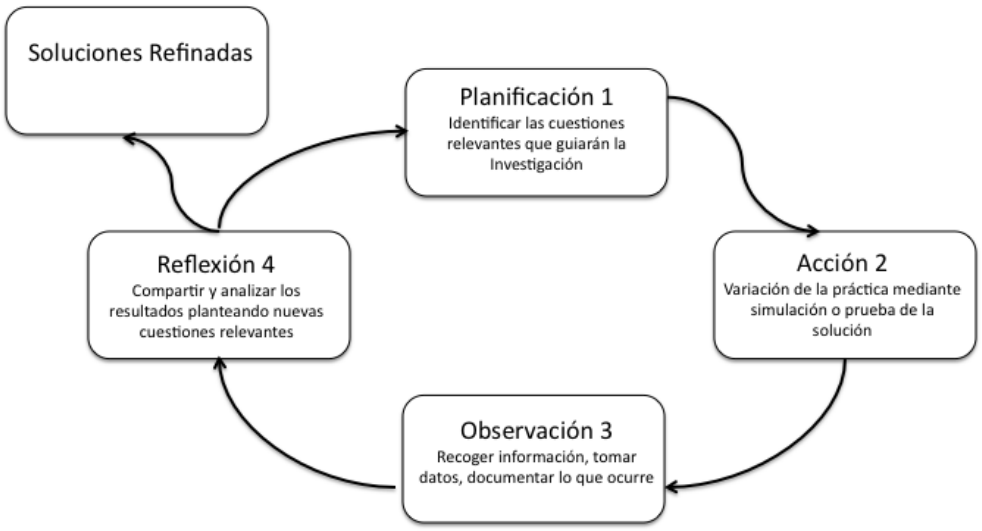

Figura B.1: Carácter cíclico de la Investigación-Acción

- Otro problema es el del contexto de consultoría utilizado, ya que impone una perspectiva demasiado restrictiva por la implicación de responsabilidades contractuales e intereses organizacionales que pueden ir en contra de lo propuesto por la misma IA.

- Por último, la falta de la definición de un modelo de proceso de investigación detallado que indique los pasos a seguir en la IA-SI.

Estas tres causas llevan asociada una consecuencia inherente que es una falta de rigurosidad del proceso de investigación, que es inadmisible. Para resolverlo se han propuesto las siguientes alternativas:

- Llevar la investigación siguiendo una perspectiva de gestión de proyectos en cada una de las fases típicas de esta metodología. Estay [8, 9] propone el uso de técnicas de gestión de proyectos para mejorar el rigor de un proyecto del IA-SI, lo cual se ha traducido en generar una estructura de proyecto que contenga los principales elementos de IA-SI. Para Estay y Pastos [9] IA y proyecto son conceptos equivalentes, ya que ambos son experiencias de trabajo únicas con resultados finales igualmente únicos y, además, comparten la idea de intervención, es decir, ambos suponen una alteración voluntaria de la realidad.

- Incluir criterios de calidad especialmente concebidos.

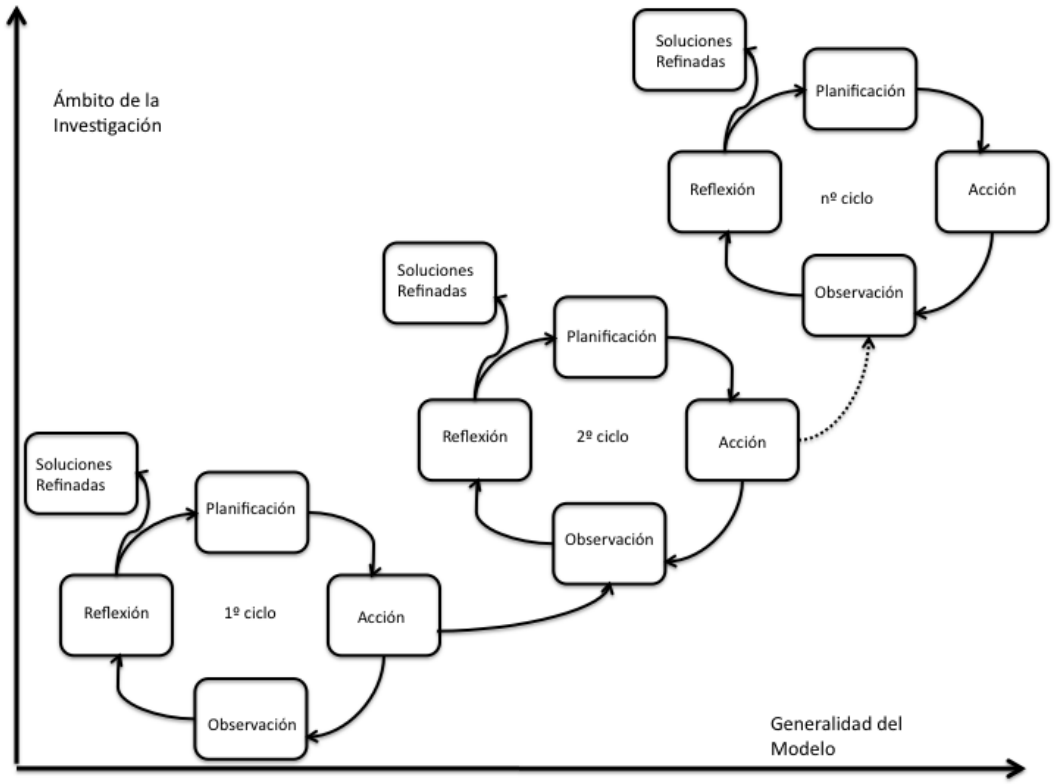

Figura B.2: Esquema de Agencias Distribuidas en una Red

- Analizar los factores que inciden en la formalización del proceso.
- Organizar el proceso con una estructura de proyecto.

B.1.2. Ciclos de la investigación

La investigación de calidad de los sistemas de información hoy en día se enfoca desde dos perspectivas distintas, por un lado la académica y por otro la práctica. Ambas están en continua interacción pero se mueven en planos diferentes. Por esta razón la Investigación-Acción de los sistemas de información opera sobre esta realidad dual, en concreto se centra en dos ciclos distintos para dos tipos de proyectos:

- Ciclos orientados a resolver problemas dentro de proyectos de Sistemas de Información. Estos proyectos consisten en el desarrollo de una solución informática (son proyectos informáticos, de desarrollo de software, de implantación y/o mantenimiento de sistemas informáticos, etc.). En este caso el investigador se encarga de resolver un problema e Investigación-Acción aparece como una herramienta adicional para el desarrollo de sistemas de información.
- Ciclos orientados a investigar dentro de proyectos de investigación. Estos proyectos son esfuerzos por buscar un resultado concreto, para lo que la metodología de la IA nos ofrece un método de trabajo y una justificación para acercarnos a un caso real con el fin de probar una teoría o hipótesis.

Existe otra propuesta de proyecto de IA-SI en la que se definen dos ciclos característicos:

- Ciclo orientado a construir una solución para generar nuevo conocimiento útil a profesionales y mejorar su práctica. El investigador interviene activamente en la realidad y se utiliza la investigación para construir modelos, teorías o conocimiento de manera informada y evidentemente influida por la realidad. En este ciclo, el interés por resolver un problema es lo que origina el interés por la investigación.

- Ciclo orientado a gestionar la investigación para producir nuevo conocimiento a la disciplina de SI y mejorar de esta forma la práctica de los investigadores. En este ciclo el que origina la inquietud por resolver ciertos problemas es el interés por la investigación.

La Investigación-Acción aplicada a los sistemas de información se puede analizar desde dos dimensiones complementarias. En primer lugar una dimensión vertical en función del tipo de proyecto. Y en segundo lugar, una dimensión horizontal en función del bi-ciclo típico de la estructura de un proyecto de IA-SI.

En 1997 Lau [10] presenta un resumen del uso de IA-SI, comentando diversos ejemplos publicados por diferentes autores referidos a la construcción y desarrollo de sistemas de información, y más especialmente, al análisis, diseño, desarrollo e implementación de software y a los procesos asociados.

Baskerville en 1999 [11] hace una introducción al uso de IA-SI indicando varias formas de utilización de la metodología. Además de describir cuatro características fundamentales que determinan dicha forma de uso: modelo de proceso (iterativo, reflexivo, linear); estructura (rigurosa, fluida); rol del investigador (colaborador, facilitador, experto); y objetivos principales (desarrollo organizacional, diseño de sistemas, conocimiento científico, entrenamiento).

B.2. Revisión Sistemática de la Literatura

Cualquier investigador que se precie debe realizar una revisión de la literatura de forma sistemática al comenzar un nuevo reto. En la mayoría de las ocasiones la revisión en sí no es el objetivo de la investigación aunque representa la mejor forma de documentarse con respecto al estado del arte del tema a abordar. Además, un estado del arte constituye la base para la formulación de propuestas más relevantes.

Debido al comienzo tan reciente de las disciplinas científicas relacionadas con la computación, no existen metodologías que sirvan para guíar el desarrollo de las revisiones sistemáticas en las mismas investigaciones de forma específica. Kitchenham [4] propone un método para realizar revisiones sistemáticas, es este mismo autor quien acuña el término *revisión sistemática*. Este método se basa en pautas desarrolladas para la investigación médica y que fueron adaptadas para usarse por un equipo de investigadores en el ámbito de la ingeniería del software. Una revisión sistemática se define como "una manera de evaluar e interpretar toda la investigación disponible relevante respecto de un interrogante de investigación particular, en un área temática o fenómeno de interés" [4].

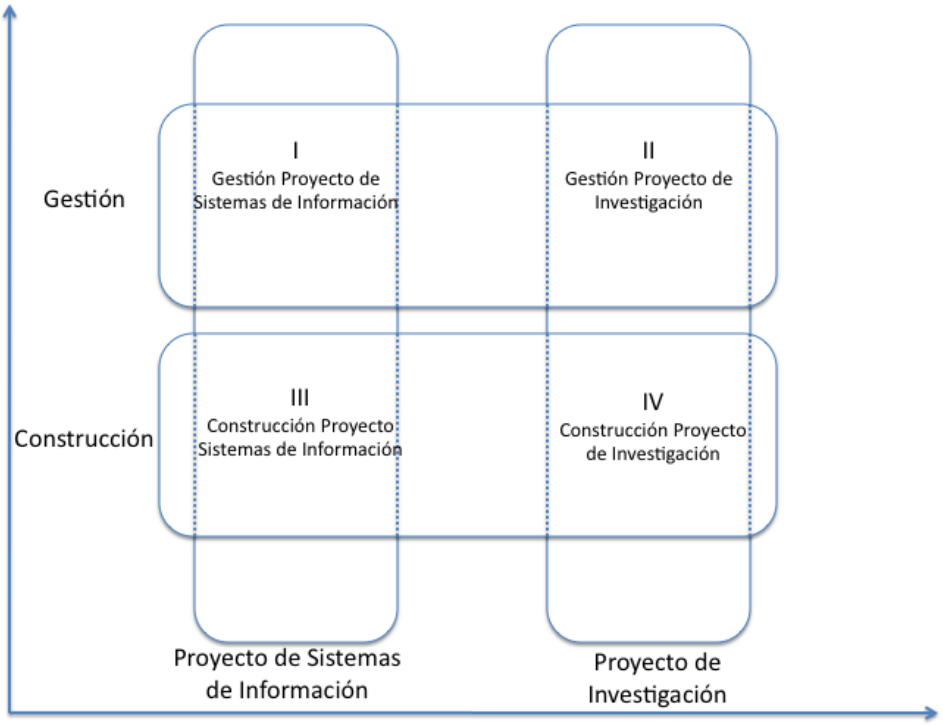

Figura B.3: Dos dimensiones de la Investigación-Acción en Sistemas de Información

Los estudios individuales que contribuyen a una revisión sistemática se denominan estudios primarios, en cambio, una revisión sistemática se considera un estudio secundario. En particular este método propone tres etapas fundamentales que son una planificación de la revisión, el desarrollo propio de la revisión y la publicación de los resultados de la revisión, que a su vez se encuentran divididas en otras etapas que detallan la forma en que se deben desarrollar.

Etapa 1: Planificación de la revisión
Identificación de la necesidad de revisión
Definición de un protocolo de revisión
Etapa 2: Desarrollo de la revisión
Identificación de la Investigación
Selección de los estudios primarios
Evaluación de la calidad del estudio
Extracción y seguimiento de datos
Síntesis de datos
Etapa 3: Publicaciones de los Resultados

Cuadro B.1: Tabla de Planificación de la revisión.

La propuesta de inicial de Kitchenham con la adaptación de Caro [12] se muestra en

el cuadro B.2. Para el caso particular de un solo investigador, en que la supervisión está a cargo de un tutor con evidentes limitaciones de tiempo.

Etapa 1: Planificación de la revisión (en sección B.2.1)
Identificación de la necesidad de revisión
Definición de un protocolo de revisión
Etapa 2: Desarrollo de la revisión (en sección B.2.2)
Identificación de la Investigación
Selección de los estudios primarios
Evaluación de la calidad del estudio
Extracción y seguimiento de datos
Síntesis de datos
Etapa 3: Publicaciones de los Resultados (en sección B.2.3)

Cuadro B.2: Tabla de Planificación de la revisión por secciones.

Una vez descrito el método de Investigación-Acción vamos a mostrar su aplicación en el desarrollo de esta tesis.

B.2.1. Etapa 1: Planificación de la revisión

Esta etapa tiene como propósito específico definir los parámetros más importantes que se tuvieron en cuenta en el momento en que se realiza la revisión. Se deben establecer las razones que justifican llevarla a cabo, la manera en que se hará la búsqueda de trabajos y la forma en que éstos serán revisados. Finalmente, se evaluará la planificación realizada. Esta etapa se ha dividido en las siguientes sub-etapas:

Identificación de la necesidad de la revisión

La necesidad de una revisión sistemática surge de la necesidad de un investigador por recopilar, de manera rigurosa e imparcial, toda la información existente sobre algún fenómeno de interés. El objetivo de dicha recopilación es iniciar otras actividades de investigación futuras. Antes de emprender una revisión sistemática, el investigador se debe de asegurar de que ésta es necesaria. En particular, es recomendable identificar y analizar cualquier revisión sistemática existente acerca del fenómeno de interés con un criterio de evaluación apropiado. Con este fin, resulta conveniente utilizar listas de verificación que contengan cuestiones como las siguientes:

Según Kitchenham [4] las razones más frecuentes que justifican la necesidad de una revisión sistemática son:

- Resumir la evidencia existente concerniente a una tecnología.
- Identificar algún vacío en la investigación actual con el objetivo de proponer nuevas áreas para investigaciones futuras.

Encontrar los objetivos de la revisión.
Fuentes usadas para la identificación de estudios primarios.
Los criterios que se incluyeron o excluyeron y la aplicación de estos.
Los criterios para evaluar la calidad de los estudios primarios y la aplicación de estos.
La extracción de los datos de los estudios preliminares.
La sistematización de los datos.
Hacer una diferenciación de los diferentes estudios investigados.
Realizar una combinación de los datos.
Cuestionarse si es razonable combinar los estudios.

Cuadro B.3: Tabla de cuestiones para una revisión sistemática.

- Proporcionar un marco de trabajo y/o los antecedentes necesarios con el objeto de posicionar nuevas actividades de investigación.

Haciendo una recopilación de todo lo expuesto hasta el momento, se pueden identificar claramente los recursos con los que se cuenta al iniciar la revisión (tales como Internet, revistas electrónicas de acceso público o restringido, actas de congresos, etc.), ya que esto puede variar a medida que se avance en la investigación. De hecho, es posible que se pueda contar con nuevos recursos, como pueden ser el acceso a investigaciones recientes, una nueva suscripción o incluso la adquisición de libros. En el caso particular de esta tesis se ha contado con una conexión permanente a Internet de banda ancha, que ha facilitado el acceso a múltiples recursos, entre los que destacamos el acceso a revistas electrónicas de prestigio por la suscripción pertinente de nuestro grupo de investigación. También consideramos interesante mencionar los medios proporcionados, gracias a la participación en los proyectos mencionados en la introducción, que nos han permitido realizar viajes y estancias en universidades y centros de investigación muy variados, que sin duda han enriquecido encarecidamente el resultado de esta tesis.

Definición del protocolo de búsqueda

En esta sub-etapa se deben definir las normas que seguirá la investigación con respecto al proceso de búsqueda en las fuentes de información definidas en la sub-etapa anterior. El protocolo de búsqueda debe contener una definición de los términos que se buscarán, las combinaciones de éstos, la estrategia de búsqueda empleada dependiendo de la fuente y la manera en que se registrarán los resultados. En nuestro caso se realizó una búsqueda de cualquier mecanismo existente para la protección de agentes, haciendo una posterior extensión a mecanismos genéricos de protección de software aplicables al paradigma del agente.

En relación con la estrategia de búsqueda, es importante establecer la manera en que se va a proceder con respecto a cada fuente empleada. Un caso particular es Internet, que está jugando un rol cada vez más importante en cuanto a la accesibilidad de la literatura científica [13]. Debido a que la información disponible en Internet es muy abundante, es necesario establecer criterios para hacer filtros que permitan obtener sólo aquella informa-

ción que sea realmente útil y de calidad. Se recomienda hacer un registro de los resultados de las búsquedas. Esto puede servir para justificar la necesidad de investigar en algún área específica, o para demostrar, cuantitativamente, que los trabajos de una determinada área son escasos, o que son muy heterogéneos, o simplemente para demostrar la rigurosidad con que se ha realizado el proceso de búsqueda.

Finalmente, es importante tener en cuenta que el proceso de búsqueda es perfectible, por lo que el protocolo puede y debe ser mejorado durante el desarrollo de la búsqueda, por ejemplo, se pueden incorporar otros términos de búsqueda o realizar otras combinaciones de los términos usados.

Definición del protocolo de revisión

La definición de un protocolo de revisión implica especificar las normas de revisión, los criterios de exclusión e inclusión, la estrategia de extracción de datos y finalmente la estrategia de síntesis. Estos criterios forman parte del protocolo y deberán estar definidos antes de emprender la revisión sistemática. Contar con un protocolo predefinido contribuye a evitar los prejuicios el investigador. Con esto se desea evitar, en la medida de lo posible, que la selección de los estudios individuales pueda estar guiada por las expectativas del investigador.

Una parte importante del protocolo de revisión es contar con una definición de la forma en que se hará la revisión de manuscritos. Dado que se han revisado un alto número de artículos científicos hay que elaborar una técnica para mecanizar esta tarea. Se ha tomado como referencia para este protocolo de revisión la estructura de artículo científico propuesta por Srba en 2004 [14] (ver figura B.4). Aunque la revisión de manuscritos requiere de una cierta habilidad, que como cualquier otra habilidad mejora con la práctica [15], se ha considerado pertinente contar con una guía que permita abordar esta tarea.

Al igual que en el protocolo anterior, éste puede ser perfeccionado durante el desarrollo de la revisión.

Evaluación planificación

Esta etapa consiste en hacer una valoración objetiva de la planificación. Ya que esta propuesta se enmarca en el contexto del desarrollo de una tesis doctoral la evaluación de la planificación tendrá que realizarla el tutor de la tesis para comprobar que efectivamente se van cumpliendo los plazos previstos y tener la capacidad de reaccionar en caso contrario antes de continuar.

B.2.2. Etapa 2: Desarrollo de la revisión

En esta etapa se lleva a cabo la revisión propiamente dicha. Su desarrollo está guiado por la planificación de la revisión; no obstante, por ser éste un proceso flexible, es posible incluir cambios que mejoren su desempeño. A continuación se definen las sub-etapas que contempla el desarrollo de la revisión.

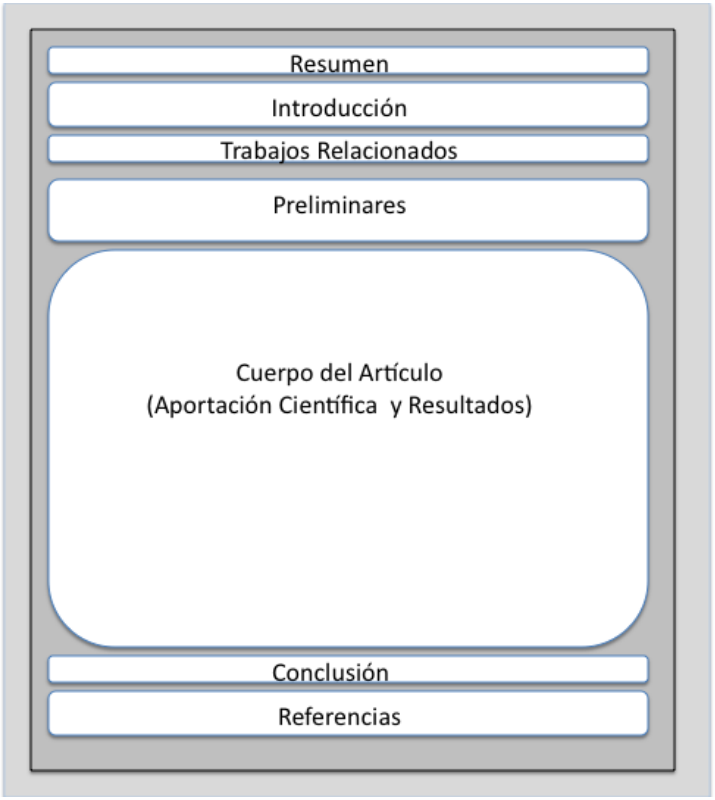

Figura B.4: Estructura artículo científico según Sbra

Búsqueda estudios primarios

La búsqueda de estudios primarios se debe realizar en base al protocolo de búsqueda que se definió para ello. Los estudios que se consideren potencialmente útiles quedarán accesibles para la siguiente etapa, ya sea en formato electrónico y/o impreso. También es posible dejar registrado el lugar en el que se pueden ubicar para que, cuando corresponda, se proceda a su selección. Para realizar esta búsqueda se ha invertido bastante tiempo, especialmente para el desarrollo de cierta destreza en la identificación de publicaciones de calidad frente a otras que no aportan un buen material.

Selección estudios primarios

La selección de los estudios se debe hacer en base al protocolo de revisión definido. Este proceso será guiado por los criterios de inclusión y exclusión definidos anteriormente. Aunque estos criterios dependen de los intereses del proyecto es recomendable dejar un registro de los motivos de exclusión. Esta selección está muy relacionada con la etapa anterior de búsqueda, en la que ya se realizaba un filtro de grano grueso. En esta etapa se realiza un filtro de grano fino, identificando los trabajos de calidad útiles relacionados con el problema planteado.

Extracción y gestión de datos

En esta sub-etapa se extrae la información de interés en los estudios, ya sea en forma de resúmenes, ideas o partes de los documentos. Es aconsejable registrar la información necesaria para la gestión, lo cual incluye el título del documento, autor o autores, fecha de publicación, ubicación física u otro tipo de información que los investigadores consideren pertinente. En la elaboración de esta tesis se ha utilizado la herramienta Mendeley Desktop para esta tarea, en especial para elaborar un registro con anotaciones y apuntes a cerca de cada uno de los trabajos revisados.

Síntesis de datos

En esta sub-etapa, al igual que en las anteriores, se debe aplicar el protocolo definido en la revisión. Consiste en registrar la información extraída de los estudios primarios siguiendo alguna estrategia definida. Los datos pueden ser sintetizados considerando, por ejemplo, el enfoque que se le desea dar a la presentación del estado del arte o la identificación del o de los fenómenos de interés. Una vez analizados los diferentes mecanismos existentes realizamos un estudio minucioso de cada uno de ellos, estableciendo una clasificación de los mismos en función del objetivo a proteger, ya sea el agente o la agencia. Asimismo realizamos un estudio de las debilidades y puntos fuertes de cada uno de estos mecanismos.

B.2.3. Etapa 3: Publicación de los datos

Esta etapa corresponde a la utilización de los resultados una vez que disponemos de ellos. Estos resultados pasan a formar parte, por ejemplo de una tesis doctoral, como ocurre en nuestro caso, que de forma adicional pueden ser comunicados, a través de la publicación de un artículo o un informe técnico. A lo largo de esta tesis se han publicado una gran cantidad de artículos con resultados relevantes, entre ellas destacamos [16], [17],[18] y [19].

B.3. Aplicación del método en este trabajo de tesis

Una vez descritos los métodos de Investigación-Acción vamos a describir la utilidad práctica de estos mediante el uso de estas técnicas en el desarrollo del presente trabajo de tesis. La características principales de estos métodos, orientación a la acción y al cambio, la focalización de un problema, el modelo de proceso orgánico que engloba etapas sistemáticas e iterativas y la colaboración entre las partes participantes sitúan a este método como una herramienta muy potente en el desarrollo de una investigación de alta calidad.

El primer punto en la aplicación del método consiste en la identificación de los diferentes roles participantes. El primero de ellos es el investigador, obviamente corresponde al autor de este trabajo. El segundo de los roles identificados es el objeto investigado, que en este caso es la problemática de la seguridad en los agentes móviles, que tienen un gran interés por las características específicas de este paradigma de computación como se

detalla en el capítulo 2. El tercer rol a identificar es que se relaciona con el grupo crítico, que es aquel para quien se investiga, en nuestro caso es el grupo GISUMSEC, subgrupo perteneciente al grupo de investigación GISUM dentro del departamento de lenguajes y ciencias de la computación de la Universidad de Málaga. Y por último no resta que identificar el rol del beneficiario, que en este caso es un grupo muy amplio de desarrolladores pertenecientes a la comunidad de los agentes móviles, puesto que todo el esfuerzo puesto en la mejora de esta tecnología va a repercutir en los beneficios obtenidos por la aplicación de los mismos.

Previamente describíamos los diferentes tipos de Investigación-Acción que nos podíamos encontrar, de diagnóstico, participativa, empírica y experimental. En principio este trabajo comenzó siendo un trabajo de diagnóstico que evolucionó a un trabajo con carácter participativo. Vamos a describir cada una de las etapas que han guiado este trabajo de investigación. Hemos de mencionar el hecho de que estas etapas forman uno de los ciclos del proceso que se ha iterado varias veces hasta obtener los resultados esperados.

La primera de las etapas es la correspondiente a la planificación en la que obtuvimos los puntos más importantes que guiaron nuestra investigación que, por su parte, están en íntima relación con el objetivo que se tomó inicialmente para la investigación. En esta etapa se pretendieron exponer las posibles alternativas para resolver el problema planteado. El objetivo planteado proporciona cierto mecanismo de seguridad robusto para los sistemas basados en agentes móviles, de manera que la utilización del mecanismo no implicase profundos conocimientos en el área de la seguridad ni complicados métodos de aplicación, que en definitiva no son viables en la práctica. Para ello se planteó la utilización de un dispositivo criptográfico para la protección tanto de los agentes como de las agencias, el siguiente paso era la elección del dispositivo. En gran medida debido a sus propiedades nos decantamos por el uso de TPM y de tarjetas inteligentes. Sin embargo, en el transcurso de esta disyuntiva y debido a los trabajos previos desarrollados con tarjetas inteligentes, pensamos en el uso del sistema smartprot para el mismo objetivo. Lo cual nos ofrecía otra vía alternativa de trabajo, que finalmente quedó abierta y se trabajó en paralelo en ambas líneas. Por un lado una solución basada en TPM y otro modelo que usa la "computación protegida".

La siguiente etapa, correspondiente a la acción como hemos mencionado se trata de una variación de la práctica cuidadosa, deliberada y controlada efectuando una simulación de la solución. En nuestro caso de investigación realizamos dos soluciones, en consonancia con la planificación obtenida. La primera de las soluciones utiliza como base un hardware criptográfico específico y está completamente elaborada usando el lenguaje de programación java, completamente integrado en la plataforma JADE. Y la segunda de las soluciones, siguiendo la misma filosofía también ha sido completamente desarrollada en java y sobre la plataforma JADE. Para esta segunda solución se han proporcionado una serie de herramientas para facilitar la aplicación de la metodología desarrollada.

La observación, recogida de información, toma de datos y documentación de lo sucedido junto con la reflexión y análisis de los resultados, al igual que las etapas anteriores se han ido realizando de forma iterativa y conjunta. Gracias a esta fase hemos podido apreciar la importancia de las herramientas de soporte para la aplicación de la metodología de "computación protegida" puesto que desde el punto de vista de un ingeniero de seguridad la solución es fácilmente entendible, sin embargo son necesarios suficientes conocimientos

criptográficos y relacionados con la seguridad como para utilizarse de forma aleatoria. Asimismo también nos ha ayudado en la comprensión de la importancia de desarrollar una solución de cara a ser usada por un gran número de usuarios sus repercusiones. Un diseño amigable, de fácil uso y por supuesto sin la menor pérdida en el nivel de seguridad proporcionado. Tras una gran labor y varias iteraciones en las fases de planificación, acción y observación podemos concluir que hemos obtenido dos soluciones suficientemente robustas para la resolución de un problema común desde dos perspectivas completamente distintas. No obstante, un aspecto revelador tras la reflexión de todo este trabajo radica en el hecho de que ambas soluciones pueden complementarse para la obtención de una solución más completa y robusta, especialmente para proporcionar seguridad en situaciones críticas.

Para concluir este capítulo vamos a describir los resultados obtenidos mediante la aplicación de este método en nuestro trabajo de investigación.

- Una extensión de JADE que utiliza algunas funcionalidad de seguridad proporcionada por el TPM (trusted platform module) y ofrece a los sistemas basados en agentes de un alto nivel de seguridad, mediante la protección del proceso de migración de los mismos agentes. Sin embargo, la utilización de esta extensión no implica grandes cambios en la metodología usada por los expertos en el ámbito de los agentes móviles.

- También hemos diseñado una serie de herramientas muy útiles para la obtención de un sistema de agentes seguro haciendo uso de la metodología de la computación protegida. La base de esta metodología consiste en establecer una serie de vínculos de dependencia entre todos los agentes del sistema, que son fruto de una partición y repartición del código de los agentes del sistema. Esta solución puede resultar más compleja que la anterior, no obstante también se ha facilitado la tarea mediante el desarrollo de herramientas automáticas de apoyo.

- Hemos aplicado ambas tecnologías a sistemas de agentes móviles reales, para los que hemos obtenido unas mejoras muy ventajosas en lo que a seguridad se refiere, especialmente en comparación con los sistemas previos, sin verse perjudicado el rendimiento del sistema.

Apéndice C

Diseño de la Solución basada en Hardware

C.1. Resultados obtenidos

Afortunadamente el resultado obtenido es el deseado. La librería de migración segura para agentes basada en el uso de la tecnología TPM está construida sobre la plataforma JADE. En la sección 3.3.1 llevamos a cabo una evaluación del producto obtenido mediante el uso de herramientas de model checking, el siguiente paso lógico es la aplicación en sistemas reales, que lo vemos descrito en la sección 3.4. En este apartado vamos a analizar el diseño de la librería. Estudiamos con detalle la arquitectura de la misma analizando los componentes que la forman y la función que realizan.

Como ya se ha mencionado a lo largo del capítulo 3, una de las ideas que hemos mantenido en mente durante toda la elaboración de este trabajo de investigación ha sido facilitar en la medida de lo posible la tarea del usuario final. A nadie se le escapa que la disciplina de la ingeniería de la seguridad es ardua compleja y generalmente de difícil aplicación práctica si no fuera por el uso de herramientas de apoyo. Por esta razón hemos realizado un diseño de la solución presentada en el capítulo 3 que tiene en consideración el perfil del usuario final de nuestra librería. Vamos a comenzar la descripción de nuestra librería, en primer lugar la vamos a mostrar desde la perspectiva del usuario, es decir el programador que tiene que desarrollar un software usando la tecnología de agentes. Este impone como requisito imprescindible que su sistema mantenga un nivel mínimo de seguridad, que como ya se ha demostrado en este trabajo de tesis JADE por sí misma lo no aporta y por tanto es insuficiente desde el punto de vista de la seguridad.

Siguiendo la pauta normal del desarrollo de este tipo de soluciones, el usuario deber definir los agentes que formarán su sistema. Para definir un agente en JADE es necesario crear una clase que herede de la clase Agent, de esta manera el agente creado dispondrá del comportamiento básico de un agente, siendo el usuario el que defina el comportamiento específico. Entre la funcionalidad básica de un agente JADE se encuentra la capacidad de migrar de un contenedor a otro. En la figura C.1 podemos ver la clase Agent de JADE. En aras de la simplificación, en la figura únicamente aparecen los métodos relacionados con la migración, que concretamente se corresponden con:

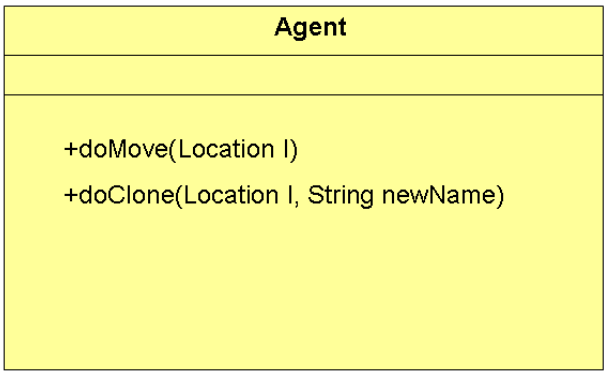

Figura C.1: Descripción de la clase Agent de JADE.

1. doMove(Location l) : Este método hace que el agente se desplace desde el contenedor en el que se encuentra hacia el contenedor destino 1 "Location1".

2. doClone(Location l, String newName): Este método permite la clonación del agente en el contenedor "Location1", tomado como nombre "newName".

Estos métodos son los mismos tanto para la migración interplataforma como para la migración intraplataforma. La diferencia entre ambos tipos de migración la establece el tipo dinámico del parámetro l. El parámetro "Location1" indica el destino de la migración que puede ser un ContainerID, si el destino es un contenedor de la misma plataforma en la que se encuentra el agente, o un PlatformID, si el destino es otra plataforma.

En el apartado de especificación mencionamos que la agencia debe proporcionar al agente los servicios necesarios para poder realizar la migración de manera segura. La plataforma JADE ofrece, a los agentes que contiene, una serie de funcionalidades a través de servicios del contenedor. Un ejemplo de estos servicios es el de movilidad que permite a los agentes desplazarse de un contenedor a otro, así como el servicio de mensajes, que se encarga de enviar mensajes de un agente a otro.

Para el desarrollo de nuestra librería se han implementando dos nuevos servicios para JADE que permiten a los contenedores ofrecer la migración tanto en modo intraplataforma como en modo interplataforma. Esto se ha logrado gracias a que se ha seguido el mismo esquema de servicios proporcionado por JADE. Nuestra librería, a la que hemos llamado SecMiLiA proporciona estos servicios superponiendo una capa, que los dota de seguridad y que permite realizar la migración de manera segura.

SecMiLiA proporciona dos servicios básicos. El primero de ellos se llama *SecureAgentMobility* y se basa en el servicio estándar de JADE AgentMobility. Se usa para migrar de forma segura entre contenedores de la misma plataforma. En segundo lugar el servicio *SecureInterPlatformMobility* usa el servicio InterPlatformMobility para migrar de manera segura entre plataformas diferentes.

Como vimos anteriormente, la clase Agent de JADE proporciona dos métodos para migrar de manera no segura, por lo que nuestra librería añade una clase que hereda de la clase Agent y que redefine los métodos de migración para que se realicen de manera segura. La clase que vamos a definir se puede observar en la figura C.2.

Entre los métodos más destacados de esta clase vamos a comentar los que siguen:

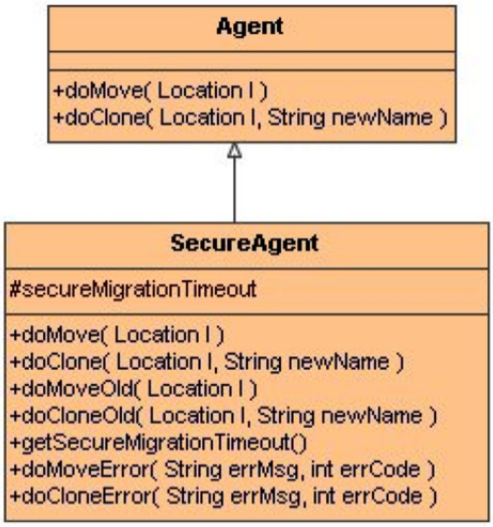

Figura C.2: Redefinición clase Agent por *SecureAgent*.

- doMove(Location l): Este método permite mover al agente al destino indicado en el parámetro l de manera segura.
- doClone(Location l, String newName): Para clonar al agente en el destino indicado por el parámetro l de manera segura. El nombre del agente clonado será el indicado en el parámetro "newName".
- *doMoveOriginal* (Location l): Permite mover al agente como lo hará el método doMove de la clase Agent.
- doCloneOriginal(Location l, String newName): Clona al agente como lo hará el método doClone de la clase Agent.
- getSecureMigrationTimeout(): devuelve el tiempo máximo que espera el Agente seguro antes de migrar. Una vez cumplido el tiempo se aborta el proceso de migración. El tiempo máximo de espera se aplica sólo para el servicio *SecureInterPlatformMobility* ya que usa mensajes ACL, mientras que el servicio *SecureAgentMobility* usa comandos de servicio. La definición semántica formal de los mensajes ACL ha sido uno de los esfuerzos mayores dentro del estándar FIPA, y otorga a este lenguaje de una gran aceptación como estándar dentro del mundo de los agentes. Desde este punto de vista, hay que entender a un agente como una entidad capaz de interpretar la semántica del lenguaje ACL y que es capaz de intercambiar conocimiento con otros agentes mediante el uso del mismo. Ya no se habla de que la interfaz de un agente (objeto) ofrezca una serie de servicios que se puedan invocar o instanciar, sino que un agente va a proporcionar ciertos servicios que se le pueden solicitar mediante un mensaje ACL que incluya dicha solicitud. Se pasa de un modelo de objetos a un modelo de agentes inteligentes basados en actos de comunicación. En cambio, al usarse comandos de servicio se detecta inmediatamente si el destino existe o no, sin

embargo al usar mensajes ACL sólo se detecta si existe el destino cuando transcurre un tiempo y no se recibe respuesta.

- *doMoveError* (String errMsg, int errCode): Este método se ejecuta cuando se ha producido un error en el proceso de que impida que el agente se mueva hacia el destino. El parámetro errMsg contiene el mensaje de error, mientras que el parámetro errCode contiene el código de error.

- doCloneError(String errMsg, int errCode): Este método se ejecuta cuando se haya producido un error en el proceso que impide que el agente se clone en el destino. El parámetro errMsg contiene el mensaje de error, mientras que el parámetro errCode contiene el código de error.

- secureMigrationTimeout: este atributo permite al usuario que crea el agente seguro definir el tiempo máximo de espera para la migración.

Tanto el método doMove como el doClone pueden ser ejecutados por cualquiera que tenga una instancia de la clase Agent, por lo que pueden ser llamados tanto por el propio agente como por la plataforma que lo contiene.

Para crear un agente seguro, el programador únicamente deberá crear una clase que herede de *SecureAgent* y redefinir los métodos secureMoveError y secureCloneError definiendo el comportamiento esperado, ya que ambos son métodos abstractos. Con esto tendremos definido el interfaz de uso de la librería, que como podemos ver es bastante sencillo. Sin embargo, lo visto hasta el momento no ofrece directamente ninguna funcionalidad, por lo que la librería incluye más elementos importantes que realizan las acciones. Entre estos elementos se encuentran los servicios que permiten llevar a cabo la migración en sí de manera segura. Puesto que estamos trabajando sobre JADE, hemos de mencionar que cualquier servicio JADE debe implementar:

- Los interfaces Helper y Slice. En JADE, los agentes pueden usar los servicios proporcionados por el contenedor. Para ello deben primero obtener un interfaz que les permita acceder al servicio. Como vimos en el capítulo 3, este es el interfaz Helper, al cual puede acceder el agente simplemente a través del nombre del servicio. Sin embargo, no todos los servicios proporcionan un Helper. Además de este interfaz de acceso, el servicio debe proporcionar el interfaz Slice, que sirve para que el servicio pueda comunicarse con otros servicios pertenecientes a contenedores diferentes.

- Las clases CommandIncomingFilter y CommandOutgoing filter, para gestionar comandos entrantes. Hemos de mencionar que entendemos por comandos de salida si provienen del contenedor del servicio y de entrada si han llegado desde otros contenedores.

- Las clases CommandTargetSink y CommandSourceSink, para consumir los comandos que le lleguen. Un servicio puede consumir un comando propio una vez haya atravesado los filtros de todos los demás servicios. El filtro origen tiene comandos de entrada propios, provenientes del contenedor del servicio mientras que el filtro destino consume comandos provenientes de otros contenedores.

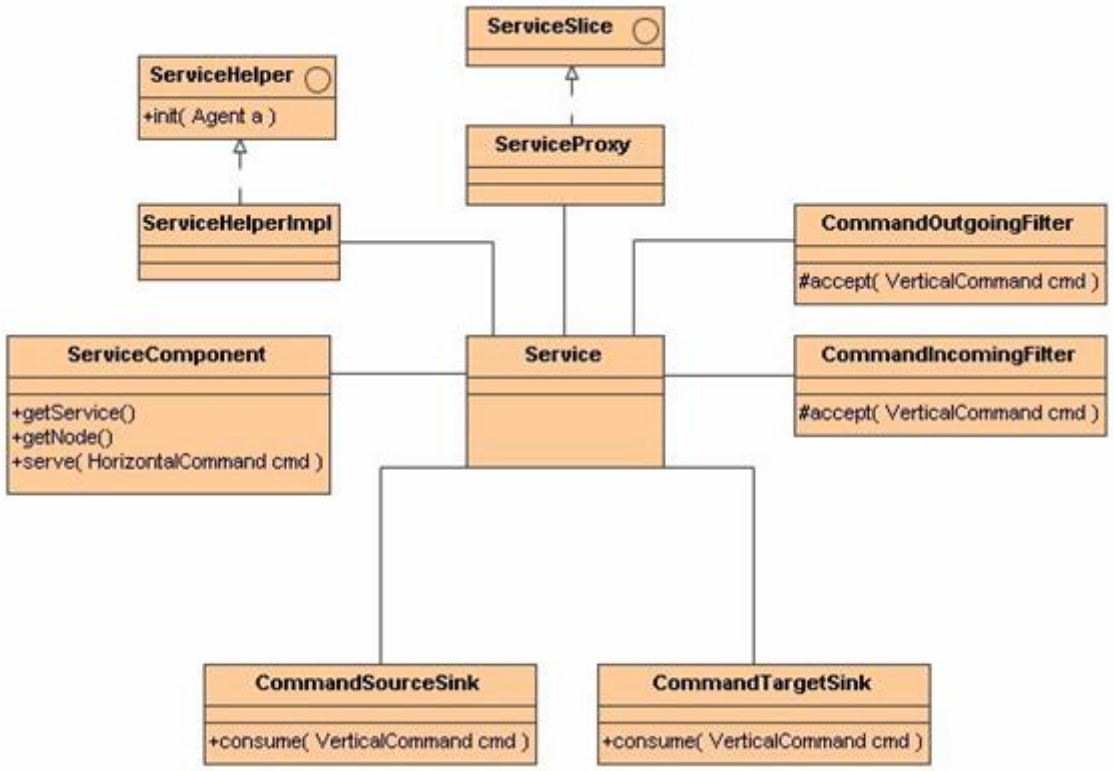

Figura C.3: Diagrama de clases de un servicio completo.

- La clase ServiceComponent, que sirve para tratar los comandos de otros servicios recibidos a través del proxy.

En la figura C.3 podemos ver el diagrama de clases de un servicio completo. Una vez vista la estructura de las clases que componen un servicio vamos a pasar a realizar una descripción de los servicios específicos de la librería de migración segura.

El servicio *SecureAgentMobility* proporciona la funcionalidad de migración de manera segura entre contenedores de la misma plataforma. En la figura C.4 podemos ver el diagrama de clases del servicio.

En este diagrama podemos ver cómo el Helper proporciona dos métodos:

- *secureMove(SecureAgent a,Location destination)*: Permite a los agentes seguros moverse al contenedor destino de manera segura.

- *secureClone(SecureAgent a,Location destination, String newName)*: Permite a los agentes seguros clonarse en el contendor destino de manera segura.

El interfaz *SecureAgentMobility* Slice define una serie de métodos que servirán para que el servicio pueda interactuar con servicios pertenecientes a otros contenedores, ya que el servicio, para poder realizar su funcionalidad, debe contactar con servicios de otros

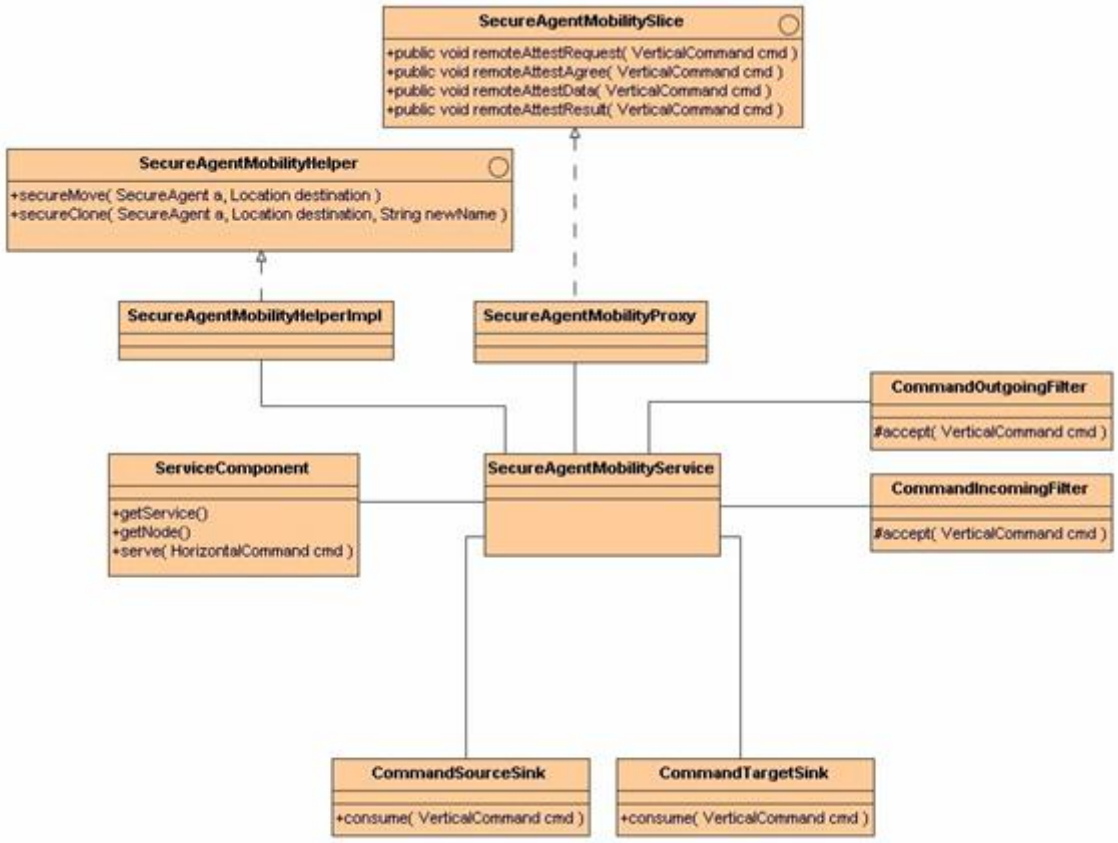

Figura C.4: Diagrama de clases del servicio.

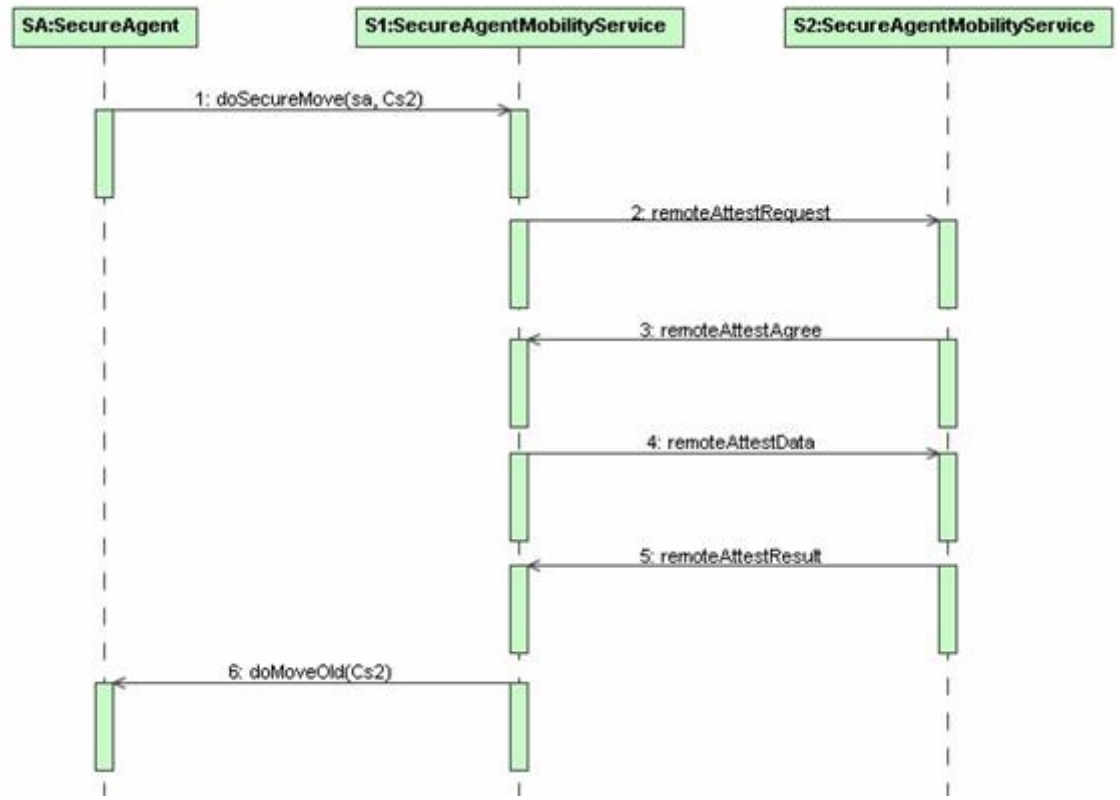

Figura C.5: Diagrama de secuencia del servicio.

contenedores. En la figura C.5 podemos ver el protocolo que sigue el servicio *SecureAgentMobility* para comunicarse con la instancia del mismo servicio perteneciente a otro contenedor.

Este diagrama muestra un caso en el que el agente solicita al servicio que lo mueve a un contenedor (C2). Los pasos son los siguientes:

1. El agente SA solicita al servicio S1 moverse al contenedor C2.
2. El servicio envía una solicitud de atestación remota al servicio S2.
3. El servicio S2 acepta la solicitud.
4. El servicio S1 envía los datos de la solicitud al servicio S2.
5. El servicio S2 responde al servicio S1 enviando el resultado de la atestación.
6. El servicio S1 inicia la migración del agente al contenedor C2.

En el diagrama de la figura C.5 se muestra al servicio como una entidad única. Sin embargo, para realizar la acción mostrada sería necesaria la participación de varios componentes pertenecientes al servicio que se han omitido en aras de la claridad del protocolo. Otro aspecto importante es que es el mismo servicio el que inicia la migración del agente invocando al método *doMoveOriginal*, cuyo funcionamiento es exactamente el mismo al de

doMove de la clase Agent, es decir, migra al agente hacia el destino. Este método se invoca sólo después de que se haya comprobado que la agencia destino efectivamente es segura. De esta manera se consigue dotar al método doMove de la clase *SecureAgent* de un comportamiento similar al doMove de la clase Agent.

El protocolo consta de cuatro mensajes básicos que describimos a continuación:

1. remoteAttestRequest: Es el de petición de atestación y contiene por un lado el nombre del agente que realizó la petición para poder identificarlo y en segundo lugar la ubicación del contenedor en el que se encuentra el agente, o lo que es igual la agencia origen. Y por último es necesaria la ubicación del contenedor destino.

2. remoteAttestAgree: Es el mensaje de respuesta de a la solicitud. Este contiene el nombre del agente que realizó la petición, la ubicación del contenedor en el que se encuentra el agente y la del contenedor destino. Además es necesario un valor nonce de 160 bits aleatorios para evitar ataques de repetición, así como los índices de los PCRs que el contenedor destino necesita saber para conocer si el contenedor origen es seguro.

3. remoteAttestData: Es para el envío de la información necesaria pasa hacer la atestación y contiene, de forma similar a los dos anteriores, el nombre del agente que realiza la petición, la ubicación del contenedor, en el que se encuentra el agente, del contenedor destino y un valor "nonce" de 160 bits aleatorios para evitar ataques de repetición. También será necesario conocer los índices de los PCR que el contenedor origen necesita saber para conocer si el contenedor destino es seguro, los valores de los PCR solicitados por el contenedor destino firmados con una AIK, y las credenciales de la AIK usada para firmar los datos.

4. remoteAttestResult: Es el de envío de los resultados de la atestación que consta del nombre del agente que realizó la petición, la ubicación del contenedor en el que se encuentra el agente y la del contenedor destino, los valores de los PCR solicitados por el contenedor origen firmados con una AIK, junto con las credenciales de la AIK usada para firmar los datos.

El contenido de estos mensajes se encapsula usando las clases AttestRequest_Interface y AttestData_Interface.

La clase AttestRequest_Interface proporciona acceso a los datos de un mensaje de solicitud de atestación:

- getAMS(): nos devuelve el AID del agente AMS de la plataforma en la que se encuentra el agente que pretende iniciar la migración.
- getRequester(): nos devuelve el AID del agente.
- getDestination(): nos devuelve la ubicación a la que quiere migrar el agente.
- getRequesterLocation(): nos devuelve la ubicación en la que se encuentra el agente.
- getReqAttestData(): nos devuelve los datos de atestación de la ubicación origen.

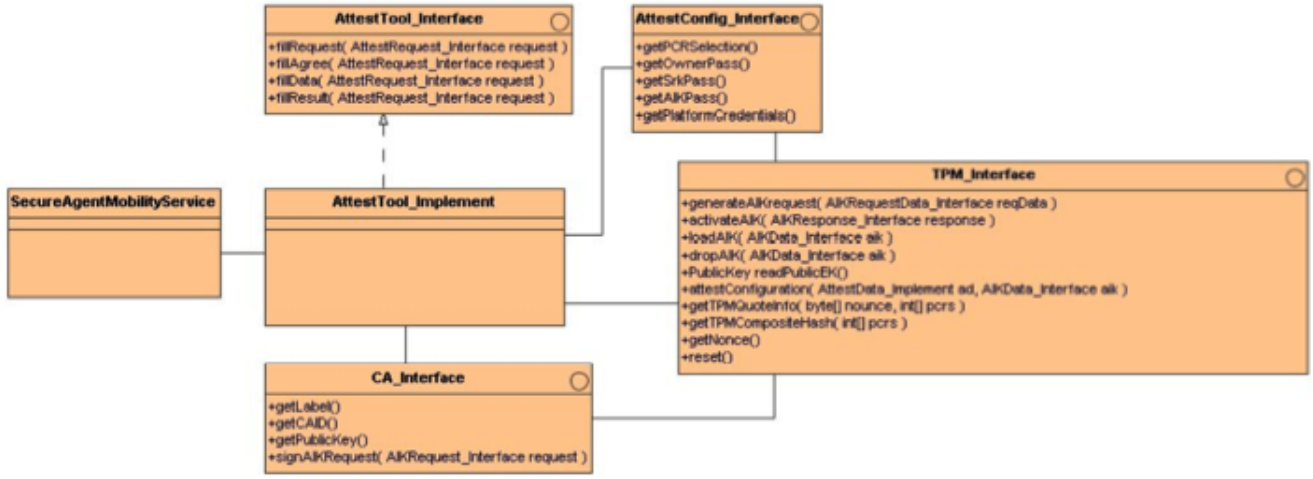

Figura C.6: Descripción del servicio Secure Agent Mobility.

- getRemAttestData(): nos devuelve los datos de atestación de la ubicación desino.
- getStatusMessage(): nos devuelve un mensaje que contiene el estado en el que se encuentra la migración.
- getStatus(): nos devuelve un valor que indica el estado en el que se encuentra la migración.
- getTimeout(): nos devuelve el tiempo máximo de validez del mensaje de petición de atestación.

La clase AttestData_Interface nos proporciona acceso a los datos de atestación de un contenedor determinado:

- getAIKCredentials(): nos devuelve las credenciales de la AIK usadas para firmar los PCR.
- getPCRSelection(): nos devuelve los índices de los PCR cuyos valores se han firmado.
- getPCRData(): nos devuelve el valor de los PCR seleccionados.
- getNonce(): nos devuelve el valor nonce usado para evitar ataques de repetición.
- getSignedPCR(): nos devuelve los datos de los PCR firmados con la clave privada de la AIK.

Ambas clases encapsulan los datos de los mensajes que componen el protocolo para realizar la atestación. En cada mensaje el contenedor correspondiente va completando los datos necesarios usando los métodos "set", para que el otro contenedor pueda seguir con el proceso de atestación. Para completar los datos de los mensajes el servicio *SecureAgentMobility* hace uso de la clase *AttestTool_Implement*, como ilustra la figura C.6

La clase *AttestTool_ Implement* se encarga de gestionar los diferentes mensajes del protocolo de atestación, esto es, generarlos en la ubicación origen del mensaje y verificarlos en la ubicación destino. La clase *AttestTool_ Implement* tiene acceso al TPM del sistema a través del interfaz TPM_Interface y a una CA a través del interfaz CA_Interface, esto le permite usar las funcionalidades de estas dos entidades para poder completar los datos de los mensajes. Además, la clase *AttestTool_ Implement* tiene acceso a la configuración del sistema a través del interfaz *AttestConfig_ Interface*.

Vamos a analizar la función de cada uno de los métodos de *AttestTool_ Interface* :

- fillRequest(AttestRequest_interface request): se encarga de completar la información necesaria para realizar una solicitud de atestación. Este método es llamado por el servicio en el contenedor origen.
- filAgree(AttestRequest_interface request): genera la información necesaria para que el contenedor origen pueda enviar la información de atestación, es decir, un valor nonce y los índices de los PCR cuyos calores se desea conocer. Este método es invocado por el servicio en el contenedor destino.
- fillData(AttestRequest_interface request): completa la información de atestación del contenedor origen, es decir, los valores PCR firmados y las credenciales de la AIK. Además completa la información necesaria para que el contenedor destino pueda enviar la información de atestación, valor nonce e índices de los PCR.
- fillResult(AttestRequest_interface request): Comprueba si el contenedor origen es seguro y en ese caso completa la información de atestación del contenedor destino, es decir, los valores PCR firmados y las credenciales de la AIK.
- fill*INFORM* (AttestRequest_interface request): Comprueba si el contenedor destino es seguro y completa la información necesaria para informar al agente solicitante.

La clase *AttestTool_ Implement* se encarga de gestionar el acceso a la funcionalidad del TPM necesaria para que el servicio pueda realizar el protocolo de atestación. Por otro lado, la clase *AttestTool_ Implement* se comporta como el *Key Cache Manager (KCM) del TPM*. El KCM es el encargado de gestionar el movimiento de clases entre el slot de las claves volátiles del TPM y el almacenamiento en dispositivos externos que son "no-volátiles". El KCM determina el momento justo apropiado para cambiar la clave y para ser reemplazada por otra.

Esta clase implementa el interfaz *AttestTool_ Interface* en el cual se definen los códigos que identifican el estado del proceso de migración segura, estos códigos son:

- ATT_OK: El estado de la migración segura es correcto hasta el momento.
- ATT_ERROR: Se ha producido un error durante el proceso de migración (por ejemplo, se ha detectado que la configuración del contenedor destino no es segura).
- ATT_REFUSE: El contenedor destino ha rechazado la migración por algún motivo (por ejemplo, el contenedor destino considera que la configuración del contenedor origen no es segura).

- ATT_FAILURE: Se ha producido un fallo en el proceso de migración (por ejemplo, no es posible comunicarse con el contenedor destino).
- ATT_TIMEOUT: Se ha superado el tiempo máximo de espera para recibir la respuesta del contenedor destino.
- ATT_BUSY: El servicio del contenedor origen o del contenedor destino está ocupado y no puede atender la petición.

Este código de errores es el que permite al agente determinar lo sucedido cuando se recibe una llamada a los métodos *doMoveError* y doCloneError.

Para poder generar y verificar los datos contenidos en los mensajes de atestación es necesario el uso del TPM y de una CA. El TPM ofrece la funcionalidad necesaria para poder generar los datos de atestación, la generación de la AIK, la producción de la firma y la de valores nonce. Por su parte, la CA permite generar credenciales que certifiquen las AIK para que los demás contenedores puedan confiar en los datos firmados por esas AIK.

El interfaz TPM_Interface proporciona acceso a las diferentes funcionalidades del TPM:

- init(*AttestConfig_Interface* config, CA_Interface ca): inicializa el interfaz con el módulo TPM de manera que este tenga acceso a la configuración de la plataforma y a la CA que certifica sus claves.
- generateAIKrequest(AIKRequestData_Interface reqData): genera una petición de clave AIK a partir de los datos contenidos en el parámetro.
- reqData: esta llamada que será enviado a una CA para poder certificarla.
- activateAIK(AIKResponse_Interface aikres): activa la clave AIK cuyos datos están contenidos en el parámetro aikres, obteniendo a su vez las credenciales de la CA para dicha clave.
- attestConfiguration(AttestData_Interface ad, AIKData_Interface aik): firma valores de PCR junto al nonce especificados en el parámetro ad, usando la AIK cuyos datos contiene el parámetro aik.
- dropAIK(AIKData_Interface aik): elimina del TPM la AIK cuyos datos contiene el parámetro aik.
- getNonce(): devuelve un valor nonce aleatorio, para evitar ataques por repetición.
- getTPMQuoteInfo(byte[] nonce, int[] pcrs): devuelve los valores de los PCR cuyos índices se indican en el parámetro pcrs junto con el nonce especificado en el parámetro.
- getTPMCompositeHash(int[] pcrs): devuelve los valores de los PCR cuyos índices se indican en el parámetro pcrs.
- reset(): resetea el módulo TPM.

- loadAIK(AIKData_Interface aik): Carga en el TPM la AIK cuyos datos contiene el parámetro aik.
- readPublicEK(): devuelve el valor de la clave pública de la EK del TPM.

El interfaz CA_Interface proporciona acceso a las diferentes funcionalidades de la CA, que son:

- getLabel(): que devuelve el valor de la etiqueta de la CA.
- getCAId(): que ofrece como salida el identificador da la CA.
- getPublicKey(): devuelve la clave pública de la CA.
- signAikRequest(AIKRequest_Interface request): genera las credenciales para una AIK, comprobando previamente las credenciales de la entidad solicitante.

El interfaz *AttestConfig_Interface* proporciona acceso a los valores de configuración de la plataforma así como a otros valores propios de la plataforma. Los métodos que proporciona son:

- getPCRSelection(): devuelve los índices de los PCR que se solicitarán a los contenedores remotos para conocer su configuración.
- getOwnerPass(): para obtener el password del propietario del TPM.
- getSrkPass(): nos ofrece el password de la clave de almacenamiento del TPM.
- getAikPass(): para conseguir el password de las AIK.
- getPlatformCredentials(): devuelve las credenciales de la plataforma.
- getGoodConfigValues(): nos permite obtener los valores de configuración válidos para compararlos con los valores recibidos de otros contenedores. Cada valor es del tipo TPM_COMPOSITE_HASH visto en el apartado dedicado a la librería tpm4java.

Un aspecto importante del protocolo de la figura C.1 es el uso de una AIK para firmar los datos de atestación. La AIK la genera el TPM y debe ser certificada por una CA. En la figura C.7 vemos el protocolo que muestra la forma en la que se genera y certifica esta clave.

Como se aprecia en el diagrama, los pasos son:

1. *AttestTool_Implement* solicita la generación de una clave AIK y de una solicitud de credenciales de AIK al TPM.
2. El TPM genera la clave AIK y la solicitud y se las entrega al AttestTool. La solicitud está cifrada de manera que sólo la CA tendría la capacidad de poder leerla.
3. *AttestTool_Implement* envía la solicitud a la CA.

Figura C.7: Certificación clave con AIK.

4. La CA descifra la solicitud y genera las credenciales correspondientes que entrega al *AttestTool_ Implement* en una respuesta a la petición cifrada para que sólo el TPM pueda leerla.

5. El *AttestTool_ Implement* envía la respuesta de la CA al TPM.

6. El TPM descifra la petición y devuelve los datos de la clave al *AttestTool_ Implement*.

En este proceso intervienen una serie interfaces que se corresponden a los diferentes valores que se envían y reciben en el protocolo. Estos interfaces son:

- AIKRequestData_Interface: contiene los datos que permiten al TPM generar la AIK y crear la petición de generación de credenciales.

- AIKRequest_Interface: contiene los datos que permiten a la CA generar unas credenciales para la AIK.

- AIKResponse_Interface: contiene los datos necesarios para que el TPM pueda obtener las credenciales generadas por la CA.

C.1.1. El servicio *SecureInterPlatformMobilityService*

Este servicio hace uso de muchos de los elementos usados en el servicio *SecureAgentMobility*. La diferencia fundamental de este servicio radica en que, al tratarse de migración entre plataformas diferentes, no se pueden enviar mensajes de servicio entre el contenedor origen y destino, ya que para ello ambos contenedores deben pertenecer a la misma plataforma. Esto obliga a comunicarse mediante mensajes ACL.

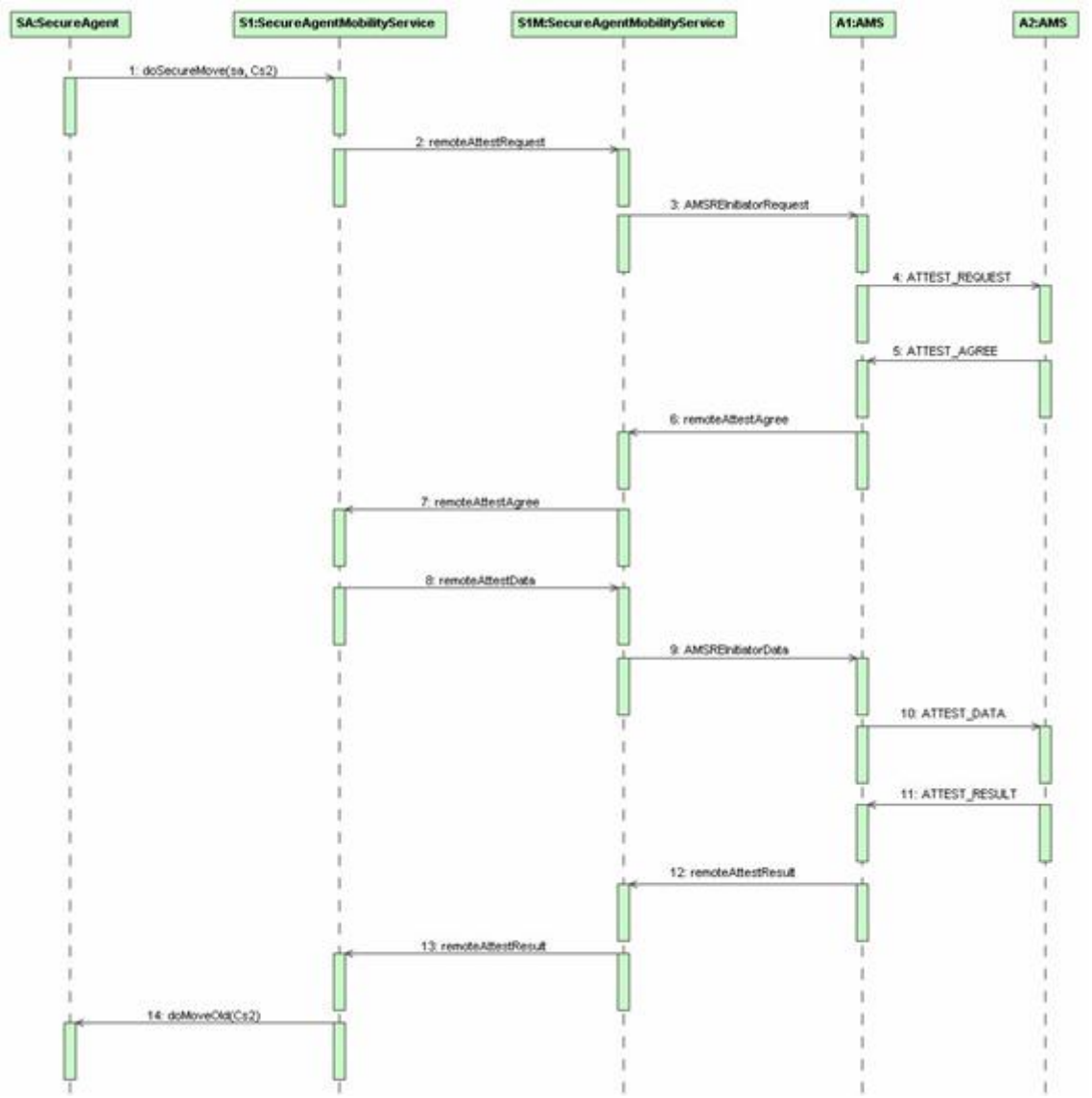

Figura C.8: Interfaz de credenciales.

En la figura C.8 podemos ver cuál es el protocolo que sigue el servicio para realizar la migración segura.

Los pasos que realiza el servicio para realizar la migración segura son:

1. El agente SA solicita al servicio S1 moverse al contenedor C2.

2. El servicio S1 envía una solicitud de atestación remota al servicio S1M del contenedor principal de su plataforma.

3. El servicio S1M envía la solicitud de atestación remota al AMS de la plataforma origen, A1.

4. A1 envía una solicitud de atestación al AMS de la plataforma destino, A2.

5. A2 acepta la solicitud y se lo notifica a A1.

6. A1 notifica la aceptación al servicio S1M.

7. El servicio S1M notifica la aceptación al servicio S1.

8. El servicio S1 envía los datos de la solicitud al servicio S1M.

9. El servicio S1M envía los datos de la solicitud a A1.

10. A1 envía los datos de la solicitud a A2.

11. A2 responde a A1 enviando el resultado de la atestación.

12. A1 envía los resultados al servicio S1M.

13. El servicio S1M envía los resultados recibidos al servicio S1.

14. El servicio S1 inicia la migración del agente al contenedor principal de la plataforma destino.

Como vemos, el servicio del contenedor origen necesita la participación del servicio del contenedor principal para que contacte con el servicio de manejo de agentes (AMS) de la plataforma destino. Esto es necesario ya que sólo es posible acceder a la clase que implementa al AMS desde el contenedor principal. La comunicación entre los AMS de la plataforma origen y destino se realiza mediante mensajes ACL. El AMS destino usa la clase *AttestTool_Implement* para tratar los mensajes del servicio, mientras que en el origen los mensajes son tratados por *AttestTool_Implement* en el servicio del contenedor origen. El funcionamiento del servicio en lo que respecta a todos los demás componentes es igual al del servicio *SecureAgentMobility*, por lo que los daremos por analizados.

C.1.2. Implementación de los servicios de la librería de migración segura

Una vez vista la arquitectura de ambas librerías y su comportamiento vamos a pasar a analizar la implementación de las clases que describen dicha arquitectura y comportamiento. Al igual que en el apartado anterior comenzaremos con el servicio *SecureAgentMobility* y luego veremos el servicio *SecureInterPlatformMobility*.

Para analizar la implementación del servicio vamos a ver lo que sucede durante todo el proceso de migración. Para ello repasamos todas las clases y sucesos que intervienen en el mismo:

1. Se realiza una llamada al método doMove de la clase *SecureAgent*, lo que provoca que se llame al método secureMove de la clase *SecureAgentMobility* HelperImpl perteneciente al servicio *SecureAgentMobility* del contenedor origen.

2. *SecureAgentMobility* HelperImpl genera la solicitud de atestación remota usando el método fillRequest de la clase *AttestTool_Implement*, añade la solicitud al comando vertical *ATTEST_REQUEST* y lo lanza.

3. La clase CommandSourceSink consume el comando y llama al método remoteAttestRequest de la clase *SecureAgentMobility* Proxy que implementa el interfaz *SecureAgentMobility* Slice.

4. El método remoteAttestRequest envía el comando horizontal *H_REMOTE_ATTEST_REQUEST* al servicio *SecureAgentMobility* del contenedor destino.

5. El comando horizontal *H_REMOTE_ATTEST_REQUEST* es capturado por la clase ServiceComponent del servicio *SecureAgentMobility* del contenedor destino y se lanza el comando vertical *REMOTE_ATTEST_REQUEST*.

6. El comando *REMOTE_ATTEST_REQUEST* es consumido por la clase CommandTargetSink, la cual obtiene la solicitud y la procesa usando el método fillAgree de la clase *AttestTool_Implement*. Posteriormente llama al método remoteAttestAgree de la clase *SecureAgentMobility* Proxy.

7. El método remoteAttestAgree envía el comando horizontal *H_REMOTE_ATTEST_AGREE* al servicio *SecureAgentMobility* del contenedor origen.

8. El comando es capturado por la clase *ServiceComponent*, que lanza un comando *REMOTE_ATTEST_AGREE*.

9. La clase *CommandTargetSink* consume el comando y procesa la petición con ayuda del método *fillData()* de la clase *AttestTool_Implement*. Posteriormente llama al método *remoteAttestData()* de la clase *SecureAgentMobilityProxy*.

10. El método *remoteAttestData* envía el comando horizontal *H_REMOTE_ATTEST_DATA* al servicio *SecureAgentMobility* del contenedor destino.

11. El comando es capturado por la clase *ServiceComponent*, que lanza un comando *REMOTE_ATTEST_DATA*.

12. La clase *CommandTargetSink* consume el comando y procesa la solicitud con ayuda del método fillResult de la clase *AttestTool*. Posteriormente llama al método remoteAttestResult de la clase *SecureAgentMobility* Proxy.

13. El método *remoteAttestResult* envía el comando horizontal *H_REMOTE_ATTEST_RESULT* al servicio *SecureAgentMobility* del contenedor origen.

14. El comando es capturado por la clase *ServiceComponent*, que lanza un comando *REMOTE_ATTEST_RESULT*.

15. La clase CommandTargetSink consume el comando y procesa la solicitud con ayuda del método fill*INFORM* de la clase *AttestTool_Implement*.

El siguiente paso consiste en llamar al método *informAgent* que ejecutará *doMoveOriginal* de la clase *SecureAgent* en el caso de que el contenedor destino sea seguro. O bien, el método *doMoveError*, en el caso de que el contenedor destino no sea seguro o haya ocurrido alguna eventualidad que impida migrar al agente.

Los pasos descritos se refieren a una llamada al método doMove de la clase *SecureAgent*, para el método doClone el proceso sería el mismo, sólo que al final el servicio invocara el método doCloneOld en lugar del método *doMoveOriginal*.

Hasta este punto hemos visto las ideas más relevantes del servicio *SecureAgentMobility*, tanto de las clases que forman el servicio en sí como de las clases más importantes necesarias para que todo funcione. A continuación vamos a analizar el servicio *SecureInterPlatformMobility*.

C.1.3. El servicio *SecureInterPlatformMobility*

La implementación de este servicio es muy similar a la del anterior ya que utiliza prácticamente las mismas clases, por lo que sólo comentaremos los aspectos que los diferencian. La gran diferencia entre ambos servicios radica en que al tratarse de migración entre plataformas diferentes los contenedores origen y destino no pueden comunicarse mediante comandos horizontales, por lo que deben hacerlo mediante mensajes ACL enviados por el AMS de cada plataforma. Otra peculiaridad de este servicio es que, dado que no es posible para un servicio acceder a la instancia del AMS a menos que sea el servicio del contenedor principal, los servicios de otros contenedores deben interactuar con el contenedor principal para que este acceda al AMS e inicie los comportamientos que gestionarán los mensajes del protocolo de atestación remota.

Para analizar la implementación del servicio vamos a ver, como hicimos con el servicio *SecureAgentMobility*, lo que sucede durante todo el proceso de migración segura, viendo todas las clases que intervienen en el proceso. Los sucesos que tienen lugar en el proceso de migración segura son, de manera detallada, los siguientes (para el caso en el que el contenedor origen no sea el contenedor principal):

1. Se realiza una llamada al método doMove de la clase *SecureAgent*, lo que provoca que se llame al método secureMove de la clase *SecureInterPlatformMobility* HelperImpl perteneciente al servicio *SecureInterPlatformMobility* del contenedor origen.

2. *SecureInterPlatformMobility* HelperImpl genera la solicitud de atestación remota usando el método fillRequest del la clase *AttestTool_Implement*, añade la solicitud al comando vertical *ATTEST_REQUEST* y lo lanza.

3. La clase CommandSourceSink consume el comando a través de su método consume y llama al método attestRequest de la clase SecureInterPlatformtMobilityProxy, que implementa el interfaz *SecureInterPlatformMobility* Slice.

4. El método attestRequest envía el comando horizontal *H_ATTEST_REQUEST* al servicio *SecureInterPlatformMobility* del contenedor principal.

5. El comando horizontal *H_ATTEST_REQUEST* es capturado por la clase ServiceComponent del servicio *SecureInterPlatformMobility* del contenedor principal y se lanza el comando vertical *ATTEST_REQUEST*.

6. El comando *ATTEST_REQUEST* es consumido por la clase CommandTargetSink, la cual crea una instancia del comportamiento AMSREInitiatorRequest y lo añade al AMS.

7. *AMSREInitiatorRequest* envía un mensaje *ATTEST_REQUEST* al AMS de la plataforma destino.

8. El comportamiento *AMSREResponder* captura el mensaje y procesa la solicitud con el método fillAgree de la clase *AttestTool_Implement*. Posteriormente envía, primero un mensaje AGREE y luego un mensaje *INFORM* al *AMSREInitiatorRequest*.

9. *AMSREInitiatorRequest* captura ambos mensajes y lanza un comando *ATTEST_DATA*.

10. La clase CommandSourceSink del contenedor principal consume el comando y llama al método attestData de la clase *SecureInterPlatformMobility* Proxy.

11. El método attestData envía el comando horizontal *H_ATTEST_DATA* al servicio *SecureInterPlatformMobility* del contenedor origen.

12. El comando es capturado por la clase ServiceComponent, que lanza un comando *ATTEST_DATA*.

13. La clase CommandTargetSink consume el comando y procesa la solicitud con ayuda del método *fillData()* de la clase *AttestTool_Implement*. Posteriormente llama al método attestData de la clase *SecureInterPlatformMobility* Proxy.

14. El método attestData envía el comando horizontal *H_ATTEST_DATA* al servicio *SecureInterPlatformMobility* del contenedor principal.

15. El comando es capturado por la clase ServiceComponent, que lanza un comando *ATTEST_DATA*.

16. El comando *ATTEST_DATA* es consumido por la clase CommandTargetSink, la cual crea una instancia del comportamiento *AMSREInitiatorData* y lo añade al AMS.

17. *AMSREInitiatorData* envía un mensaje *ATTEST_DATA* al AMS de la plataforma destino.

18. El comportamiento *AMSREResponder* captura el mensaje y procesa la solicitud con el método fillResult de la clase *AttestTool_Implement*. Posteriormente, envía primero un mensaje AGREE y luego un mensaje *INFORM* al *AMSREInitiatorData*.

19. *AMSREInitiatorData* captura ambos mensajes y lanza un comando *ATTEST_RESULT*.

20. La clase CommandSourceSink consume el comando y llama al método attestResult de la clase *SecureAgentMobility* Proxy.

21. El método attestResult envía el comando horizontal *H_ATTEST_RESULT* al servicio *SecureInterPlatformMobility* del contenedor origen.

22. El comando es capturado por la clase ServiceComponent, que lanza un comando *ATTEST_RESULT*.

23. La clase CommandTargetSink consume el comando y procesa la solicitud con ayuda del método *fillINFORM* de la clase *AttestTool_Implement*. Posteriormente llama al método *INFORM* Agent que ejecuta el método *doMoveOriginal* de la clase *SecureAgent* en el caso de que la plataforma destino sea segura o el método *doMoveError*, también de la clase *SecureAgent*, en el caso de que la plataforma destino no sea segura o haya ocurrido alguna eventualidad que impida migrar al agente.

Bibliografía

[1] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. Efficient software-based fault isolation. *In Proceedings of the 14th ACM Symposium on Operating Systems Principles.*, pages 203–216, 1993.

[2] R. Glass, I. Vessey, and V. Ramesh. Research in software engineering: an analysis of the literature. *Information and Software Technology.*, 44(8):491–506, 2002.

[3] F. Ruiz, M. Polo, and M. Piattini. Utilización de la investigación-acción en la definición de un entorno para la gestión del proceso de mantenimiento de software. *En 1er Workshop en Métodos de Investigación y Fundamentos Filosóficos en la Ingeniería del Software y Sistemas de Información (MIFISIS). El Escorial, España.*, 2002.

[4] B. Kitchenham. Procedures for performing systematic reviews. *Technical Report TR/SE-0401, Keele University.*, 2004.

[5] R. Obrien. *An Overview of the Methodological Approach of Action Research.* University of Toronto., 1998.

[6] F. Kock, N. Lau. Information systems action research: Serving demanding masters. *Information Technology and People (special Action Research in Information Systems).*, 14(1):6–11, 2001.

[7] G. Padak, N. Padak. Guidelines for planing action research projects. *Ohio Literacy Resource Center.*, 1994.

[8] J. Estay, C. Pastor. Improving action research in information systems with project management. *Proceedings of the 2000 Americas Conference on Information Systems, Long Beach, California (USA).*, pages 1558–1561, 2000.

[9] J. Estay, C. Pastor. Towards a project structure for action-research in information systems. *In 10th Annual Business and Information Technology Conference (BIT), Manchester (UK).*, 2000.

[10] F. Lau. A review on the use of action research in information systems studies. *In Information Systems and Qualitative Research. Chapman and Hill. London.*, pages 31–68, 1997.

[11] R. Baskerville. Investigating information systems with action research. *Communications of the Association for Information Systems.*, 2(19), 1999.

[12] M. Caro, A. Rodriguez, C. Calero, E. Fernandez-Medina, and M. Piattini. Análisis y revisión de la literatura en el contexto de proyectos de fin de carrera: Una propuesta. In *En VIII Congreso Chileno de Educación Superior en Computación.*, 2005.

[13] S. Lawrence. Online or invisible? *Nature.*, 411(6837):521–523, 2001.

[14] J. Srba. How to read and present a scientific paper. *Disponible en http://www.cs.auc.dk/hans/Dat5/slides.pdf.*, 2004.

[15] D. Benos, K. Kirk, and J. Hall. How to review a paper. *Advances in Physiology Education.*, 27(2):47–52, 2003.

[16] Antonio Muñoz and Antonio Maña. Tpm-based protection for mobile agents. *International Journal of Security and Communication Networks and Information Security*, 1, November 2009.

[17] A. Muñoz, A. Maña, and D. Serrano. The role of trusted computing in the secure agent migration. *ACM Computing Surveys*, 6(5):30–58, November 2009.

[18] Antonio Maña, Antonio Muñoz, and Daniel Serrano. Towards secure agent computing for ubiquitous computing and ambient intelligence. In Jadwiga Indulska, Jianhua Ma, Laurence Tianruo Yang, Theo Ungerer, and Jiannong Cao, editors, *UIC*, volume 4611 of *Lecture Notes in Computer Science*, pages 1201–1212. Springer, 2007.

[19] Javier Lopez, Antonio Maña, and Antonio Muñoz. A secure and auto-configurable environment for mobile agents in ubiquitous computing scenarios. In Jianhua Ma, Hai Jin, Laurence Tianruo Yang, and Jeffrey J. P. Tsai, editors, *UIC*, volume 4159 of *Lecture Notes in Computer Science*, pages 977–987. Springer, 2006.

[20] N. Karnik. *Security in Mobile Agents systems.* PhD thesis, Department of Computer Science, University of Minnesota., 1998.

[21] S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(3):1–40, 1996.

[22] *JADE, documentation and resources. available at: http://jade.tilab.com/.*

[23] William M. Farmer, Joshua D. Guttman, and Vipin Swarup. Security for mobile agents: Issues and requirements, 1996.

[24] W. Jansen and T. Karygiannis. Mobile agent security. *Nist special publication 800-19, National Institute of Standards Technology.*, 2000.

[25] Volker Roth. Mutual protection of co-operating agents. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, pages 275–285. Springer-Verlag Inc, 1999.

[26] D. Chess. Security considerations in agent-based systems. In *First Conference on Emerging Technologies and Applications in Communications (etaCOM)*, May 1996.

[27] J. Mir and J. Borrell. Protecting mobile agent itineraries. In *Mobile Agents for Telecommunication Applications (MATA)*, volume 2881 of *LNCS*, pages 275–285. Springer-Verlag, 2003.

[28] E.Bierman and E. Cloete. Classification of malicious host threats in mobile agent computing. *In Proceedings of the anual research conference of the South African institute of computer scientists and information technologist on Enablement through technology. South Africa Institure for Computer Scientists and Information Technologists.*, pages pages 141–148., 2002.

[29] Y. Minsky, R. van Renesse, F. Schneider, and S.D. Stoller. Cryptographic support for fault-tolerant distributed computing. *In Seventh ACM SIGOPS European Workshop.*, 1996.

[30] F. Hohl. A framework to protect malicious hosts attacks by using reference states. In *International conference on distributed computing systems (ICDCS)*, 2000.

[31] G. Vigna. Cryptographic traces for mobile agents. *In Mobile Agents and Security.*, volume 1419 of LNCS. Springer-Verlag., 1998.

[32] F. Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. *In Mobile Agents and Security.*, volume 1419 of LNCS. Springer-Verlag., 1998.

[33] Sander and C.F. Tschudin. Protecting mobile agents against malicious hosts. *Mobile Agents and Security*, 1419 of LNCS. Springer-Verlag., 1998.

[34] J. Domingo-Ferrer. A provably secure additive and multiplicative privacy homomorphism. In *5th Information Security Conference (ISC)*, 2002.

[35] J. Ordille. When agents roam, who can you trust? In *Technical report, Computing Science Research Center, Bells Lab*, 1996.

[36] H. Kim and L. Moreau. Trust relationships in a mobile agent system. In Springer-Verlag, editor, *5th International Conference on Mobile Agents (MA)*, volume 2240 of *LNCS*, 2001.

[37] S.Robles. *Mobile Agent Systems and Trust, a Combined View Toward Secure Sea-of-Data Applications.* PhD thesis, Universitat Autonoma de Barcelona, 2002.

[38] Sau-Koon Ng. *Protecting Mobile Agents Against Malicious Hosts.* PhD thesis, The Chinese University of Hong Kong, 2002.

[39] J. Riordan and B. Schneier. Environmental kay generation towards clueless agents. In Springer-Verlag, editor, *Mobile Agents and Security*, volume 1419 of *LNCS*, 1998.

[40] Bennet S. Yee. A sanctuary for mobile agents. In *Secure Internet Programming*, pages 261–273, 1997.

[41] U. G. Wilhelm, S. Staamann, and L. Buttyan. Introducing trusted third parties to the mobile agent paradigm. In Springer-Verlag, editor, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *LNCS*, 1999.

[42] S. Funfrocken. Protecting mobile web-commerce agents with smartcards. In *First International Symposium on Agent Systems and Applications (ASA) / Third International Symposium on Mobile Agents (MA)*, 1999.

[43] Zijiang Yang, Shiyong Lu, and Ping Yang. Runtime security verification for itinerary-driven mobile agents. In *IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC)*, pages 177–186, 2006.

[44] G. Necula and P. Lee. Proof-carrying code. *Technical Report CMU-CS-96-165, Carnegie Mellon University.*, 1996.

[45] O. Goldreich. Towards a theory of software protection. In ACM, editor, *19th Ann. ACM Symposium on Theory of Computing.*, pages 182–194, 1987.

[46] Amir Herzberg and Shlomit S. Pinter. Public protection of software. *ACM Trans. Comput. Syst.*, 5(4):371–393, 1987.

[47] Derrick Grover, editor. *The protection of computer software—its technology and applications.* Cambridge University Press, New York, NY, USA, 1989.

[48] Antonio Maña and Ernesto Pimentel. An efficient software protection scheme. In *In Proceedings of the 16th International Conference on Information Security: Trusted Information*, pages 200–1. Kluwer Academic Publishers, 2001.

[49] A. Maña and A. Muñoz. Mutual protection for multiagent systems. *Proceedings of the Third International Workshop on Safety and Security in Multiagent Systems (SASEMAS'06)*, 2006.

[50] Antonio Muñoz, Antonio Maña, and Daniel Serrano. Secmilia: An approach in the agent protection. *Availability, Reliability and Security, International Conference on*, 0:341–348, 2009.

[51] Antonio Munoz, Antonio Mana, Rajesh Harjani, and Marioli Montenegro. Agent protection based on the use of cryptographic hardware. In *COMPSAC '09: Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference*, pages 312–317, Washington, DC, USA, 2009. IEEE Computer Society.

[52] *Trusted Computing Group:TCG Specifications. Available online at: https://www.trustedcomputinggroup.org/specs/*, 2005.

[53] Luca Vigano. Automated security protocol analysis with the avispa tool. In *Proceedings of MFPS*, 2006.

[54] Liana Bozga, Yassine Lakhnech, and Michaël Périn. Hermes: An automatic tool for verification of secrecy in security protocols. In *15th International Conference on Computer Aided Verification (CAV),*, volume 2725 of *lncs*, 2003.

[55] David Basin, Sebastian Modersheim, and Luca Vigano. An on-the-fly model-checker for security protocol analysis. In *In Proceedings of Esorics, LNCS 2808*, pages 253–270. Springer-Verlag, 2003.

[56] Per Bjesse and Koen Claessen. Sat-based verification without state space traversal. In *In Formal Methods in Computer-Aided Design*, pages 372–389. Springer, 2000.

[57] Y. Boichut, P-C. O Kouchnarenko, and F. Oehl. Improvements ont the genet and klay technic to automatically verify security protocols. In *AVIS, ENTCS*, 2004.

[58] Antonio Muñoz, Antonio Maña, and Daniel Serrano. Avispa in the validation of ambient intelligence scenarios. *Availability, Reliability and Security, International Conference on*, 0:420–426, 2009.

[59] Iliano Cervesato. The dolev-yao intruder is the most powerful attacker. In *Proceedings of the Sixteenth Annual Symposium on Logic in Computer Science | LICS'01*, pages 16–19. IEEE Computer Society Press. Short, 2001.

[60] Angelo Troina, Ro Aldini, and Roberto Gorrieri. The java virtual machine specification, http://java.sun.com/docs/books/vmspec. In *In Proc. of GC'04, volume 3267 of Springer LNCS*, pages 77–92, 2005.

[61] Eric Bruneton, Romain Lenglet, and Thierry Coupaye. Asm: A code manipulation tool to implement adaptable systems. In *In Adaptable and extensible component systems*, 2002.

[62] Markus Dahm. Byte code engineering with the bcel api. 2001.

[63] Norbert A. Streitz, Achilles Kameas, and Irene Mavrommati, editors. *The Disappearing Computer, Interaction Design, System Infrastructures and Applications for Smart Environments*, volume 4500 of *Lecture Notes in Computer Science*. Springer, 2007.

[64] J. Krogstie. Requirements engineering for mobile information systems. In *Proceedings of the Seventh International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ)*, 2001.

[65] Julio Abascal. Ambient intelligence for people with disabilities and elderly people. acm's. In *Special Interest Group on Computer-Human Interaction (SIGCHI), Ambient Intelligence for Scientific Discovery (AISD) Workshop*, 2004.

[66] Anand S. Rao and Michael P. Georgeff. Bdi agents: From theory to practice. In *In proceedings of the first International Conference on Multi-Agent Systems (ICMAS-95*, pages 312–319, 1995.

[67] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. A methodology for agent-oriented analysis and design, 1999.

[68] Brian Henderson-Sellers and Paolo Giorgini. *Agent-oriented methodologies / Brian Henderson-Sellers, Paolo Giorgini.* Hershey, Pa. : Idea ; London : Eurospan [distributor], 2005. Formerly CIP.

[69] G. Necula. Proof-carrying code. *In Proceedings of 24th Annual Symposium on Principles of Programming Languages.*, 1997.

[70] J. Stern, G. Hachez, F. Koeune, and J.J Quisquater. Robust object watermarking: Application to code. In *Info Hiding, Springer-Verlag.*, volume LNCS 1768., pages pp. 368–378, 1999.

[71] O. Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 182–194, New York, NY, USA, 1987. ACM.

[72] Siani Pearson. Trusted computing: Strengths, weaknesses and further opportunities for enhancing privacy. In Peter Herrmann, Valérie Issarny, and Simon Shiu, editors, *iTrust*, volume 3477 of *Lecture Notes in Computer Science*, pages 305–320. Springer, 2005.

[73] Nicholas Carr. *The Big Switch: Rewiring the World, from Edison to Google.* W.W. Norton & Company, 2009.

[74] John Krogstie, Kalle Lyytinen, Andreas Lothe Opdahl, Barbara Pernici, Keng Siau, and Kari Smolander. Research areas and challenges for mobile information systems. *Int. J. Mob. Commun.*, 2(3):220–234, 2004.

[75] *Times Online. available at: http://technology.timesonline.co.uk/.*

[76] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2001.

[77] A. Maña. *Protección de Software Basada en Tarjetas Inteligentes.* PhD thesis, University of Málaga., 2003.

[78] A. Maña, J. López, J. Ortega, E. Pimentel, and J. Troya. A framework for secure execution of software. *International Journal of Information Security*, 3(2)(2004), Novembre 2004. ISSN 1615-5262 (Print) 1615-5270 (Online).