View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by The Oberta in open access

Metodología para el Desarrollo Automatizado de Aplicaciones Seguras basadas en Agentes Móviles

C. Garrigues, S. Robles, A. Moratalla

Departament d'Enginyeria de la Informació i de les Comunicacions,
Universitat Autònoma de Barcelona
08193 Bellaterra - Espanya
carles@deic.uab.es

Resumen En este trabajo presentamos una metodología para el diseño de aplicaciones basadas en agentes móviles seguros, y unas herramientas para aplicar esta metodología. Los recientes avances en el campo de la seguridad de los agentes móviles pueden desbloquear el uso de esta tecnología, pero falta aún superar la complejidad que suponen al programador de las aplicaciones. La metodología propuesta, y el uso de nuestras herramientas, permiten agilizar el proceso de diseño de arquitecturas criptográficas, y de los agentes que las utilizan. La implementación presentada es muy flexible, y permite utilizarse con cualquier plataforma de agentes que siga los estándares de IEEE-FIPA y que permita los esquemas de protección conducidos por agente.

1. Introducció

El interés por los agentes móviles vuelve a incrementarse tras algún tiempo de aparente bloqueo [1]. Esta situación parece responder a los recientes avances en algunos aspectos críticos sobre seguridad, como aquellos descritos en [2]. La falta de mecanismos de seguridad adaptados a las particularidades de esta tecnología, junto a la carencia de aplicaciones decisivas mostrando su utilidad, y a la falta de consciencia sobre sus posibilidades por parte de la industria, han sido los principales factores que han frenado el impulso de los agentes móviles [3]. Todo ello, pese a sus numerosos beneficios tecnológicos.

Sin embargo, para que los recientes avances en seguridad puedan alentar al desarrollo de nuevas aplicaciones seguras debe franquearse un impedimento añadido, que es la dificultad para implementar y utilizar estos esquemas. No es suficiente la validez teórica y técnica de los sistemas de seguridad que protegen a los agentes móviles. Es además necesario que puedan desarrollarse de una manera simple, sin que sea preciso un conocimiento profundo de criptografía, y que puedan también utilizarse de forma sencilla. Es preciso, pues, crear una nueva metodología de diseño y desarrollo que facilite sus labores tanto a los diseñadores de nuevos esquemas criptográficos de protección de agentes como a los desarrolladores de las aplicaciones que los usaran. Será también necesario dotar a esta metodología de las herramientas básicas necesarias que permitan aplicarla.

No existe mucha bibliografía sobre estos aspectos concretos, ya que normalmente el trabajo en este área se centra en los mecanismos de protección de los agentes y se

dejan aparte las cuestiones más básicas de usabilidad que afectan al desarrollador. Los autores ya han trabajado previamente en las ideas básicas tras esta nueva concepción en la asistencia al desarrollo de aplicaciones seguras, desarrollando una prueba de concepto de un compilador de arquitecturas criptográficas [4]. Existen otros trabajos en esta área sobre la generación de agentes móviles que permiten crear código seguro [5]. Sin embargo, la seguridad que se consigue en ellos protege a la plataforma frente a agentes maliciosos, pero no a los propios agentes. En cambio, nuestra propuesta contempla también la protección contra ataques sobre el código y los datos del agente, un aspecto imprescindible en aplicaciones seguras basadas en agentes móviles.

En el presente artículo proponemos una metodología completa para el diseño de aplicaciones basadas en agentes móviles protegidos con arquitecturas criptográficas. Este tipo de protección ha marcado un punto de inflexión en los mecanismos de seguridad de agentes móviles, ya que es el primero que hace una aproximación conducida por agente en vez de por plataforma [6]. Se presentan también las herramientas básicas implementadas y utilizadas en la metodología, describiendo con mayor detalle el constructor de agentes, pieza clave en la generación de los agentes seguros. Finalmente, se ofrecen las perspectivas que abre esta nueva familia de herramientas tanto al desarrollador de aplicaciones seguras como al diseñador de esquemas de seguridad.

2. Características de nuestros agentes móviles

Para entender el proceso de generación de agentes con nuestro entorno de desarrollo, debemos primero entender que tipo de agentes móviles estamos creando.

En primer lugar, nuestros agentes móviles utilizan itinerarios fijados, lo que significa que se deben especificar de antemano, en el momento de crear el agente. Desde nuestro punto de vista, un itinerario incluye el código a ejecutar en cada plataforma y el conjunto de saltos entre las diferentes plataformas que definen su comportamiento global. Aunque muchos autores consideran siempre los itinerarios como dinámicos, veremos más adelante como podemos proporcionar la misma flexibilidad que un itinerario dinámico con un itinerario fijado. Además, como veremos, la utilización de itinerarios fijados es más aconsejable, debido a que aporta ventajas como que permite la protección del itinerario, se facilita en gran medida su reutilización, y otras.

En segundo lugar, los itinerarios de nuestros agentes móviles se especifican de forma explícita. Esto significa que el código relativo al salto entre las diferentes plataformas se separa del código específico de cada una de ellas. En el caso contrario, cuando el itinerario está implícito en el código, las tareas a ejecutar y los saltos a las plataformas sucesoras se implementan en un mismo código. Cuando el itinerario es explícito, éste se proporciona como una estructura independiente que contiene el código de cada plataforma y el orden de ejecución de cada uno de ellos.

Cuando el itinerario se especifica explícitamente, el agente debe ser provisto de un código que sepa manejar la estructura que contiene el itinerario. Este código, denominado *código de control*, es el responsable de poner en ejecución las tareas locales en cada plataforma y de mover al agente a la siguiente plataforma cuando su ejecución a terminado.

La protección de la estructura que guarda el itinerario (cuando éste es explícito) se hace utilizando mecanismos criptográficos, los cuales evitan que el itinerario sea accedido o manipulado por terceras partes no autorizadas. Hasta este momento, diferentes protocolos se han propuesto para la protección de itinerarios [7,8,9,2]. Debido a que la elección entre uno u otro protocolo depende de los requerimientos de la aplicación concreta, no tiene ningún sentido que todos los agentes lleven el itinerario protegido mediante el mismo algoritmo. Por esta razón, nuestro esquema de desarrollo proporciona un mecanismo para implementar estos protocolos de una manera flexible y reutilizable.

Para poder entender como los itinerarios de los agentes pueden ser protegidos, en el siguiente apartado veremos como son los itinerarios que nosotros consideramos.

2.1. Itinerarios

Nuestros itinerarios están compuestos de elementos básicos denominados nodos. Los nodos pueden ser de diferentes tipos en función de la relación que haya entre ellos y sus nodos adyacentes. Por ejemplo, el tipo de nodo más simple es la secuencia, en el cual el agente ejecuta el código local correspondiente a la plataforma y únicamente tiene una plataforma sucesora a la cual saltar. Por lo tanto, una vez terminada la ejecución del código local, el agente simplemente salta a la plataforma siguiente.

En cambio, otros tipos de nodos más complejos tienen un conjunto de subitinerarios sucesores que requieren alguna acción especial antes de que el agente pueda hacer el salto a la siguiente plataforma. Por ejemplo, en el caso del nodo clonación, el agente debe clonarse y enviar un clon a cada uno de los subitinerarios que sucedan al nodo actual. En el caso de los nodos alternativa, el agente debe ejecutar un método de condición que servir para decidir cual de los siguientes subitinerarios debe escoger el agente para proseguir su ejecución.

La razón por la cual nuestros itinerarios pueden estar compuestos de esta variedad de nodos es permitir al programador que defina sus itinerarios con la misma flexibilidad que define el flujo de control de sus programas. De esta manera, los diferentes tipos de nodos pretenden ser el equivalente a las estructuras de control de flujo de cualquier lenguaje de programación: instrucciones condicionales, bucles, etc.

3. Protección de itinerarios y generación de agentes automatizada

La protección del itinerario del agente tiene como objetivo conseguir que cada una de las plataformas pueda acceder la código y los datos que les corresponden. De esta manera, la plataforma no puede saber ni modificar el comportamiento del agente en el resto del itinerario. Esta protección del itinerario permite tratar el problema más delicado en lo referente a la seguridad de agentes móviles: los ataques de la plataforma contra el agente. Hay muchas maneras de proteger a la plataforma de posibles ataques del agente (con mecanismos de Sandboxing, por ejemplo), pero conseguir proteger completamente al agente de cualquier ataque de la plataforma es imposible. Sin embargo, en ciertos entornos como los propuestos en [10], la protección del itinerario aporta una solución muy adecuada a los ataques de las plataformas. Estos entornos están caracterizados por el hecho de que hay una fuerte competencia entre las plataformas pero lealtad al usuario.

A pesar del hecho de que proteger el itinerario de los agentes incrementa la seguridad de los agentes considerablemente, los protocolos de protección de itinerarios son raramente utilizados. La razón de esta falta de uso es que los protocolos son bastante difíciles de implementar, incrementando considerablemente el coste de la implementación final. Pueden requerir, por ejemplo, obtener certificados de plataformas o realizar operaciones criptográficas para obtener una parte del itinerario del agente [6]. En consecuencia, la programación puede implicar el uso de una infraestructura de clave pública, un conocimiento profundo del entorno de ejecución de los agentes y, en general, un amplio conocimiento sobre criptografía.

Otro problema es la falta de reutilización de estas implementaciones. Puesto que los mecanismos de seguridad están normalmente centrados en la protección de un itinerario concreto, la implementación no puede reutilizarse para proteger un itinerario distinto. Si, tal y como acabamos de mencionar, la implementación de estos mecanismos requiere los mayores esfuerzos, la falta de reutilización se convierte en un problema importante del proceso de desarrollo actual de agentes móviles.

Para solucionar estas limitaciones, hemos creado un esquema de desarrollo compuesto por tres componentes principales: la herramienta de diseño de itinerarios (IDT), el constructor de agentes (AB), el lanzador de agentes (AL) y el recuperador de resultados (DR). En la figura 1 podemos ver un esquema con los componentes principales que constituyen nuestra propuesta.

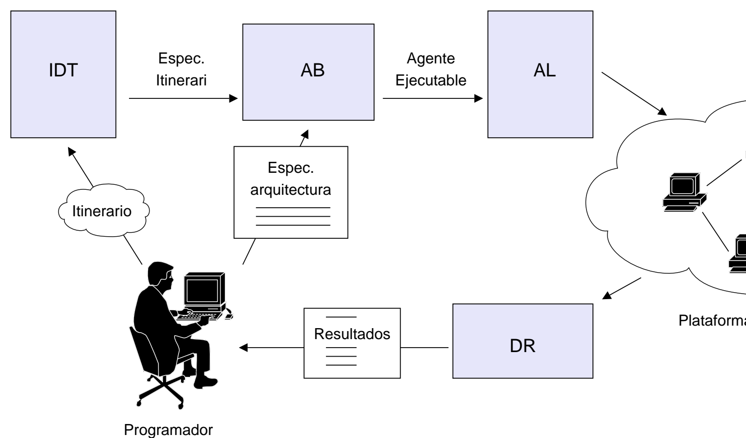

Figura 1. Diagrama del esquema de desarrollo propuesto

3.1. El constructor de agentes

El elemento clave de la simplificación del desarrollo es el constructor de agentes. Las entradas que recibirá esta herramienta serán, por un lado, el itinerario que el agente debe seguir y, por otro, el protocolo de protección de este itinerario. Utilizando estas

dos entradas, el constructor de agentes generará el agente final que será introducido en la primera plataforma del itinerario con el lanzador de agentes. Como veremos más adelante, tanto el itinerario como el protocolo de protección se proporcionan mediante especificaciones.

De hecho, nuestro principal objetivo de simplificar el desarrollo de agentes móviles seguros, se consigue gracias al hecho que los mecanismos de protección del itinerario se implementan utilizando el nuevo lenguaje que hemos creado. Este lenguaje permite implementar operaciones criptográficas de manera fácil. Por ejemplo, para cifrar unos datos con la clave pública de una determinada plataforma se necesita solamente una instrucción.

Otro elemento importante de este trabajo es que los desarrollos se dividen en dos partes completamente independientes: la definición del itinerario y la implementación del protocolo de protección asociado. Esta división aporta mucha flexibilidad y incrementa la capacidad de reutilización de código, debido a que una especificación del itinerario concreta puede ser utilizada con cualquier especificación de protocolo y viceversa.

3.2. Herramienta de diseño de itinerarios

Para facilitar el trabajo del programador aún más, nuestro esquema incluye una herramienta de diseño de itinerarios. Utilizando esta aplicación, el programador define gráficamente el conjunto de plataformas que componen su itinerario y asigna a cada plataforma su código local correspondiente. Una vez hecho esto, esta herramienta genera una especificación del itinerario estándar que puede ser protegida con cualquier protocolo de protección para generar el itinerario final.

Esto significa que la especificación del itinerario no determina el protocolo que se utilizará para proteger el itinerario. Sin embargo, un cierto protocolo puede imponer restricciones sobre el tipo de itinerarios que puede proteger. Tal y como hemos visto en la sección anterior, los itinerarios están compuestos de diferentes tipos de nodos. Algunos protocolos pueden no soportar la protección de un itinerario por el hecho de que incluya un tipo de nodo no soportado. Por este motivo, la herramienta de diseño de itinerarios permite al programador especificar qué arquitectura utilizará para proteger el itinerario que está definiendo. De esta forma, la herramienta puede asegurarse de que todos los nodos que se incluyan están soportados por la arquitectura especificada.

3.3. El lanzador de agentes

El lanzador de agentes es la herramienta responsable de enviar los agentes generados por el constructor de agentes a la primera plataforma del itinerario. La implementación que se ha hecho consiste sencillamente en una aplicación cliente que contacta con el servicio de migración de las plataformas. Concretamente, contacta con la parte de inmigración de este servicio para enviarle el agente como si éste estuviera migrando desde otra plataforma. De esta manera, la plataforma de ejecución de agentes no ha tenido que ser modificada en absoluto.

Una de las ventajas de esta herramienta es que soluciona una limitación común presente en los sistemas de agentes móviles actuales: una vez se ha iniciado la ejecución

de un sistema de agentes o plataforma, nuevos agentes sólo pueden ser arrancados de manera local, teniendo acceso directo a la máquina en cuestión. Con nuestro lanzador, en cambio, nuevos agentes pueden ser enviados remotamente a cualquier plataforma en ejecución.

3.4. El recuperador de resultados

La herramienta de recuperación de resultados es una aplicación que sigue el paradigma cliente-servidor. Como su nombre indica, permitirá al propietario del agente recuperar los resultados de la ejecución una vez este haya terminado. La parte cliente es bien simple: se encargará de conectar con la parte servidora, que ha sido instalada en la plataforma remota, y descargará los resultados del agente en el caso que estos estén disponibles.

En lo referente a la parte servidora, ésta será la que permitirá que los agentes depositen sus resultados antes de finalizar la ejecución. Esto hará que estos resultados sean almacenados en disco, de tal manera que, en caso de que la plataforma se reinicie, los resultados podrán ser recuperados igualmente. Esta parte servidora se ha implementado como un agente móvil que se ejecuta permanentemente en la plataforma, lo cual ha permitido no tener que añadir un nuevo servicio a la plataforma de ejecución de agentes y, por tanto, no ha sido necesario modificarla.

3.5. Metodología de desarrollo de agentes seguros

Finalmente, vamos a ver la secuencia de pasos que nuestra metodología propone para diseñar, implementar y poner en ejecución a nuestros agentes móviles. En primer lugar, es necesario que el programador especifique el plan de viaje del agente en la herramienta de especificación de itinerarios. En este plan de viaje constarán sólo las plataformas que conforman la ruta que el agente debe seguir. A continuación, en esta misma herramienta, el programador introducirá las tareas que el agente debe ejecutar en cada plataforma. Según la metodología de nuestra propuesta, mediante esta herramienta de diseño, el programador también podrá validar el itinerario, por ejemplo mediante una simulación, comprobando que no existe ningún camino que pueda generar conflictos entre varios agentes. La parte de simulación de esta herramienta, sin embargo, está todavía en una fase de diseño y no se dispone aún de una implementación.

Una vez generada la especificación del itinerario, el siguiente paso es la generación del agente que transporte este itinerario protegido. Para ello, el programador podrá utilizar la especificación de cualquier arquitectura ya existente o creará una nueva especificación en caso de ser necesario. Recordemos que, gracias a su nivel de abstracción y facilidad de implementación, las especificaciones de las arquitecturas podrán ser generadas por cualquier experto en seguridad sin conocimientos de programación alguno. Luego, estas mismas especificaciones, cualquier programador podrá reutilizarlas para generar su agente seguro.

A partir de aquí, con el itinerario y la arquitectura de protección del mismo, el constructor de agentes se encargará de generar un agente ejecutable preparado para ser lanzado a la primera plataforma. La puesta en ejecución del agente se hará desde nuestro lanzador de agentes, y podrá ser realizada por el mismo programador o cualquier

usuario que necesite los servicios proporcionados por el agente. En este momento, el agente se ejecutará de forma autónoma por el conjunto de plataformas previsto. Al finalizar la ejecución, el agente habrá generado probablemente un conjunto de resultados que podrán ser recogidos por su propietario mediante la herramienta de recuperación de resultados.

En este punto, podemos ver como nuestra metodología de desarrollo y ejecución de agentes permite distinguir tres roles completamente independientes: En primer lugar, el programador del agente, que genera el itinerario, lo protege y genera el agente ejecutable. En segundo lugar, el experto en seguridad, que diseña el algoritmo de protección utilizado por el constructor de agentes. Por último, el usuario final del agente que, como en cualquier otra aplicación, puede ejecutar el agente y obtener sus resultados sin ningún conocimiento previo de seguridad ni de programación. La distinción de estos tres roles nos demuestra hasta que punto nuestra propuesta aporta flexibilidad y reutilización a las implementaciones, dividiendo el desarrollo de todo un sistema en componentes independientes que pueden ser utilizados por personas totalmente distintas. En la siguiente figura 2 podemos ver una representación esquematizada de nuestra metodología propuesta:

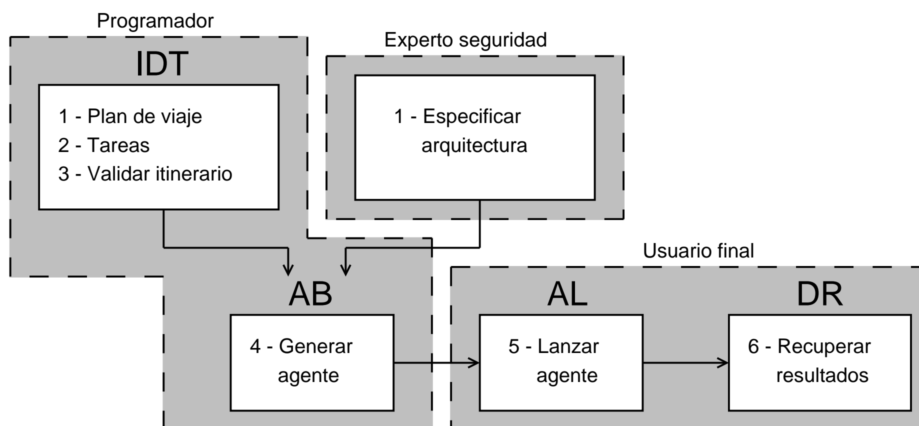

Figura 2. Metodología y roles en la generación automatizada de agentes.

4. Especificación de protocolos de protección de itinerarios

Estas cuatro herramientas que hemos presentado conforman un esquema de desarrollo que simplifica en gran medida el desarrollo de agentes móviles seguros. Como hemos visto, la simplificación más significativa se produce en el proceso de implementación de los mecanismos de protección del itinerario, lo cual se realiza a través de una especificación. Por supuesto, nuestro esquema prevé también la generación automatizada del código de control que gestiona el itinerario protegido y la ejecución del

agente. En la presente sección, mostraremos el motivo por el cual hemos creado un nuevo lenguaje de especificación y veremos qué requisitos este lenguaje debe cumplir.

Para simplificar la implementación de los mecanismos de seguridad asociados a una arquitectura debemos facilitar tanto el procesamiento del itinerario como la aplicación de mecanismos de seguridad sobre éste. Para ello, la primera opción que nos deberíamos plantear sería crear un *framework* de aplicación para este propósito. Este *framework* se podría implementar en un lenguaje orientado a objeto como Java como un conjunto de clases e interfaces, las cuales podrían ser utilizadas y extendidas para generar la implementación de la arquitectura final.

El problema de crear un *framework* de aplicación es que la implementación de nuestras arquitecturas no será reutilizable si en algún momento necesitamos cambiar el lenguaje de programación de nuestros agentes. En segundo lugar, el experto que diseña los mecanismos de seguridad no tiene porque ser un programador, con lo cual le puede resultar muy costoso utilizar el *framework* de aplicación si debe primero aprender a programar en un lenguaje de programación completo como Java o C++.

Por estos motivos decidimos definir un nuevo lenguaje de especificación diseñado específicamente para la implementación de arquitecturas criptográficas. Las implementaciones en este lenguaje serán siempre las mismas independientemente de si nosotros generamos el agente final en un lenguaje u otro. De esta manera, aunque cambiemos nuestro constructor de agentes para un entorno de plataformas diferente, nuestras especificaciones de las arquitecturas se mantendrán siempre igual. Además, al diseñador de arquitecturas le será mucho más fácil utilizar un lenguaje de especificación reducido con un conjunto de instrucciones de alto nivel pensado para la implementación de arquitecturas que no un lenguaje completo como Java o C++. A la vez, será también más difícil que introduzca errores en la programación.

Por lo tanto, una vez vista la necesidad de definir un nuevo lenguaje, analizamos los requerimientos que este lenguaje debería cumplir considerando nuestro principal objetivo: permitir la implementación de arquitecturas criptográficas de una manera fácil y flexible. En primer lugar, debería ser suficientemente genérico como para permitir la implementación de cualquier protocolo, sin imponer ninguna restricción. Esto no quiere decir que éste deba ser un lenguaje de propósito general. Al contrario, debería incluir el conjunto de instrucciones mínimas que permitiesen la representación de cualquier protocolo de protección. Esto podría ayudar a hacer el lenguaje lo más simple posible y, por lo tanto, las implementaciones serían también simplificadas en mayor medida. Para conseguir el máximo grado de simplificación, el último requerimiento es que el lenguaje proporcione instrucciones de muy alto nivel, para permitir implementar operaciones complejas (como las relacionadas con la criptografía) de la manera más fácil posible.

5. El lenguaje de especificación

El proceso de definición de este lenguaje a sido largo, y ha pasado por diferentes aproximaciones y pruebas de concepto. Una de nuestras primeras aproximaciones fue utilizar la misma representación que se ha utilizado en muchas ocasiones para definir arquitecturas: una gramática. Una gramática es un conjunto de reglas a partir de las cuales es posible generar todas las palabras de un lenguaje. Las gramáticas se utilizan

habitualmente en la Teoría de Autómatas para describir lenguajes. Un ejemplo de arquitectura que utiliza una representación en forma de gramática es la propuesta en [9] que podemos ver en la siguiente figura 3:

$$Itinerary_i = E_i\,(LocalCode_i\,,\,Data_i\,,\,Platforms_i\,,\\ E_{i+1}\,(Itinerari_{i+1}\,)\,)\quad|\quad Nil$$

Donde E_i es un cifrado utilizando la clave pública de la plataforma i.

Figura 3. Definición recursiva del protocolo propuesto en [9]

Una de las ventajas de las gramáticas es que permiten especificar como queremos que sea la estructura que guarda el itinerario sin necesidad de especificar los pasos o el algoritmo que acabe generando esta estructura. Además, esto produce representaciones muy compactas de las arquitecturas. Esta es la razón por la cual, en un principio, consideramos la posibilidad de representar las arquitecturas utilizando gramáticas. La idea es que la gramática genere la palabra que representa el itinerario protegido. Esta aproximación, sin embargo, presenta bastantes problemas. El problema principal es que no permite definir de manera no ambigua los protocolos, debido a que no permite tratar diferentes tipos de nodos. Este es el motivo por el cual hemos extendido esta representación basada en gramáticas añadiéndole más elementos. Las características principales del lenguaje de especificación que hemos creado finalmente se describen a continuación:

Los protocolos de protección del itinerario se especifican mediante un conjunto de reglas. Cada regla tiene un nombre en mayúsculas y un cuerpo que viene a continuación del símbolo '='. Debe existir siempre una regla con el nombre "ITINERARY", porque esta es la regla desde la cual el compilador empezará a procesar la especificación. En el cuerpo de cada regla se deben especificar las operaciones a realizar en el nodo actual. Para permitir la modularización del código, podemos hacer llamadas a otras reglas desde la actual. En la figura 4 podemos ver un ejemplo en el cual este lenguaje se utiliza para especificar el protocolo de protección anidada de itinerarios:

```
         ITINERARY = NEST#i
         NEST[SEQ] = aencrypt ( hostName#i, [ localCode#i : NEST#j ] )
   NEST[ALT_START] = aencrypt ( hostName#i, [ localCode#i : [ NEST#sf...sl ] ] )
 NEST[CLONE_START] = aencrypt ( hostName#i, [ localCode#i : [ NEST#sf...sl ] ] )
  NEST[CLONE_JOIN] = aencrypt ( hostName#i, [ localCode#i : NEST#j ] )
```

Figura 4. Especificación del protocolo de protección anidada de itinerarios.

Como podemos ver en la figura 4, la mayoría de elementos de nuestro lenguaje pueden tener un subíndice asociado a ellos. Los subíndices se separan de los elementos mediante el símbolo '#'. Se utilizan para especificar que el elemento está asociado con el nodo representado por el subíndice. Por ejemplo, el subíndice *j* representa al nodo sucesor del actual, por lo que si escribimos *hostName#j* estaremos haciendo referencia al nombre del *host* del nodo sucesor al actual.

Como podemos ver en la figura 4, más de una regla puede ser especificada con el mismo nombre, siempre y cuando se coloquen etiquetas diferentes entre corchetes ([ ]) después del nombre de la regla. Estas etiquetas hacen referencia al tipo de nodo. Por lo tanto, cuando el compilador encuentra diferentes reglas con el mismo nombre, escogerá una u otra en función del tipo que sea el nodo actual.

Además de estos dos elementos que acabamos de mencionar, nuestro lenguaje también proporciona otros elementos que pueden ser utilizados para hacer la programación más sencilla. Estos elementos, comunes en cualquier lenguaje de programación actual, pueden ser variables, llamadas a funciones externas con parámetros, creación de bucles mediante recursividad, y otros.

5.1. Operaciones criptográficas

Un subconjunto de las operaciones básicas estará destinado a los aspectos criptográficos de la arquitectura. En concreto, existen cinco operaciones que permitirán realizar las siguientes funciones: cifrar utilizando un criptosistema simétrico, cifrar en un esquema asimétrico, firmar, realizar un resumen *hash* y generar claves aleatorias. El cuadro siguiente muestra estos operadores con sus operandos. La implementación de cada uno de ellos variará en función del esquema de seguridad que se utilice para generar el agente (ver siguiente sección).

sencrypt Cifra el mensaje utilizando un criptosistema simétrico. Recibe como parámetros la clave y el mensaje.

aencrypt Cifra el mensaje utilizando un criptosistema asimétrico. Recibe como parámetros el nombre de la plataforma y el mensaje a cifrar.

sign Firma el mensaje con la clave secreta del usuario. Sólo recibe un parámetro, el mensaje.

hash Realiza un resumen (*hash*) del mensaje. El mensaje sobre el que realizar el resumen es el único parámetro.

keygen Genera una clave aleatoria. No recibe ningún parámetro.

Estas operaciones, juntamente con el resto, permiten especificar cualquier arquitectura criptográfica con sólo una restricción: la arquitectura debe utilizar siempre las mismas funciones para cifrar y calcular resúmenes. Todas las operaciones de cifrado o resumen en un misma arquitectura deben utilizar siempre el mismo algoritmo.

6. Implementación

La metodología descrita puede seguirse gracias a la herramientas diseñadas, tal como ha sido presentado hasta ahora. Para probar la viabilidad técnica de nuestra propuesta, y para realizar pruebas con aplicaciones reales, hemos realizado una implementación

de las herramientas básicas de soporte. Todas las herramientas han sido integradas en un único entorno de desarrollo (IDE), que podemos ver en la figura 5, desde el que el programador de aplicaciones puede acceder a todas las fases de la metodología. En elemento clave en la integración de estos módulos ha sido el diseño de la información que se intercambian, y que ha sido basada principalmente en XML. Todas las herramientas funcionan de manera independiente y autónoma, coordinadas desde el IDE diseñado. Desde éste IDE, además, es posible acceder a otros elementos del sistema, como el compilador de Java, o bibliotecas de seguridad específicas. Se ha primado siempre realizar un diseño abierto, en el que podrán integrarse nuevas herramientas a medida que se vayan implementando. El sistema de agentes móviles que usamos es una extensión del entorno de plataformas Jade [11], que soporta una movilidad segura y sigue los estándares definidos por FIPA [12]. Este entorno utiliza Java como el lenguaje de programación principal.

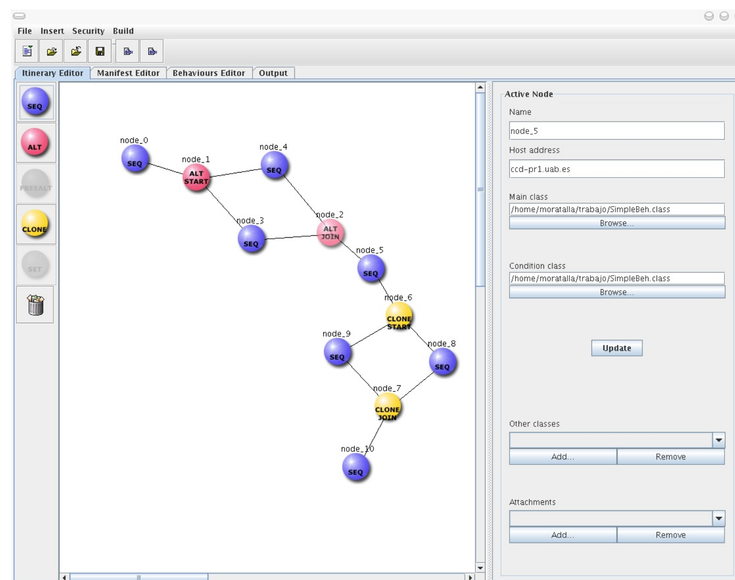

Figura 5. Entorno de desarrollo de agentes (IDE).

6.1. Herramientas principales

Tal como se ha descrito en secciones anteriores, disponemos de una herramienta para diseñar gráficamente el itinerario de un agente. El programador especifica de forma sencilla el itinerario, asignándole la tarea a ejecutar en cada plataforma. El código de la tarea se puede importar de algún archivo o se puede crear en la propia herramienta, que permite compilarlo. También ofrecemos la posibilidad de que el programador añada

recursos (a través de la inclusión de archivos) que el agente necesitará durante su recorrido. Cada recurso tendrá asignada una plataforma donde será utilizado. La interfaz gráfica ha sido diseñada utilizando *Java Swing*. Además del itinerario, el programador especifica que arquitectura quiere que protega el agente que acaba de crear, permitiendo la herramienta escoger solo aquellas arquitecturas que soportan todos los tipos de nodos que ha utilizado. En la especificación incluimos también toda la información del agente, como su nombre, su descripción, etc. La especificación del itinerario y toda esta información asociada se almacena en un archivo XML. Una vez creado este archivo, el programador puede guardarlo para usarlo más adelante, o bien generar el agente que ha especificado. Esta tarea se lleva a cabo en el módulo que explicaremos a continuación, que está plenamente integrado en nuestra herramienta gráfica.

El constructor de agentes es el encargado de construir el agente y dejarlo listo para ser enviado a la primera agencia de su recorrido. Observemos que hasta ahora, solo tenemos una especificación del itinerario del agente, no el agente mismo, pero ya disponemos de toda la información necesaria para construirlo aplicándole la arquitectura que el programador ha decidido. En la especificación que nos llega de la herramienta de diseño, se explicita la arquitectura escogida por el programador. Hay que notar que en el proceso de construcción del agente, dependiendo de la arquitectura seleccionada, utilizaremos unos algoritmos criptográficos o otros. Estos algoritmos están codificados en diferentes bibliotecas, y escogemos una u otra dependiendo de la elección del programador. Estos algoritmos criptográficos, necesitan acceder información relevante, como las claves públicas de las plataformas que visitaremos. Para poder acceder a estas claves, utilizamos el acceso a un directorio siguiendo en estándar de LDAP. Estos directorios almacenan estas claves, certificados y una relación de los esquemas criptográficos que implementa cada plataforma.

El constructor tiene como entradas la especificación del itinerario del agente y la especificación formal de la arquitectura seleccionada. Lo primero que debe hacer, es validar que ambas especificaciones son correctas y compatibles. Una vez validado el proceso básico es ir aplicando las reglas que formula la arquitectura al itinerario específico que estamos construyendo. Durante este proceso accedemos a la biblioteca criptográfica escogida y a la información de los servidores LDAP. La salida de nuestro constructor es un agente listo para ser lanzado. Su formato será un archivo JAR donde irá almacenado el agente (una instancia de un objeto concreto) y serializado con el itinerario protegido, el código de control necesario para extraerlo y todos los archivos auxiliares necesarios para su ejecución. Además añadimos un fichero *manifest* donde explicitamos información sobre el agente, como por ejemplo su nombre y su descripción.

Una vez tenemos nuestro agente creado, solo queda ponerlo en ejecución en la primera plataforma de su recorrido. para esto utilizamos la herramienta de lanzamiento de agentes. Este utiliza el servicio de movilidad interplataforma de Jade [13]. La idea es muy sencilla, consiste en emular el módulo de movilidad de una agencia cualquiera. El resultado será que esta herramienta se comunica con la primera plataforma donde queremos movernos y aplica el protocolo de movilidad. La plataforma destino atiende nuestra petición, recibe nuestro agente y comienza su ejecución si es aceptado.

6.2. El código de control

El código de control del agente ha de ser capaz de realizar diversas tareas. En primer lugar, se encarga de ver cual es la siguiente plataforma del itinerario y hacer que el agente salte a ella. En el caso de nodos alternativa, esto implicará ejecutar antes un método de condición que escogerá a cual de las siguientes plataformas posibles hay que saltar. En lo que se refiere a los nodos conjunto, el código de control tiene que generar un clon del agente para cada uno de los subitinerarios que se encuentran a continuación. Cuando estos clones lleguen al nodo unión, deben juntar todos sus resultados en un solo agente y este único agente es el que debe continuar la ejecución. De todo esto se tiene que ocupar también el código de control.

Otras tareas de este código están relacionadas con mecanismos de protección del agente. Por ejemplo, el código de control debe encargarse de que los resultados que se generan a lo largo de la ejecución no puedan ser modificados ni eliminados en otras plataformas. También puede ocuparse de ayudar a las plataformas a evitar ataques de *replay*, comprobando que el agente no está siendo reenviado a la misma plataforma.

Como podemos ver, estas tareas no dependen de los mecanismos que se utilicen para proteger el itinerario. Por lo tanto, el código que genera el constructor para estas tareas puede ser siempre el mismo. En cambio, para extraer la tarea local a ejecutar en cada plataforma se requiere un código que sí depende de como se haya protegido el itinerario. Por lo tanto, este código no puede ser siempre el mismo. Su generación es un punto en el cual aún estamos trabajando.

Nuestra intención es que el código de extracción de las tareas locales se pueda generar de forma automática. La idea es: a nosotros nos proporcionan una especificación a muy alto nivel donde se define la manera como se ha de proteger el itinerario. Entonces, mediante mecanismos de ingeniería inversa, queremos implementar un algoritmo que sea capaz de entender cómo se ha protegido cada una de las tareas del itinerario y que genere el código de desprotección y extracción de estas tareas.

Hacer la generación automática de esta parte del código de control permitiría al programador olvidarse completamente de todo aquello relacionado con el código de control. Esto supondría una simplificación en el desarrollo muy importante. En el caso que no consiguiéramos hacer la generación automática de este código, la alternativa sería ampliar el lenguaje de especificación para que, además de permitir especificar cómo se protege el itinerario, permita también especificar cómo se desprotege. En la implementación actual, el código de control se especifica como si fuera una tarea más del itinerario, proporcionándolo de manera independiente para que el constructor de agentes lo pueda diferenciar del resto de tareas.

6.3. Utilización de nuestro esquema para la protección de itinerarios dinámicos

Como ya hemos mencionado, nuestros agentes son creados usando itinerarios prefijados y explícitos. Sin embargo, la mayoría de las aplicaciones y la investigación que se lleva a cabo sobre agentes móviles considera que los itinerarios son dinámicos. Esto significa que los agentes tienen un conjunto de plataformas a través de las cuales se pueden mover y, a finalizar su ejecución en cualquiera de ellas, los agentes pueden escoger cualquier otra plataforma como siguiente. Evidentemente, estos itinerarios están

implícitos en el código. Para conseguir esta flexibilidad en agentes construidos a partir de itinerarios dinámicos, debemos redistribuir el código de nuestro agente de manera ligeramente distinta, tal y como veremos a continuación.

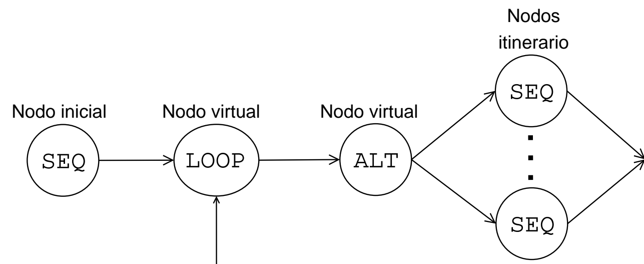

Figura 6. Estructura de un itinerario fijado que se comporta como un dinámico.

En primer lugar, debemos utilizar el concepto de *nodos virtuales*. Un nodo virtual es un tipo especial de nodo en el que no se especifica la siguiente plataforma a saltar. Más específicamente, lo que especifican estos nodos es que el agente debe permanecer en la misma plataforma del nodo en el que está para proseguir el itinerario. El uso de estos nodos virtuales permite definir un itinerario fijado que se comporte de la misma manera que uno dinámico, reorganizando el código del agente con el esquema de nodos que vemos en la figura 6.

Como muestra la figura, colocamos un nodo inicial desde el cual iniciar la ejecución del agente. Este nodo se fija de antemano. A continuación, colocamos dos nodos virtuales, un *bucle* y una *alternativa*. En el nodo alternativa colocaremos el código de decisión de salto a la siguiente plataforma y en el nodo bucle colocaremos el código de decisión sobre si debe finalizar la ejecución del agente en la última plataforma.

7. Conclusiones

Es este trabajo hemos presentado una metodología para el desarrollo de aplicaciones seguras basadas en agentes móviles. También se ha mostrado el conjunto de herramientas que hemos implementado para soportar esta metodología, y que están preparadas para ser usadas sobre la plataforma de agentes Jade. Estas contribuciones facilitan la tarea al desarrollador de aplicaciones que utilizan agentes móviles. Precisamente, uno de los elementos clave que frenan actualmente la utilización de esta tecnología es la complejidad para implementar los mecanismos de seguridad que garantizan la protección de los agentes móviles. A través del sistema descrito en este artículo hemos conseguido reducir enormemente esta dificultad, permitiendo el desarrollo rápido y cómodo de aplicaciones. Además de las herramientas básicas para el diseño de agentes y su construcción, se han desarrollado también otras destinadas al usuario final. El lanzador de agentes, por ejemplo, permite disponer de repositorios de agentes móviles listos para

ser enviados a una plataforma de ejecución; o el recuperador de resultados, que permite obtener los resultados parciales o finales de los agentes de manera síncrona o asíncrona.

Un aspecto destacable del esquema es su flexibilidad para utilizar e integrar nuevas arquitecturas criptográficas que aparezcan en el futuro, sin tener que modificar en absoluto ninguno de sus componentes. Bastará simplemente especificar la nueva arquitectura utilizando el lenguaje de especificación descrito en la sección 5, y ya podrá utilizarse desde el entorno de desarrollo implementado. La independencia de la metodología con la plataforma específica es otro aspecto notable de este trabajo, que hace que nuestro esquema sea altamente reusable, pudiéndose utilizar perfectamente en otras plataformas que sigan los estándares IEEE-FIPA y que permitan un modelo de protección orientado a agente.

Actualmente, estamos ampliando el entorno de desarrollo con nuevas herramientas que permitan la validación de los agentes móviles diseñados a través de la simulación, y de la validación formal de ciertas propiedades de seguridad, como que en ciertas plataformas el secreto de la información esté garantizado. La generación automática del código de control a partir de la especificación de la arquitectura criptográfica es un punto importante a solucionar para facilitar aun más la tarea del programador de mecanismos de protección. El seguimiento de agentes, y la incorporación de mecanismos de tolerancia a fallos en el esquema para la generación no solo de agentes seguros, sino fiables, son otros aspectos en los que estamos trabajando y que continúan la línea de este artículo.

8. Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia y Tecnología, a través de sus proyectos TIC-2003-02041 y TIN2004-20447-E , y de la Generalitat de Catalunya a través de la ayuda a grupos de investigación consolidados, SGR2005-00319.

Referencias

1. Robles, S., Ortega, J., Ametller, J.: Security Solutions and Killer Applications: Unlocking Mobile Agents. British Computer Society Expert Update **7**(3) (2004) 18–23
2. Mir, J., Borrell, J.: Protecting Mobile Agent Itineraries. In: Mobile Agents for Telecommunication Applications (MATA). Volume 2881 of Lecture Notes in Computer Science., Springer Verlag (2003) 275–285
3. Vigna, G.: Mobile Agents: Ten Reasons For Failure. In: Proceedings of 2004 International Conference on Mobile Data Management (MDM'04), IEEE Computer Society (2004)
4. Garrigues, C., Robles, S., Moratalla, A., Borrell, J.: Building Secure Mobile Agents using Cryptographic Architectures. In: 2nd European Workshop on Multi-Agent Systems. (2004) 243–254
5. Ahmed, T.M.: Generating Mobile Agent Securely by Using MASL. In: Proceedings of the First International Workshop on Services and Infrastructure for the Ubiquitous and Mobile Internet (ICDCSW'05), IEEE Computer Society (2005) 291–296
6. Ametller, J., Robles, S., Ortega, J.A.: Self-Protected Mobile Agents. In: AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, Washington, DC, USA, IEEE Computer Society (2004) 362–367

7. Karnik, N.M., Tripathi, A.R.: Security in the Ajanta mobile agent system. Software Practice and Experience **31**(4) (2001) 301–329
8. Roth, V.: Empowering Mobile Software Agents. In: Proc. 6th IEEE Mobile Agents Conference. Volume 2535 of Lecture Notes in Computer Science., Springer Verlag (2002) 47–63
9. Robles, S., Mir, J., Borrell, J.: MARISM-A: An Architecture for Mobile Agents with Recursive Itinerary and Secure Migration. In: 2nd. IW on Security of Mobile Multiagent Systems. (2002)
10. Farmer, W.M., Guttman, J.D., Swarup, V.: Security for mobile agents: Issues and requirements. In: Proceedings of the National Information Systems Security Conference. (1996) 591–597
11. JADE: Java Agent DEvelopment Framework. `http://jade.cselt.it` (2006)
12. IEEE FIPA: Foundation for intelligent physical agents. `http://www.FIPA.org` (2006)
13. Ametller, J., Robles, S., Borrell, J.: Agent Migration over FIPA ACL Messages. In: Mobile Agents for Telecommunication Applications (MATA). Volume 2881 of Lecture Notes in Computer Science., Springer Verlag (2003) 210–219