Generación de Agentes Móviles Seguros a Partir de Itinerarios y Arquitecturas Criptográficas

C. Garrigues, S. Robles, J. Borrell, A. Moratalla
Departament d'Informàtica, Universitat Autònoma de Barcelona
carles@ccd.uab.es

Resumen En este trabajo presentamos un esquema para la creación de agentes móviles seguros a partir de una representación del itinerario que deben seguir, el código a ejecutar, y una especificación de la arquitectura criptográfica con la que estarán protegidos. Este tipo de agentes es muy útil en aplicaciones seguras, pero su programación resulta muy complicada. El esquema propuesto facilita enormemente esta tarea, automatizando el proceso a partir de especificaciones concretas. En la práctica hemos conseguido un sistema de generación de código de fácil utilización que permite, no sólo la programación de agentes móviles seguros, sino también la creación sencilla de nuevos esquemas criptográficos para la protección del código y los datos de los agentes. El sistema puede usarse para desarrollar aplicaciones de propósito general basadas en el framework de ejecución de agentes Jade.

1. Introducción

Uno de los aspectos más importantes del nuevo paradigma de programación con agentes móviles [8] es la introducción de itinerarios explícitos. Estos itinerarios incorporan diferentes tipos de nodos (secuencias, alternativas, conjuntos, ...) para proporcionar flexibilidad al programador a la hora de definir el recorrido de los agentes [7]. Los agentes móviles con este tipo de itinerarios poseen una estructura separada del código donde se especifica el conjunto de plataformas que deben ser visitadas, el código de la tarea que debe ejecutarse en cada una de ellas, y la estrategia que determinará el recorrido final del agente. Esta estructura independiente hace posible una utilización flexible y extensible de los agentes y permite, al mismo tiempo, añadir mecanismos de seguridad para proteger tanto el código y los datos como el propio itinerario.

Los mecanismos de protección del agente deben asegurar la integridad, el secreto y la autenticidad del itinerario en los dos principales estados del agente móvil: durante el tránsito o migración, y mientras se ejecuta en una plataforma. Durante la migración, el itinerario no debe poder ser manipulado ni accedido, ni su origen falseado. Mientras el agente está en una plataforma, ésta sólo debe poder acceder a la parte del itinerario que le concierne, y no al resto. De esta manera, las aplicaciones basadas en agentes móviles son adecuadas en entornos donde existe una fuerte competencia entre las plataformas pero lealtad al usuario [2].

Una posible manera de guardar y proteger la información del itinerario dentro del agente se especifica mediante *arquitecturas criptográficas* [3]. Cada arquitectura criptográfica proporciona unos mecanismos concretos para la protección del itinerario. Estas

arquitecturas permiten desligar la programación del agente de la de los esquemas de seguridad. El uso de arquitecturas está indicado en aquellas aplicaciones que requieren la utilización de agentes móviles y al mismo tiempo un alto grado de seguridad.

Aunque el uso de estas arquitecturas para la protección del itinerario es conveniente, existen algunos problemas. La implementación de agentes móviles basados en arquitecturas criptográficas resulta excesivamente costosa debido, principalmente, a los mecanismos de seguridad asociados. Fácilmente la implementación de estos mecanismos puede resultar más costosa que la propia aplicación. Estos mecanismos pueden necesitar, por ejemplo, obtener certificados de plataformas, o realizar funciones criptográficas para descifrar parte del itinerario en una plataforma [1]. Esto hace que la programación de la arquitectura requiera el uso de repositorios de claves, el conocimiento detallado de las plataformas en las que se ejecutan los agentes y, en general, un nivel de conocimientos de criptografía elevado.

Otro problema es la poca reutilización de estas implementaciones. Debido a que los mecanismos de seguridad que se implementan normalmente se utilizan para proteger un itinerario concreto, la implementación no puede volver a ser utilizada para proteger otro itinerario distinto. Si, como acabamos de mencionar, la implementación de estos mecanismos requiere los mayores esfuerzos, la falta de reutilización se convierte en un serio problema del esquema de desarrollo actual.

En la práctica, las arquitecturas criptográficas no son utilizadas ya que, pese a sus numerosas ventajas respecto a otros esquemas menos flexibles, como [6], no pueden ser implementadas directamente. Esto supone un fuerte freno al desarrollo de este tipo de aplicaciones y al despliegue completo del nuevo paradigma de programación basado en agentes móviles.

En este trabajo presentamos un novedoso esquema que permite la generación automática de agentes móviles seguros con arquitecturas criptográficas. De esta manera se simplifica significativamente el desarrollo de aplicaciones seguras. La pieza principal del esquema propuesto es un generador de código seguro, que a partir de la especificación del itinerario y la especificación de la arquitectura que debe protegerlo, genera el código del agente móvil final. En el esquema global otros componentes se encargan de asistir el diseño del itinerario y de enviar el agente seguro a una plataforma para su ejecución.

El esquema presentado está siendo desarrollado en el momento de escribir este artículo y ya existe una prueba de concepto funcionando que valida la viabilidad de la propuesta. Para llevar a cabo este prototipo hemos diseñado especificaciones para describir el itinerario y las arquitecturas criptográficas, hemos implementado un generador de código seguro, y un lanzador de agentes.

La siguiente sección de este trabajo presenta el escenario global donde se encuadra la generación de agentes seguros. En la tercera sección se introduce el lenguaje de especificación de arquitecturas que permitirá reducir la complejidad de la generación. Posteriormente, se muestran algunos detalles de la implementación y las conclusiones del trabajo.

2. Escenario

Nuestro esquema de desarrollo de agentes móviles parte de un itinerario y de una arquitectura criptográfica. El objetivo es construir un agente móvil encapsulando el itinerario según la arquitectura proporcionada, y ejecutarlo. Este esquema de desarrollo utiliza tres componentes básicos: la interfaz gráfica de generación de itinerarios (IGGI), el generador de código (GC), y el lanzador de agentes. En la figura 1 podemos ver un esquema de los componentes que forman nuestro marco de desarrollo.

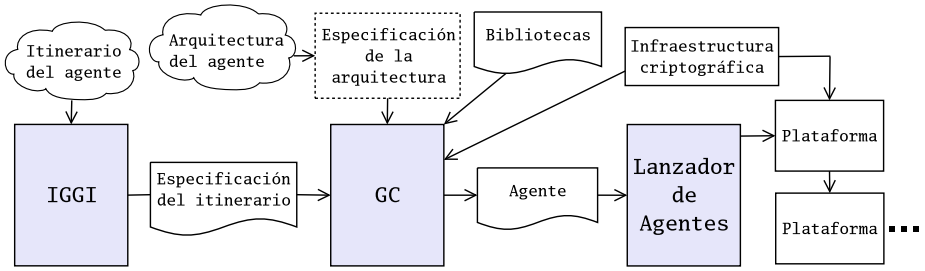

Figura1. Componentes del esquema de desarrollo

La simplificación del proceso de desarrollo tiene como elemento central el generador de código del agente. Las entradas que requiere este generador para construir el agente, el itinerario y su arquitectura, se proporcionarán mediante especificaciones. La primera especificación, la del itinerario, será generada de forma asistida por una interfaz gráfica de generación de itinerarios. La segunda especificación, la de la arquitectura, será desarrollada por el programador. A partir de estos dos componentes, se construirá el agente y será entregado a un lanzador de agentes, el cual se encargará de ponerlo en ejecución en la primera plataforma del itinerario.

La especificación de la arquitectura del agente describe la estructura y los mecanismos de protección necesarios para encapsular el itinerario de forma segura dentro del agente. La especificación del itinerario contiene, en primer lugar, una descripción de la topología de la red de plataformas por las que el agente se puede mover. Para cada plataforma de la red, especifica también el código local a ejecutar en esa plataforma y, en caso de tener que tomar una decisión sobre la próxima plataforma a saltar, como es el caso de los nodos alternativa, especifica el código de decisión de salto.

El objetivo inicial de este trabajo, la simplificación del desarrollo de aplicaciones basadas en agentes móviles, se consigue principalmente gracias a la implementación de los mecanismos de protección del itinerario por medio de especificaciones. La especificación de la arquitectura permite representar operaciones complejas de forma simple, haciendo posible, por ejemplo, el uso de una sola operación para hacer un cifrado con la clave pública de una plataforma.

Otro de los aspectos clave de nuestro trabajo es que el desarrollo de las aplicaciones se divide en dos partes totalmente independientes: la generación de la especificación

del itinerario y la generación de la especificación de la arquitectura. Esta división aporta flexibilidad y mayor capacidad de reutilización, gracias a que una cierta especificación del itinerario se puede utilizar con cualquier especificación de una arquitectura y a la inversa. Los tres componentes básicos del escenario –el generador de código, la interfaz gráfica y el lanzador de agentes– se describen a continuación.

El generador de código

La herramienta fundamental del esquema de desarrollo es el generador automático del código del agente. Este generador utilizará la especificación del itinerario y la especificación de la arquitectura y construirá un itinerario protegido que se almacenará dentro del agente. El agente será entonces capaz de recorrer todas las plataformas del itinerario y realizar la tarea asignada en cada una de ellas según corresponda.

Las operaciones disponibles en la especificación de la arquitectura se implementarán en bibliotecas de operaciones. Una misma operación podrá tener diferentes implementaciones en bibliotecas diferentes, ganando flexibilidad a la hora de generar el código de los agentes. Por ejemplo, podremos especificar al generador de código que utilice una biblioteca u otra en función de la infraestructura criptográfica que se requiera para el almacenamiento y la emisión de certificados.

El lanzador de agentes

El lanzador de agentes será el encargado de enviar el agente generado a la primera plataforma del itinerario para que empiece su ejecución. Esta herramienta está dividida en dos módulos. El primero está fuera de la plataforma y es el invocado por el usuario directamente. Este módulo es muy ligero y puede ejecutarse incluso en dispositivos móviles. El segundo módulo está dentro de la plataforma y está en contacto directo con los mecanismos necesarios para la creación de agentes. Cuando recibe un agente lo registrará en la plataforma y empezará su ejecución.

Una de las ventajas de esta herramienta es que soluciona una limitación común de las plataformas actuales: una vez ha sido arrancada una plataforma, sólo ésta puede iniciar la ejecución de nuevos agentes, utilizando habitualmente un agente especializado. Mediante nuestro lanzador, los agentes pueden ser iniciados en cualquier plataforma de manera remota.

La interfaz gráfica

Para facilitar más aún el trabajo del programador, nuestro esquema incorpora una interfaz gráfica generadora de itinerarios. En esta interfaz el programador definirá gráficamente el conjunto de plataformas que componen el itinerario y asignará a cada una de ellas el código local correspondiente. A partir de aquí, esta herramienta generará la especificación del itinerario.

El formato de la especificación del itinerario será estándar para todas las arquitecturas. Además, la especificación no determinará el tipo de arquitectura que debe proteger el itinerario. Sin embargo, una determinada arquitectura podrá imponer restricciones respecto al tipo de itinerarios que puede soportar. Por ejemplo, podría existir una arquitectura muy simple que no soportase la protección de itinerarios con alternativas.

Por este motivo, en el momento de crear un nuevo itinerario en la interfaz gráfica, deberá indicarse la arquitectura que se utilizará posteriormente para protegerlo y, de esta manera, esta herramienta se asegurará de que la arquitectura soporta el itinerario introducido. Sin embargo, para hacer la generación de código final, se podrá utilizar cualquier otra arquitectura que soporte también el itinerario especificado.

3. El lenguaje de especificación de arquitecturas criptográficas

Una vez visto el escenario global en la sección anterior, pasamos a analizar con detenimiento las cuestiones relacionadas con el componente principal: el Generador de Código. La principal entrada de este módulo, además del itinerario, es la especificación de la arquitectura criptográfica con la que deberá ser protegido. Si bien a nivel teórico parece suficiente disponer de la gramática que representa los mecanismos de protección, en la práctica es necesario proporcionar, además, información referente al proceso de codificación. En esta sección se detalla como a través de un lenguaje para la especificación de la arquitectura es factible la creación automática de los agentes seguros.

La especificación de la arquitectura que protege el itinerario se puede llevar a cabo utilizando la representación empleada hasta el momento en los trabajos previos [4]. Una de las características principales de esta representación es que permite especificar como queremos que sea la estructura que guarde el itinerario sin necesidad de especificar los pasos o el algoritmo que acaben generando esta estructura. En la figura 2 podemos ver un ejemplo en el que se utiliza esta representación para especificar el diseño de la arquitectura con itinerario recursivo.

$$
\begin{aligned}
\text{Agent} &= (\ \text{ControlCode},\ \text{Results},\ \text{Itin}\ ) \\
\text{Itin} &= \text{Itin}_{\text{Seq}} \mid \text{Itin}_{\text{Alt}} \mid \text{Itin}_{\text{Set}} \\
\text{Itin}_{\text{Seq}} &= Epk_{\text{Seq}}\ (\ \text{SEQ}\ (\ \text{LocalCode},\ \text{NextPlatformName}\ ),\ \text{Itin}\ ) \\
\text{Itin}_{\text{Alt}} &= Epk_{\text{Alt}}\ (\ \text{ALT}\ (\ \text{LocalCode},\ \text{NextPlatformName},\ \text{Struct}\ )\ ) \\
\text{Itin}_{\text{Set}} &= Epk_{\text{Set}}\ (\ \text{SET}\ (\ \text{LocalCode},\ \text{NextPlatformName},\ \text{Struct}\ )\ ) \\
\text{Struct} &= \text{Itin},\ \text{Struct} \mid \text{Itin}
\end{aligned}
$$

Figura2. Diseño de la arquitectura con itinerario recursivo.

En estas expresiones se define la estructura general del agente, el cual consta de tres componentes esenciales: el código de control (*ControlCode*), los resultados (*Results*) y el itinerario protegido (*Itin*). El código de control es aquel común a todas las plataformas, encargado de regular el movimiento del agente a lo largo del itinerario. Los resultados se almacenarán también en una estructura independiente para poder ser recuperados cuando el agente acabe su recorrido. Por último, el itinerario protegido se guarda en una estructura recursiva denominada *Itin*, que contiene la información de cada plataforma cifrada con su clave pública –función *Epk*–.

La estructura *Itin* está formada por el código local (*LocalCode*), el nombre de la siguiente plataforma (*NextPlatformName*) y una estructura cifrada con la información del resto del itinerario, que puede ser más o menos compleja en función de si se trata de un nodo secuencia o un nodo alternativa o conjunto.

A pesar de ser una representación muy adecuada para definir arquitecturas, la generación de código a partir de ésta puede resultar muy complicada. El motivo es precisamente que esta especificación no muestra los pasos necesarios para generar la estructura del itinerario protegido, lo que implica que el algoritmo que construye esta estructura debe generarse automáticamente.

Para evitar una implementación excesivamente compleja, hemos optado por la definición de un lenguaje de especificación de arquitecturas que represente en cierto modo los pasos que se deben seguir para construir el itinerario. Este lenguaje se ha denominado CASL (*Cryptographic Architectures Specification Languaje*).

Las características deseables de este nuevo lenguaje son las siguientes: En primer lugar, debe ser suficientemente genérico como para permitir la implementación de cualquier arquitectura, sin imponer ninguna restricción. En segundo lugar, debe estar orientado a la implementación de arquitecturas, incluyendo el conjunto de instrucciones mínimas que aseguren la capacidad de representación de cualquier arquitectura. Por último, debe simplificar la implementación de arquitecturas, proporcionando operaciones de muy alto nivel que permitan hacer operaciones complejas de forma simple.

3.1. Características del lenguaje

El lenguaje que hemos diseñado divide el código en bloques de instrucciones. El bloque de instrucciones que se ejecuta en cada momento depende de la posición en la que nos encontremos dentro del recorrido del itinerario en el momento de construirlo. Estos bloques de código contienen las operaciones necesarias para encapsular la información de cada tipo de nodo dentro de la estructura que define la arquitectura.

De esta manera, el flujo de ejecución viene determinado por el recorrido del itinerario que debe seguir el agente. Los diferentes instantes del recorrido determinarán la ejecución de su bloque de instrucciones correspondiente, por lo que podemos pensar en éste como un lenguaje 'orientado a itinerarios'.

El lenguaje está basado en una pila. Las instrucciones del lenguaje son en realidad operadores o operandos que se introducen en la pila de forma implícita. Los operadores sacan de la pila sus operandos y devuelven a ésta el resultado de la operación.

El itinerario se recorre empezando por el final. El motivo de esta decisión es poder disponer de toda la información relativa a los sucesores de un nodo en el momento de hacer el tratamiento de dicho nodo. Esto es indispensable debido a que la información que interesa al agente en una determinada etapa del itinerario hace siempre referencia a etapas posteriores del mismo. Por ejemplo, el agente necesita disponer del nombre de las plataformas sucesoras a partir de la actual para poder continuar el itinerario en todo momento.

La consecuencia de este recorrido inverso es que las operaciones que utiliza la arquitectura pueden disponer de los operandos necesarios generados en etapas anteriores. De esta manera, se evita significativamente el crecimiento excesivo de la pila.

El lenguaje proporciona un conjunto de operandos que toman su valor automáticamente en función del nodo que se esté tratando en cada momento. Estos operandos aportan toda la información asociada a un nodo y forman parte del lenguaje. Esta información incluye el nombre de la plataforma, el código local y el nombre o la lista de nombres de las plataformas sucesoras en el itinerario.

El lenguaje no dispone de estructuras de control. Esta decisión está motivada por el hecho de que, en la implementación de arquitecturas, las estructuras de control sólo se utilizan para permitir el recorrido por los diferentes nodos del itinerario. En nuestro lenguaje, el conjunto de bloques de código disponible permite añadir las operaciones requeridas en cualquiera de los diferentes puntos del recorrido del itinerario, haciendo innecesarias las estructuras de control. En la figura 3 podemos ver un itinerario genérico en el que se muestran los diferentes tipos de nodos y los bloques de código asociados a cada uno de ellos según el punto del recorrido.

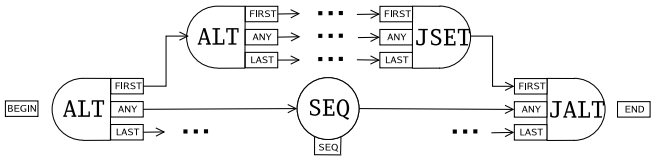

Figura3. Bloques de código de un itinerario genérico.

Los bloques de código donde el programador incluye las operaciones están representados en esta figura mediante etiquetas adjuntas a cada tipo de nodo. En el lenguaje CASL, estos bloques se designan utilizando el nombre de las etiquetas de la figura 3. En la figura 4 podemos ver un ejemplo de código en CASL con las diferentes operaciones agrupadas en los bloques de código, en función del instante en el que deban ejecutarse dentro del recorrido del agente.

```
SEQ { LocalCode;
      NextPlatformName;
      3; composeN;
      PlatformName;
      asymmetricCipher; }
JALT {
    LAST {
      LocalCode;
      NextPlatformName;
      3; composeN;
      PlatformName;
      asymmetricCipher;
      dup; }}
JSET {...} //Igual que el JALT
ALT {
   FIRST {
      addToList;
      2; dropN;
      LocalCode;
      NextPlatformName;
      3; composeN;
      PlatformName;
      asymmetricCipher; }
   ANY {
     addToList;
     2; copyN; }
   LAST {
     createList;
     2; copyN; }}
SET {...} //Igual que el ALT
END { Null; }
```

Figura4. Especificación en CASL de la arquitectura con itinerario recursivo.

Las instrucciones que observamos en la figura 4 corresponden a las operaciones del diseño de la arquitectura representada en la figura 2. Por ejemplo, para los nodos de tipo secuencia (bloque `SEQ`), la expresión

$$Epk_{\mathrm{Seq}}\ (\ \mathrm{SEQ}\ (\ \mathrm{LocalCode},\ \mathrm{NextPlatformName}\ ),\ \mathrm{Itin}\ )$$

se traduce en introducir en la pila el `LocalCode` y el `NextPlatformName`, y componerlo con la estructura generada en el paso anterior formando un solo objeto en el *top* de la pila. Luego, este objeto se cifra con la clave pública de la plataforma mediante la operación `–asymmetricCipher–`, a la cual le pasamos el nombre de la plataforma introduciéndolo previamente en la pila.

El hecho de utilizar un lenguaje orientado a pila tiene la ventaja de que es una estructura de datos muy simple que luego será fácil de implementar en cualquier lenguaje de programación. El lenguaje no es evidentemente de propósito general, sino que está orientado al diseño de arquitecturas. El conjunto de operaciones disponibles inicialmente estará orientado a formar estructuras y a aplicar funciones criptográficas que las protejan. Sin embargo, los programadores de agentes podrán añadir nuevas funcionalidades a este lenguaje mediante la programación de nuevas bibliotecas de operaciones.

A parte del conjunto de operaciones que codifican los mecanismos de protección de una arquitectura, el lenguaje dispone también de unas directivas que permiten especificar información relacionada con la arquitectura, como los tipos de nodos que ésta soporta, los bloques de código asociados a cada tipo de nodo, etc. También permite definir nuevos tipos de nodos. Por ejemplo, una arquitectura podría definirse un nuevo tipo de nodo para formar itinerarios con bucles.

3.2. Operaciones criptográficas

Un subconjunto de las operaciones básicas estará destinado a los aspectos criptográficos de la arquitectura. En concreto, existen cinco operadores 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 1 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). Como el resto de operadores, obtendrán los operandos necesarios directamente de la pila de ejecución.

Estos operadores, juntamente con el resto, permiten especificar cualquier arquitectura criptográfica con sólo una restricción: todas las operaciones de cifrado o resumen en un misma arquitectura deben utilizar siempre el mismo algoritmo.

4. Implementación y resultados

En el momento de escribir este artículo se dispone de la implementación de un prototipo del esquema propuesto, que proporciona una prueba de concepto de los componentes principales. Como plataforma de ejecución de agentes se ha tomado una extensión del *framework* Jade [5] que soporta la movilidad segura. Esta plataforma utiliza Java como principal lenguaje de desarrollo.

Operador	Función	Operandos
$sencrypt$	Cifra el mensaje utilizando un criptosistema simétrico	Clave, Mensaje
$aencrypt$	Cifra el mensaje utilizando un criptosistema asimétrico	Nombre plataforma, Mensaje
$sign$	Firma el mensaje con la clave secreta del usuario	Mensaje
$hash$	Realiza un resumen (*hash*) del mensaje	Mensaje
$keygen$	Genera una clave aleatoria	—

Cuadro1. Operadores criptográficos de CASL

El principal componente implementado es el generador de agentes móviles seguros, descrito en esta sección. Además, se han realizado las implementaciones de la interfaz gráfica para el diseño de itinerarios y el lanzador de agentes.

El generador de código recibe de la interfaz de generación de itinerarios la especificación del recorrido que deberá realizar el agente, la arquitectura que deberá utilizar, el esquema de seguridad y otras informaciones (nombre del agente, descripción, identificador de la clave secreta del programador, etc.). Toda esta información está contenida en una especificación XML. Además, al generador se le proporcionan todas las clases correspondientes a las tareas que el agente debe realizar en cada etapa del itinerario. Estas clases están agrupadas por parejas formadas por una clase principal y archivo JAR. Cada pareja corresponde a una plataforma del itinerario, donde la clase principal contiene la tarea que el agente debe lanzar en primer lugar y el archivo JAR contiene el resto de clases necesarias para su ejecución.

La primera función que realiza el generador de código es la validación de las especificaciones del itinerario y de la arquitectura. Una vez terminadas las validaciones, el generador construye el itinerario protegido recorriéndolo a partir de su especificación y procesando cada nodo de la manera indicada en la especificación de la arquitectura.

Las operaciones del lenguaje de especificación de arquitecturas están implementadas en las bibliotecas proporcionadas al generador de código. La biblioteca donde se implementan las operaciones criptográficas se identifica mediante el nombre del esquema de seguridad que utiliza. Desde esta biblioteca se accede a las infraestructuras necesarias para realizar las operaciones solicitadas. Por ejemplo, la instrucción de cifrar asimétricamente (`aencrypt`) puede consultar un servidor de LDAP para obtener los certificados X.509 con las claves públicas de las plataformas, o bien acceder a un anillo local de PGP con el mismo objetivo. Hacerlo de una forma u otra dependerá de la biblioteca que le hayamos proporcionado al generador.

Una vez construida la estructura del itinerario protegido, se crea la instancia del agente final. Para ello, se incorpora al agente un código especial para extraer de la estructura que almacena el itinerario el código local correspondiente a cada plataforma. Éste es el denominado “código de control”. Este código realiza el proceso inverso al de construcción del agente seguro, descifrando las partes protegidas y haciendo posible la ejecución del código original del agente. La generación del código de control se podrá hacer a partir de la especificación en CASL de la arquitectura.

A partir del código del agente y de la instancia del itinerario protegido se construye una instancia del agente móvil final. Esta instancia es serializada y se incluye en un archivo JAR que contiene, también, el código compilado de la clase del agente. Este archivo JAR es el que se introducirá en el lanzador de agentes para que finalmente sea puesto en ejecución.

5. Conclusiones

En el presente trabajo hemos presentado una solución al problema de la implementación de agentes móviles basados en arquitecturas criptográficas. Las arquitecturas de agentes nos permiten proteger el itinerario de los agentes móviles mediante el uso de mecanismos criptográficos. La implementación de estos mecanismos resulta demasiado compleja para que el programador la lleve a cabo directamente. Nuestra solución facilita este desarrollo a través de un generador de agentes móviles seguros.

Un elemento clave en la simplificación del proceso de generación de agentes es el uso de especificaciones para representar tanto las arquitecturas criptográficas como los itinerarios. La especificación de las arquitecturas se realizan en un lenguaje con operaciones de alto nivel respaldado por un conjunto de bibliotecas que proporcionan la implementación de estas operaciones.

Las ventajas de nuestra propuesta no se limitan únicamente a la simplificación del desarrollo, sino que también incluyen otros aspectos destacables como la reusabilidad del código y la flexibilidad. La reusabilidad es clara si tenemos en cuenta que ahora las especificaciones del itinerario y de la arquitectura se realizan de forma totalmente independiente. Ésto permite utilizar una misma implementación de la arquitectura para generar diferentes agentes a partir de itinerarios distintos. De igual modo, el itinerario puede reutilizarse para crear agentes basados en cualquier esquema de protección. Además, es posible generar diferentes implementaciones de un mismo agente cambiando únicamente las bibliotecas de operaciones proporcionadas al generador de código. El esquema resultante es, por tanto, muy flexible.

A partir del esquema propuesto se ha implementado un prototipo que permite generar, sin esfuerzo, agentes móviles seguros a partir de especificaciones de alto nivel.

Las líneas de continuación derivadas de este trabajo incluyen, entre otros, un repositorio de arquitecturas, que permitiría centrar el desarrollo en la funcionalidad de las aplicaciones, la verificación de las arquitecturas e itinerarios, y la simulación para validar las aplicaciones antes de su despliegue final.

6. Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia y Tecnología, a través de sus proyectos TIC-2003-02041 y TIC2001-5108-E, y de la Generalitat de Catalunya a través de la ayuda del DURSI a grupos de investigación consolidados, 2001-SGR-00219.

Referencias

1. J. Ametller, S. Robles, and J. A. Ortega-Ruiz. Self-protected mobile agents. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 362–367, Washington, DC, USA, 2004. IEEE Computer Society.
2. W. M. Farmer, J. D. Guttman, and V. Swarup. Security for mobile agents: Issues and requirements. In *proceedings of the National Information Systems Security Conference*, pages 591–597, 1996.
3. J. Mir and J. Borrell. Protecting mobile agent itineraries. In *Mobile Agents for Telecommunication Applications (MATA)*, volume 2881 of *Lecture Notes in Computer Science*, pages 275–285. Springer Verlag, October 2003.
4. S. Robles, J. Mir, J. Ametller, and J. Borrell. Implementation of secure architectures for mobile agents in marism-a. In *Mobile Agents for Telecommunication Applications (MATA)*, volume 2521 of *Lecture Notes in Computer Science*, pages 182–191. Springer Verlag, 2002.
5. S. Robles, J. Mir, and J. Borrell. Marism-a: An architecture for mobile agents with recursive itinerary and secure migration. In *2nd. IW on Security of Mobile Multiagent Systems*, Bologna, July 2002.
6. V. Roth. Empowering mobile software agents. In *Proc. 6th IEEE Mobile Agents Conference*, volume 2535 of *Lecture Notes in Computer Science*, pages 47–63. Spinger Verlag, 2002.
7. M. Straßer, K. Rothermel, and C. Maiöfer. Providing Reliable Agents for Electronic Commerce. In *Proceedings of the International IFIP/GI Working Conference*, volume 1402 of *Lecture Notes in Computer Science*, pages 241–253. Springer-Verlag, 1998.
8. James E. White. Telescript technology: Mobile agents. In Jeffrey Bradshaw, editor, *Software Agents*. AAAI Press/MIT Press, 1996.