J Comput Virol (2007) 3:267–283
DOI 10.1007/s11416-007-0056-y

ORIGINAL PAPER

TAMAP: a new trust-based approach for mobile agent protection

Salima Hacini · Zahia Guessoum · Zizette Boufaida

Received: 31 January 2007 / Revised: 24 May 2007 / Accepted: 4 June 2007 / Published online: 28 June 2007

Abstract Human activities are increasingly based on the use of distant resources and services, and on the interaction between remotely located parties that may know little about each other. Mobile agents are the most suited technology. They must therefore be prepared to execute on different hosts with various environmental security conditions. This paper introduces a trust-based mechanism to improve the security of mobile agents against malicious hosts and to allow their execution in various environments. It is based on the dynamic interaction between the agent and the host. Information collected during the interaction enables generation of an environment key. This key allows then to deduce the host's trust degree and permits the mobile agent to adapt its execution accordingly to the host trustworthiness, its behavior history and the provided Quality of Service (QoS). An adaptive mobile agent architecture is therefore proposed. It endows the mobile agent with the ability to react with an unexpected behavior.

1 Introduction

New information systems and recent applications are often distributed and characterized by a dynamic, unpredictable and aggressive environment. Furthermore, distributed applications such as telecommunication systems, information management, on-line auctions or service brokering are now increasingly being designed as a set of mobile agents. These mobile agents are active and autonomous software entities that can suspend their behavior, move to another host of the network, and continue their activity, deciding where to go and what to do along the way [21,44]. They provide several advantages to design and control distributed applications such as autonomy, dynamic adaptation, data and control distribution, a better use of the network resources and communication reduction with respect to latency, bandwidth and connection time. The benefits from using mobile agents are great. However, mobile agents bring two kinds of serious security risks: the malicious mobile agent and the malevolent host threats.

Research works related to the security of the mobile agents follow thus two aspects. The first aspect concerns the protection of the host against malicious mobile agents while the second aspect concerns the protection of the mobile agents against malevolent visited hosts. The host protection has been a subject of a large interest and has emphasized a number of techniques such as code signing [40] sandboxing [19, 44], proof carrying code [38] state appraisal [14] and path histories [11]. These techniques provide an acceptable host security level. However, the mobile agent protection against malicious host remains an open issue and represents the scope of this paper.

Malicious host may try to attack mobile agent in order to obtain service without providing payment or to remove private information from the agent's memory. Other examples of such attacks are malicious alteration of its code and control of its execution. Attacks can also be due to a host who tries to analyze the behavior of a malicious agent. It is the case, for example, of a malware (e.g., the mobile agent) and the anti-virus (e.g., the host with a sandbox) [3]. The agent is vulnerable while it is running on the host's execution platform.

S. Hacini (✉) · Z. Boufaida
LIRE, Mentouri University, Constantine, Algeria
e-mail: salimahacini@gmail.com

Z. Boufaida
e-mail: zboufada@gmail.com

Z. Guessoum
LIP6, Pierre et Marie Curie University, Paris, France
e-mail: zahia.guessoum@lip6.fr

Its owner therefore requires some guarantees concerning the protection of the agent against malicious host threats. Thus, the mobile agent has to protect itself from any act aiming at the deterioration, the destruction or the handling of its code, its state or its data.

The protection of mobile agents against malicious hosts' behaviors represents a challenging research area [3,5–7,16,30,44]. Several approaches have been introduced such as tamper proof hardware [24,48,55], function hiding [46,47], black box [25] or clueless agents [41]. These approaches help to enhance the security of code executing in an untrusted environment. Nevertheless, they present different disadvantages. For instance, the tamper proof hardware has a prohibitive cost and the function hiding approach is not suitable for any kind of mobile agent task. It is restricted to the polynomial and rational functions. So, an important challenge is to introduce a solution with a reasonable cost and an acceptable level of security. The protection approach presented here has a dual interest since it helps the trusted mobile agent to protect itself against malicious hosts' attacks and could also be exploited by malicious mobile agent to resist customer's analysis (e.g., AV software).

This paper deals with the protection of the mobile agent behavior from any analysis aiming to divulge it. For this purpose, we propose an adaptive mobile agent architecture which offers the mobile agent the ability to react with an unexpected behavior. The mobile agent reaction is based on its ability to estimate the host trustworthiness, to control its own behavior history and to exploit the various Qualities of Service provided by the application. The host trustworthiness estimation is based on the diagnostic of an environment key. The environment key generation is derived from the information collected during the dynamic interaction between the mobile agent and the host.

The paper is organized as follows: Sect. 2 exposes some related work from which we inspired our idea. Section 3 gives an overview of the proposed approach. Section 4 describes the trust estimation process. Section 5 shows the adaptive mobile agent architecture and outlines the functionality of its principal components. Section 6 gives further details about the mobile agent implementation. Finally, Sect. 7 summaries our contribution and describes the future work.

2 Related work

The technology of mobile agents has introduced some serious problems and has emphasized existing security issues. Several works have underlined the importance of protecting host against malicious mobile agents [5–7,30]. This problem is out of the scope of this paper. Moreover, mobile agents become especially vulnerable when traveling among hosts network. The protection of the mobile agents against malicious hosts is thus an open issue [5,6].

This section presents the most important techniques for providing security in the protection of mobile agents. It also underlines the evolution of the use of the trust concept as factor enhancing security in mobile agent based systems.

2.1 Agent code protection

The mobile agents are exposed to various threats from hosts they visit. This problem is difficult because the visited host has a full control on the mobile agent execution. Several approaches have been proposed. They principally aim at making the attacks useless or detectable. Among the existing approaches, we find:

Cryptographic approaches

Examples of such approaches are execution tracing or hiding function. The execution tracing approach enables detection of any possible visited host misbehavior such as the modification of the mobile agent code, state, and execution flow. It is based on cryptographic traces that are collected during an agent's execution on different hosts. It has some limitations, such as the potential large size. Another limitation of this approach is that the agent's owner needs to wait until it obtains suspicious results in order to run the verification process. Also, this technique is considered to be too difficult to use in the case of multi-threaded agents [49,50]. A new version of the execution tracing technique, proposed by Tan and Moreau [49] modifies the original approach by assigning the trace verification process to a trusted third party, the verification server, instead of depending on the agent's owner.

In hiding function approach, Sander and Tschudin [46,47], describe an approach to code protection that relies on the execution of encrypted functions. This approach is based on executing a program embodying an encrypted function on a mobile agent platform. The mobile agent code is a kind of encrypted program which can be executed on encrypted data without decryption of code and data at all. This approach ensures that the platform does not learn anything substantial about the encrypted function. Thus, it offers the confidentiality of the execution. However, the approach remains theoretical and coding is applied only on a restricted whole of functions (polynomial and rational functions).

Environment key

Riordan and Schneier [41] use data of current environment to construct a decryption key. When the proper environment information is located, the key is generated, the cipher-text is decrypted, and the resulting plain-text is acted upon. Neither

can the mobile agent precisely predict its own execution at the receiver host, nor can the host foresee the incoming agent task. The approach allows the agent owner to specify some constraints on the environment where the mobile agent will execute. Filiol [15] uses the environment key as a basic technique to forbid a Malware code analysis. Environment key generation can be used when the receiver is not aware of the conditions required to the execution of the requested service. This case corresponds to our mobile agent behavior analysis problem and is used by our approach to estimate host trustworthiness.

Trusted hardware approaches

They are used by researches to guarantee a certain behavior of a system. Herzberg and Pinter [24] describe a device that can be used to protect software against piracy. A more recent approach by Yee and Tygar [55] ensures that the system functions securely. These approaches are powerful but they have a too expensive cost.

Obfuscation techniques

Approaches using Obfuscation techniques are explored to protect the code of a mobile agent from reverse engineering for some minimum time. Obfuscation transforms a program into another program that has equivalent behavior but which is harder to understand. Fritz Hohl [25] converts the agent of origin into a black box by using obfuscation algorithms. An expiration date is attached to the black box. This approach prevents any attempt with sophisticated code analysis and any replay. It allows complex functions but guarantees protection only for a certain time interval. Barak et al. [2] studied the theoretical limits of obfuscation techniques and showed that in general achieving completely secure obfuscation is impossible. Larry D'Anna [12] states that obfuscation can prevent a malicious host from observing or predictably tampering with code and data in the running system, however, it cannot prevent the program from being reverse engineered. Beaucamps et al. have also studied how to practically obfuscate executable code. They define τ-obfuscators whose role is to hide non trivial results at least for time τ [3]. This practical obfuscation technique has been essentially considered in the case of malware context and under some very particular conditions. We use obfuscation techniques in our approach to obtain various versions of the same task and thus to provide the mobile agent several equivalent behaviors that generate the same result (see Sect. 3.3).

The k-out-of-n threshold scheme approach

They are also approaches based on the subdivision of the transmitted secrecy in several small secrecies. It is the case in the k-out-of-n threshold scheme approach [4] the secrecy of the transaction is distributed between n duplicated mobile agents. The latter are emitted towards different hosts. The confidentiality is dealt with since no mobile agent knows the totality of information. This approach does not take into account the integrity but it is an example of the "secure distributed computing" concept: parts can jointly calculate the result of a particular function without revealing the inputs. Filiol [17] uses a similar approach to protect antivirus from analysis by an attacker. He describes a combinatorial, probabilistic malware pattern scanning scheme that limits black-box analysis. This protection can only be bypassed in the case where there is collusion between a number of attackers. This idea appears also in the code-on-demand or the co-operating agent techniques. The Code-On-Demand approach [51] protects the mobile agent integrity by using dynamically upgradeable agent code, in which new agent function modules can be added and redundant ones can be deleted at runtime. This approach enhances code privacy, reduces transport cost and helps the recoverability of agents after malicious attacks; while the co-operating agent technique [26,42,43] distributes critical tasks of a single mobile agent between co-operating agents. This technique reduces the possibility of the shared data being pilfered by a single host. This idea is employed by our approach where the service (secret) is subdivided in several tasks and an abstract expression shows the selected execution.

2.2 Trust estimation

Trust plays an important role in e-commerce and e-business applications. It is a key to their acceptance and their general deployment.

The concept of trust has been a subject of large interest in different research areas like economics, game theory and multi-agent systems [8,13]. Obtaining and maintaining trust estimation is a serious open problem. Emphasis in the literature is mostly on techniques for preventing malicious agents from harming their execution environment. Many general trust models have been proposed to introduce the trust notion in the context of general distributed system applications [1,18,32]. However only a small number of these models have addressed the issues of integrating trust with security in mobile agent based systems [19].

Beth [54] proposed one of the earliest trust models for authentication in distributed systems focusing on relationship modeling while Abdul-Rahman et al. [1] provided a general model based on recommendations. But these models did not address the trust dynamics based on behavior.

Wilhelm et al. [52,53] give one of the more comprehensive discussions on the issue of trust in mobile systems. They identify what they referred as the four foundations of trust, namely: blind trust, trust based on (a good) reputation, trust

based on control and punishment and trust based on policy enforcement. Their solution to the trust in mobile agent systems problem is the CryPO protocol, based on tamper-proof hardware to provide tamper-proof environments, which are the foundation for the agent executor. Agents assert which environment manufacturers they trust. The protocol uses certificates and encryption technology to ensure security and is essentially an extension of the certification framework. This solution provides an easier way for a new service provider to establish itself in the market; it also allows an agent owner to protect specific data in a mobile agent. However, the used trust notion is static and is mapped to a particular manufacturer of the hardware; hence this approach does not allow dynamic trust update, and then limits the flexibility in application.

Manchala [35,36] develops a model based on trust-related variables such as the cost of the transaction and its history, and defines risk-trust decision matrices. The latter are used together with fuzzy logic inference rules to determine whether or not to transact with a particular party.

Tan et al. [49] propose a trust model specifically for mobile agent security using Trusted Third Parties (TTPs) in the form of verification servers, but they do not address how trust can be integrated with security systems

Braynov et al. [8] give a solution that does not rely on collecting and analyzing information about untrustworthy agents. Instead, they propose an incentive-compatible mechanism in which agents truthfully reveal their trustworthiness at the beginning of every interaction. In this mechanism, agents report their true level of trustworthiness, even if they are untrustworthy.

Cahill et al. propose the SECURE project [9] which investigates the design of security mechanisms for pervasive computing based on the human notion of trust. The central contribution of SECURE is to provide entities with a basis for reasoning about trust and risk embodied in a computational framework that can be adapted to a variety of application scenarios. But, it is not clear how they develop the dynamical adaptation of trust.

Dimitrakos [13] introduces metrics, costs and utility functions as parameters of an algorithm that produces the trust policy for a given trusting decision. Nevertheless, this work lacks a quantitative definition of the various involved measures.

Josang [28,29] proposes a scheme for propagating trust through computer networks based on public key certificates and trust relationships, and demonstrates how the resulting measures of trust can be used for making decisions about electronic transactions. He also defines a model of trust composed of a reliability trust as the probability of transaction success and a decision trust derived from the decision surface. Trust adaptability with time has not been considered in the model.

Chin Lin et al. [32–34,37] suggest a hybrid trust model employing soft trust mechanisms with constructs such as recommendation, direct experiences via interactions and observations. These mechanisms are used to complement hard trust (based on cryptographic mechanisms) for enhancing the mobile agent security in situations where full authentication trust is not available due to absence or unavailability of trusted third parties.

Castelfranchi et al. [10] claim the importance of a cognitive view of Trust. They argue in favor of a cognitive view of trust as a complex structure of beliefs and goals, implying that the trustor must have a "theory of the mind" of the trustee. Such a structure of beliefs determines a degree of trust and an estimation of risk, and then a decision to rely or not on the other. The decision is also based on a personal threshold of risk acceptance/avoidance. They explain rational and irrational components and uses of trust. Their work represents a support of our approach where trust value is based on trust ingredients (evaluation of the situation and the behavior).

These trust models generate a subjective single value which does not reflect the exact cause of the lack of trust. Thus, the provided decision could be inadequate. Some of them use also the reputation of the host as factor intervening in the trust estimation. It is not necessary for the mobile agent to know the reputation of the visited hosts that can be new in the network. In our approach, the trust estimation is dynamic and it is used to enable the mobile agent to adapt its execution in untrusted environments.

Our approach exploits opportunities offered by the dynamic adaptability mechanism [21,31,39] to protect the mobile agent against the visited host. The dynamic adaptability mechanism is used to offer the mobile agent the possibility to modify its behavior. This ability makes it unpredictable and complicates its analysis. The idea is that the mobile agent must verify the customer trustworthiness and present to him, according to the trust he inspires, an appropriate behavior.

3 The TAMAP principles

TAMAP is a Trust-based Approach for Mobile Agent Protection. It is founded on three main principles: a protection protocol, a trust estimation mechanism and a dynamic adaptation process.

The protection protocol aims to use reliable mechanisms for the protection of the mobile agent code (e.g., encryption, hashing, digital signature, etc.). It also protects the service provider and the mobile agent communication.

The trust estimation mechanism enables to establish host trustworthiness and thus the selection of the most appropriate Quality of Service (QoS). This mechanism relies on the customer data and their comparison with those held by the service provider. This assessment enables the estimation

of the trust degree and thus the selection of suitable QoS (see Sect. 4). Furthermore, collected data are used for the environment key computation. The environment key is computed by the mobile agent on the side of the customer as well as on the service provider side. It is employed as a symmetrical key to encrypt and to decrypt the emitted QoS. Consequently, the trust estimation mechanism is based on the generated environment key and it is identified by the following steps:

1. The trust parameters inventory (the establishment of a set of parameters whose values must be checked).
2. The trust parameters evaluation.
3. The trust quantification.
4. The QoS selection.

The dynamic adaptation process uses the provided QoS and the mobile agent history to choose the most adequate execution (see Sect. 5). This selection is based on the ability of the mobile agent to vary its behavior. The adaptation is statically designed and dynamically performed. It is supported by a reflexive mobile agent architecture.

The proposed approach is based on some assumptions:

- A contract is built beforehand between the customer and the service provider.
- The service provider knows some confidential information concerning the customer (e.g., contract reference or password).
- The host knows something about itself or about the environment that the agent does not know.
- The information that the customer knows has an incidence on the agent owner decision to execute or not the requested service.
- The mobile agent has to calculate the environment key without knowing whether the host private information is right or not.
- The service provider can already have an idea on the former trustworthiness of the host.

In this section, we first introduce an example that will be used to illustrate the proposed approach. We then describe the protection protocol, present the mobile agent behavior model and specify the proprieties of the provided service. Finally, we discuss the features of a secure environment, underline the trust and security relationship and give a definition of trust.

3.1 A simple example

The context of this work is given by the scenario where a service provider uses a mobile agent to perform a service. Arrived at the customer host, the mobile agent tries to detect the various conditions which make it trustful and which must be taken into account for the execution of the provided service. The mobile agent generates an environment key. It then uses this key for the trust estimation and thus the selection of the most suitable behavior.

To illustrate our approach, we consider the example where an online provider offers some services relying on some kinds of heuristic viruses scanners. The mobile agent tries to protect its behavior to prevent its analysis. Its protection is mainly based on its capacity of adaptation which enables it to generate, for the same input, various behaviors.

The service provides different QoS. It is appropriate for the reaction of the mobile agent after the trust estimation step. Examples of QoS are:

QoS0: Do not perform the service.
QoS1: Do not perform the service and notify the host.
QoS2: Scan the computer and display the result.
QoS3: Scan the computer, display the result and set up an antivirus of a lower performance.
QoS4: Scan the computer, display the result and set up an antivirus of a medium performance.
QoS5: Scan the computer, display the result and set up an antivirus of a higher performance.
QoS6: Scan the computer, display the result and set up an antivirus of lower performance and enable its update.
QoS7: Scan the computer, display the result and set up an antivirus of medium or higher performance and enable its update.
QoS8: Scan the computer, display the result and set up an antivirus of higher performance and enable its update.

3.2 Protection protocol

This protocol aims to protect a mobile agent code against malicious hosts as well as the service provider and the mobile agent communication. Its primarily objective is the generation of the environment key. The latter has an important role since it defines the trust degree of the target host. It is also used, at the customer side, to decrypt the received QoS. This key is calculated using information collected at the target host and the mobile agent identifier, which is unique. It is based on a one way hash function coupled with a public key encryption scheme. In order to protect the environment key and to avoid transmitting it, the environment key is calculated on the customer side and on the service provider side.

When the customer needs a service, it sends a request to all the service providers. These providers send their proposals. Those are then analyzed by the customer to select the best one. The customer informs the corresponding service provider which generates private and public keys, and assigns an adequate QoS to each behavior of the mobile agent. The mobile agent moves then to the customer host.

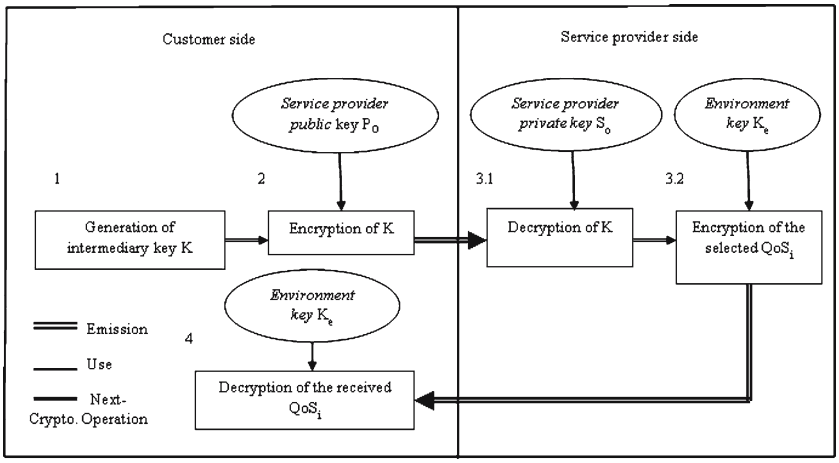

Fig. 1 Mobile agent protection protocol

We propose to use simultaneously the asymmetric and the symmetric cryptographic methods and assume the existence of the following couples of public and private keys:

– Agent owner (or service provider) keys: (P_o, S_o)
– Host keys: (P_h, S_h)

The main steps of this protocol are (see Fig. 1):

1. The mobile agent starts interacting with the host in order to obtain the needed information to generate the environment key (see Box 1).
2. The mobile agent encrypts the intermediary key K with the service provider public key P_o, signs it with the host private key S_h and sends it with the signature to the service provider (see Box2). Then, it calculates the environment key.
3. The service provider verifies the signature and deciphers the received key (see Box 3.1). It analyses the key, computes the trust value (see Sect. 4.3), selects the corresponding QoS, encrypts it with the environment key (computed at the service provider side), signs it and sends it with the signature to the costumer (see Box 3.2).
4. At the customer side, the mobile agent tries to decipher the QoS with the environment key (see Box 4). In case of success, it executes the requested service.

Further details are given in Sect. 4.

3.3 Mobile agent behavior model

The behavior of the mobile agent is related on the provided service. The service proposed by the mobile agent must be able to be deployed on various environments by taking into account their level of security. Consequently, the objective is of being able to adapt the QoS offered by the application according to the variations of the security levels of the host. These variations depend on confidential information held by the visited host as well as availability of resources affecting directly the fulfillment of the selected service.

The QoS selection is based on a preliminary phase represented by the estimation of the trustworthiness of the visited host. Thus, the mobile agent behavior depends on the selected category of QoS. It can:

1. Perform the requested service and leave the host (if the host trustworthiness is high).
2. Perform a degraded service and leave the host (replace the requested service by another which is less useful because the trust value is not significant).
3. Leave the host without performing the service (if the host trustworthiness is too low).

Our aim is to make complex the dynamic analysis of the mobile agent behavior. In order to thwart dynamic analysis, the mobile agent designer must arrange the code that it will take very different execution paths on different runs. Consequently, determining which basic blocks were always invoked, and in which combinations, would require a great effort.

Based on the service decomposition (the service is decomposed into several tasks including dummy ones) and obfuscation techniques [22,23], each QoS category can be performed in different ways. This propriety provides the mobile agent with the ability to have different reactions for the same QoS. Therefore, this aptitude complicates its analysis since its behavior is unexpected.

Let $E = \{E_1, E_2, \ldots, E_n\}$ be a set of n abstract expressions that are used to implement the different behaviors of the mobile agent.

Let $A = \{A_1, A_2, \ldots, A_p\}$ be a set of p functional tasks (including dummy ones) that are included in the different executions. Each E_j $(j < n)$ is a calls sequence of a subset

of A and can be viewed as a sequence of p bits (each bit indicates if a specific task is included or not).

If we consider p tasks, we can have 2^p-2 possible combinations; We do not consider the case where no task is executed. An abstract expression is associated to each combination and shows a specific behavior. This propriety increases the number of mobile agent possible executions and thus complicates its behavior analysis. The mobile agent executions vary from a service to another and are related to the service degradation degree and to the provided alternatives.

3.4 A secure environment

When an executor receives a mobile agent, the owner loses all control over the agent's code and data. The executor can reverse-engineer the code, analyze the data, or modify either one. If no direct attack is feasible, the executor can still experiment with the agent by supplying it with arbitrary data to observe its reactions and by resetting it to its initial state. The executor could even do this with a copy of the agent on an isolated platform. Thus, the owner has to trust the executor in order to prevent its illegal use of these methods. For this reason, the evaluation of the executor trustworthiness presents a primordial stage in our approach.

A mobile agent trusts a host when it believes that it is secure. Thus, it is important for mobile agents to be able to identify their hosts in order to trust them (or to have an opinion on their trustworthiness).

Control is required to establish trust and to enhance security. Thus, if a mobile agent does not trust the host, it uses, for example, observation to prevent or at least detect its misbehavior. Moreover, if the host knows that it is observed, it is more reliable.

Different definitions of trust have been proposed. The commonly used in literature is from Gambetta [18] "*…trust, (or symmetrically, distrust) is a particular level of the subjective probability with which an agent will perform a particular action, both before he can monitor such action (or independently of his capacity to monitor it) and in a context in which it affects his own action*".

Castelfranchi et al. in [10] claim that the richness of the mental ingredients of trust cannot and should not be compressed simply in the subjective probability estimated by the actor for its decision. The question therefore is: why does one need an explicit account of the mental ingredients of trust?

We agree with Castelfranchi and state that host trustworthiness relies on several parameters and malicious behaviors. We use the definition proposed by Josang [27]: "*trust is the extent to which one party is willing to depend on somebody, or something, in a given situation with a feeling of relative security, even though negative consequences are possible*". This definition is suitable for dynamic environment. In the current context, the trust definition requires to answer the following questions:

- How can the agent perceive its environment so that it can emit a right opinion on the trust and the profile of this host?
- How can various perceptions be aggregated to generate the environment key?
- How can this key determine exactly the category of customer?
- How can this key, in case of misbehavior, explain the origin of the failure to avoid a reaction that would not be adapted?

Section 4 describes the steps followed by the trust estimation mechanism.

4 Trust estimation

The trust estimation requires the inventory of trust parameters, their evaluation and the trust quantification. It enables a selection of the action to be undertaken and which is presented, in our case, by the selected QoS.

The trust estimation remains a quite subjective task. It is a result of a monitoring of (1) the target host by observing it, inspecting it and/or questioning it and (2) the mobile agent reaction by performing the service, reducing it or do not achieve it.

To evaluate trust, each mobile agent uses (see Fig. 2):

- Observation mechanism that captures all parameters apt to contribute to the perception of the environment and to inform about the host's trust degree,
- Interaction mechanism that enables the mobile agent to ask the host questions for which the agent owner already knows the answer. In this case, trust could be based on private information

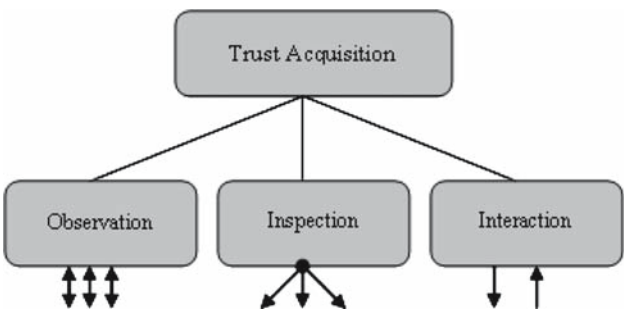

Fig. 2 Trust acquisition mechanisms

– Inspection mechanism that allows examining the environment in the search of particular information.

There are various techniques for observation and inspection. We refer, for example, to Rutkowska's Red Pill method [45] and more generally to Virtualization-based rootkit[1] techniques and the relevant detection issues (http://en.wikipedia.org/wiki/Rootkit; http://3psilon.info/Les-rootkits.html).

The concept of trust is based on the parameters that inform on the host trustworthiness. Based on the values of these parameters, the trust is quantified.

This section shows the characteristics of parameters that contribute to estimate trust, presents algorithms that define the environment key generation as well as the trust quantification and the selection of the most suitable QoS.

4.1 Trust parameters

The quantitative dimensions of trust are based on the ones of its cognitive constituents [10]. The trust estimation is based on the analysis of a set of parameters values which represent a prerequisite to the required service execution. We assume that the values of all these parameters could produce an environment key. For a good protection of the mobile agent, the environment key has to inform about the host trustworthiness. Its analysis must highlight the causes of the obtained result (trust value); particularly, when the trust degree is low. So, the key should not be a simple value but an aggregation of a set of significant values.

Thus, to determine the host trustworthiness, the following parameters must be identified:

– Parameters that make an interaction trustworthy.
– Parameters that determine a level of trust of every costumer.
– Parameters that determine a category of costumers to which the host belongs.
– Software and hardware parameters that may affect perception of trust and service fulfillment.
– Reputation (may not exist) of a visited host provided by the agent's owner or a third party and witch is related to the history of the host's transactions.

We note that the kinds of parameters in, for example, the second and fourth points are different. The second point expresses all parameters that are related to the privacy of the host (e.g., password) while the fourth point shows parameters that are associated to the host infrastructure and that are not private. Thus, we distinguish between internal and external trust. Internal trust specifies internal characteristics of the trustee while external trust designates technical or environmental obstacles against the fulfillment of the service. We subdivide parameters into two sets:

– Internal parameters which define internal trust. They include all information apt to authenticate the relation existing between the customer and the service provider as identity (name, organization, etc.), contracts (type, reference, service acronym, validity date, etc.), certificate, private information (password, etc.), etc.
– External parameters which define external trust (circumstances, reputation, situation, environment, infrastructure, etc.).

The composition of internal/external parameters produces different trust estimations. When opting for the best reaction it will adopt, the service provider must distinguish between the two kinds of parameters (if the host is failing or diffident this does not mean automatically that it is malicious). The analysis of the generated environment key enables to make a meticulous diagnostic of lack of trust.

Each parameter j has two attributes: its weight W_j and its importance I_j. The weight expresses the parameter power while the importance defines its impact in the final decision; especially when the trust value is low or not significant (suspicion).

The trust parameters inventory and the definition of the parameters attribute values are realized by the mobile agent designer. They are related to the service security criteria.

4.2 Environment key generation

The present subsection emphasizes the different steps followed by both the service provider and the mobile agent to perform a requested service. It mainly gives details on the environment key generation which represents the support of the trust estimation mechanism (see Fig. 1). The environment key generation task is based on the data collected during the interaction between the mobile agent and the host.

As soon as the mobile agent arrives at the customer host, it executes some actions which enable trust acquisition (see Table 1).

We use, in the following algorithms, as capital letters the operation that is performed ($\mathcal{H}, \mathcal{E}, \mathcal{D}, \mathcal{S}$ and $\mathcal{V}$ for respectively the hashing, the encryption, the decryption, the signature and the verification), and as subscripts the name of the entity which perform the operation ("a" for mobile agent and "h" for service provider host).

In order to evaluate the customer trust degree, the service provider must execute some actions (see Table 2). These actions enable the customer trustworthiness estimation and thus, the selection of the suitable service.

[1] A rootkit is a tool that is designed to hide itself and other processes, data, and/or activity on a system.

Table 1 The mobile agent execution Algorithm

Input: Agent Owner public key P_o, current Host keys (P_h, S_h), collected data, EQ and SQ
Output: EM and ES

1: Collect data (correspond to parameters values); let {d_1, d_2... d_k} be the set of collected data.
2: Apply a SHS (Secure Hash Standard) one way function to each data
For i: = 1 to k do M_i: = $\mathcal{H}(d_i)$ End For.
3: Concatenate all digests and obtain the intermediary key M = ($M_1.M_2...M_k$).
4: Encrypt M using the agent owner public key P_o: EM= $\mathcal{E}_a$ (M).
5: Sign EM using the host private key S_h and obtain: SM= $\mathcal{S}_a$ (EM).
6: Send both EM and SM to the service provider.
7: Apply hashing to the result of the third step, let D= $\mathcal{H}$(M) be the final digest.
8: Apply D $\oplus$ id (where id is a unique mobile agent identifier) to generate an environment key K_e. K_e will be used to decrypt the selected QoS_i.
9: Receive EQ with its signature SQ.
10: Verify SQ using the Boolean operation $\mathcal{V}$(EQ, SQ):
10.1. Decrypt SQ using the service provider public key P_o and obtain the Received Digest: RD= $\mathcal{D}_a$ (SQ).
10.2. Apply hashing to the received EQ and obtain the Computed Digest: CD= $\mathcal{H}$(EQ)
10.3. Compare RD with CD: $\mathcal{V} \leftarrow$ (RD = CD)
11: Try to decrypt the received EQ using K_e: $\mathcal{D}_a$ (EQ) = QoS_i.
12: If the decryption is successful then Execute the selected service.

A more detailed description of the main steps of these algorithms is given in the appendix.

4.3 Trust quantification

The host trust estimation T is relative to parameters attributes values and is calculated according to importance I_J of the parameter J, of its weight W_J and the factor S_J. S_J has a binary value. It corresponds to the result of the validation test of the collected parameter j value.

$$T = \sum_{i=1}^{k} W_j I_j S_j. \quad (1)$$

The service provider has a database which includes for each customer its own parameters values with their corresponding hashing (see Table 3). A matching of the received and existent hashing data is performed. It enables the attribution of the value 1 or 0 to the factor S_j of each parameter j respectively in cases of matching or not matching. Then, the trust T is estimated using the previously mentioned formula (1).

The selection of the suitable QoS relies on the value of T. The variations of this value are described by a granulation of the interval [0, 100]. Table 4 gives an example of such granulation. Our trust estimation is based thus on intervals of values and not on thresholds. This provides a greater flexibility of reaction. Indeed, rather than having rigid reactions as it is the case in the trust models using thresholds, the reaction here is adapted according to values belonging to a particular interval. For example, if the trust value belongs to the good interval (e.g., [71–100]) the customer is trusted. Three cases occur:

1. The trust value tends to 71, QoS6 is selected
2. The trust value tends to 100, QoS8 is selected or
3. The trust value tends to middle of this interval, QoS7 is selected.

The mobile agent can, after receiving the appropriate QoSi, perform the selected service. On the other hand, if the obtained trust value is considered to be low (e.g., [0–30]), it is possible before selecting the adequate QoS, to look for the exact cause of this failure. This is realized by seeking among the parameters which had a factor S equal to zero.

The parameters are subdivided into sets (see Table 5):

1. Parameters with higher importance (3)

Table 2 The service provider execution Algorithm

Input: Agent Owner public keys (P_o, S_o), Host public key P_h, EM and ES
Output: EQ and SQ

1: Receive EM and SM.
2: Verify SM using the Boolean operation $\mathcal{V}$(EM, SM):
  2.1. Decrypt SM using the host public key P_h and obtain the Received Digest:
    RD = $\mathcal{D}_h$(SM).
  2.2. Apply hashing to the received EM and obtain the Computed Digest:
    CD= $\mathcal{H}$(EM)
  2.3. Compare RD with CD: $\mathcal{V} \leftarrow$ (RD = CD)
3: Decrypt EM using its private key S_o and obtain: M = $\mathcal{D}_h$(EM).
4: Obtain k digests $\mathcal{H}(d_1)$, $\mathcal{H}(d_2)$... $\mathcal{H}(d_k)$ from the obtained M (based on the parameters number and the size of each digest).
5: Check the obtained digests with the digests present in the database (see Table 3).
6: Estimate the host trustworthiness by calculating the value of T (see Subsection 4.3).
7: Compare the trust values with the intervals values and select the action to be undertaken (see Table 4).
8: Select QoS_i.
9: Apply hashing to the result of the fourth step, let D= $\mathcal{H}(\mathcal{H}(d_1).\mathcal{H}(d_2)...\ \mathcal{H}(d_k))$ be the final digest.
10: Apply D $\oplus$ id (where id is the identifier of the emitted mobile agent) to generate a key K_e.
11: Encrypt the selected QoS_i using K_e: EQ=$\mathcal{E}_h$ (QoS_i)
12: Sign the EQ using its private key S_o: SQ=$\mathcal{S}_h$ (EQ).
13: Emit EQ and the signature SQ to the customer.

Table 3 Example of customer records

Parameter	Parameter value	Digest (SHA-1)
Service identifier	Mobi-Scan	6654b57274e1bdfc2953324999cbaef0bb470be9
Acronym	Esprit 7	bca3fbec4d267cacddeaa0dcbc091f72d3f32fd7
Validity date	December 31th, 2007	de01cecfc4fc72d91128210329614afab1ff09ce

Table 4 Example of estimation intervals with their related feedback

Interval of trust estimation	Feedback
	QoS0
0–30	QoS1
	QoS2
	QoS3
31–70	QoS4
	QoS5
	QoS6
71–100	QoS7
	QoS8

Table 5 Example of values of parameters attributes

Parameter	Weight	Importance value (3, 2, 1)	Reaction in case of failure (relied on the importance of the parameter)
Identifier	20	3	QoS0
Acronym	10	2	QoS3
e-mail	5	1	QoS6

2. Parameters with medium importance (2)
3. Parameters with lower importance (1)

Given the importance of some parameters, they can deeply influence the choice of the action to be undertaken and thus the selection of the adequate QoS. The values assigned to the attributes (the weight and the importance) of each parameter define its impact on the final decision (see Table 5).

Note that there is only one case which will generate the execution of the complete service. It is the one where the attacker knows all customer data (collected by interaction). Furthermore, the data collected by inspection are also valid.

We suppose too that this attacker has intercepted the customer private key. This case seems very improbable.

5 Mobile agent architecture

In the proposed approach, the adaptation is statically designed and dynamically performed. This solution consists in estimating, during the construction of the application, the different variations of the environment and defining the adaptation rules. Among the different techniques that enable the realization of the adaptation, the reflexivity seems to be a promising solution. It constitutes a support of the application development and provides mechanisms enabling to express treatments in extremely generic terms. The intrinsic properties of the reflexivity (introspection and intercession) give the mobile agent the possibility to reason and to act on itself. The proposed reflexive architecture highlights two conceptual levels:

A basic level

It contains the functional code and formulates the application logical implementation as well as its semantics. It comprises five components: an *Interface*, a *Memory*, a *Library*, a *key generator* and a *Manager*. The interface allows communication with the host and permits data acquisition. These data are used for the mobile agent behavior selection. Collected data and execution trace are stored in a memory and the tasks used by the various behaviors are in the library.

A meta-level

It contains the non-functional code and describes the adaptation process.The latter is ensured by an *Adaptor* component.

5.1 Architecture components

The proposed mobile agent reflexive architecture (see Fig. 3) contains the following components:

Interface

It manages the communication between the mobile agent and its current environment. It comprises two sub-components:

A sensor: It is responsible for the environment perception and the data acquisition. These data are useful for the host trustworthiness estimation. Three types of acquisition can be considered: the observation, the inspection and the interaction. The type of acquisition is related to required data. For example, the interaction mechanism is used for the acquisition of the identity, the inspection mechanism is used to research of a specific file and the observation mechanism controls the number of times the customer tries to write a password.

An actuator: it allows the execution of the sequence of actions related to the selected behavior. These actions can be a simple alarm, a modification task, a transmission/reception message, a stop of the execution or/and a migration.

Library

It contains the tasks of the mobile agent code. The compositions of these tasks generate the different agent behaviors. The purpose is to offer more confidentiality and more flexibility to the behavior. In order to vary the mobile agent behavior, alternatives for some tasks are proposed. These alternatives are mainly provided by the obfuscation algorithms. Furthermore, some dummy tasks are used to complicate the mobile agent behavior analysis and to increase the security level. Various combinations of the tasks are supplied by a set of abstract expressions used to implement the mobile agent behaviors. The library comprises also all the security tasks used by the protection protocol.

Let $A = \{A_1, A_2 \ldots A_q\}$ be a set of tasks. The set A can be subdivided on two subsets: F and S. $F = \{A_1, A_2, \ldots, A_p\}$ is a set of functional tasks that corresponds to the application and $S = \{A_p + 1, \ldots, A_q\}$ is a set of security tasks (e.g., cryptography, hashing, digital signature, etc.).

Memory

It contains the hashed collected data and the execution trace. Both are signed by the host private key. They will be used, by the service provider, to check the validity of collected data and to confirm the execution of the selected behavior.

Key generator

Its role is the generation of the environment key. It allows the safeguard of hashed data into the memory and their emission to the service provider using the actuator component. It is also used to decrypt the received QoS.

Manager

In addition of its management role, the manager determines the actions to be performed by selecting the tasks stored in the library. In order to increase the complexity of the mobile agent behavior analysis, each selected task is related to various abstract expressions. An abstract expression is defined as a combination of useful and dummy tasks. For instance, if the selected rule is R8 and (T0, T1) are dummy tasks the combination could be (T0, task2.3), (T1, task2.3),

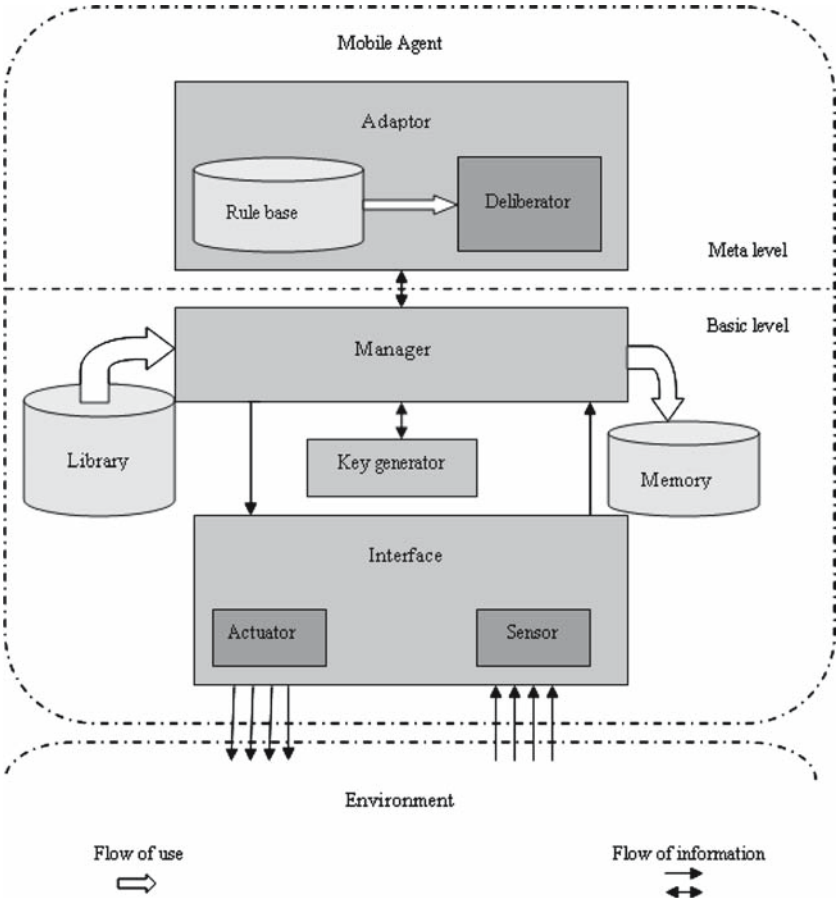

Fig. 3 The mobile agent reflexive architecture

(task2.3, T0, T1), (T0, task2.3, T1) or (T1, task2.3, T0). Moreover, the manager allows the use of the security tasks.

Adaptor

It applies the adaptive policy. It comprises a rules base and a deliberator.

The rules base: It contains the execution rules. These rules are described as follows: If <Condition> Then <Action>. The left side of the rule comprises the selected QoS and its right side specifies the corresponding task. The rules base includes the behavioral rules (see Table 6) and rules of exceptions like if <Time of execution expired> then <notify and leave>.

The deliberator: It is an execution engine. It uses the received QoS and the rule base to choose the set of rules that can be triggered. It then selects the most appropriate rules among the selected ones. This selection determines the mobile agent behavior. It uses the weight of the rules. The weight is initialized to zero and is then incremented each time the rule is triggered along the way of the mobile agent. A random number generator selects the suitable rules among the rules which have the lower weight. The behavior history of the mobile agent is related to the rules weights.

Table 6 Example of behavioral rules at the initialization stage

Weight	Rule
0	R0: if QoS0 then exec (task0.1)
0	R1: if QoS0 then exec (task0.2)
0	R2: if QoS1 then exec (task1.1)
0	R3: if QoS1 then exec (task1.2)
0	R4: if QoS1 then exec (task1.3)
0	R5: if QoS1 then exec (task1.4)
0	R6: if QoS2 then exec (task2.1)
0	R7: if QoS2 then exec (task2.2)
0	R8: if QoS2 then exec (task2.3)
0	R9: if QoS3 then exec (task3.1)
…	…
0	…

5.2 An overview of a mobile agent execution

As soon as the mobile agent arrives at the host, it starts the environment key generation operation. During this step, the

mobile agent collects data using the sensor component. The key generator component applies the hashing, encryption and digital signature operations to the collected data using the corresponding tasks present in the library via the manager component. It emits the result to the service provider and saves a copy in the mobile agent memory. Then, it calculates the environment key. After that, the mobile agent awaits the arrival of the selected QoS from the service provider. As soon as the message carrying the selected QoS arrives, the mobile agent identifies it (checks the digital signature) and uses the environment key to decrypt it. Then, the deliberator utilizes the selected QoS and the rule base to select the set of rules that can be triggered. Based on the random number generator, the deliberator selects the suitable rules among the rules which have the lower weight. Finally, The manager selects the chosen tasks from the library and the actuator allows the execution of the sequence of actions related to the selected behavior.The latter executes actions and delivers a report of the execution to the manager. This report is saved in the mobile agent memory.

With an aim of increasing the flexibility of the system, a set of exceptions is defined. If any problem occurs during one of the execution steps, the manager notes the non-progression for a determined duration. Then, it asks the deliberator to check the set of the exceptions for a possible re-establishment. If the whole of the exceptions does not correspond to the problem mentioned, it stops the execution immediately, notifies the problem and leaves the current host to migrate to the next host or to the host of origin.

6 Some implementation aspects

We built a prototype to demonstrate the possibility of TAMAP mobile agents. Java (JDK 1.5) was used as the language for the mobile agent development due to the fact that it provides cross-platform portability.

Java enables also dynamic extension possibility through the usage of Class-Loader. This means that Java can dynamically load and use classes at runtime [20,51]. The Java program does not need to know all classes at compile time. This ability is very useful for the dynamical load of the library tasks. Furthermore, Java has a rich library that comprises several security functions.

We present in this subsection the adaptor which is the most important part of the mobile agent code. The adaptor contains the deliberator and the rule base (see Fig. 3). As shown previously the deliberator is an execution engine rather than an inference engine. This choice is based on the fact that the execution engine offers a faster execution and the weight of the mobile agent is more reduced. The deliberator is represented by the AgRunEngine class and the rule base corresponds to the AgRuleBase class (see Fig. 4).

The most important parts of the execution engine code are:

– The selection of rules that match the requested Qos:

```
for(int i=0; i<ruleBaseEng.getRuleBaseSize();i++){
if(ruleBaseEng.getRule(i).getQosValue()==qosValue){
matchedQosValueList.add(new
AgMatchedQos(ruleBaseEng.getRule(i).getWeight(),i)); } }
```

– The search of the minimum rule weight:

```
minWeight=((AgMatchedQos)matchedQosValueList.getFirst()
).getRuleWeight(); for(int i=0;i<matchedQosValueList.size();i++)
{if(((AgMatchedQos)matchedQosValueList.get(i) ).getRuleWeight()
<minWeight)
minWeight=((AgMatchedQos)matchedQosValueList.get(i)
).getRuleWeight();
}
```

– The collection of rules that match the minimum weight:

```
for(int i=0;i<matchedQosValueList.size();i++){
if(((AgMatchedQos)matchedQosValueList.get(i)).getRuleWeight()
==minWeight)matchedWeightList.add((AgMatchedQos)
matchedQosValueList.get(i));
}
```

– The selection of the rule to be triggered

```
selectedRuleIndex = ((AgMatchedQos)matchedWeightList.get
(safeRandom(matchedWeightList.size()))).getRuleIndex();
selectedRule= ruleBaseEng.getRule(selectedRuleIndex);
```

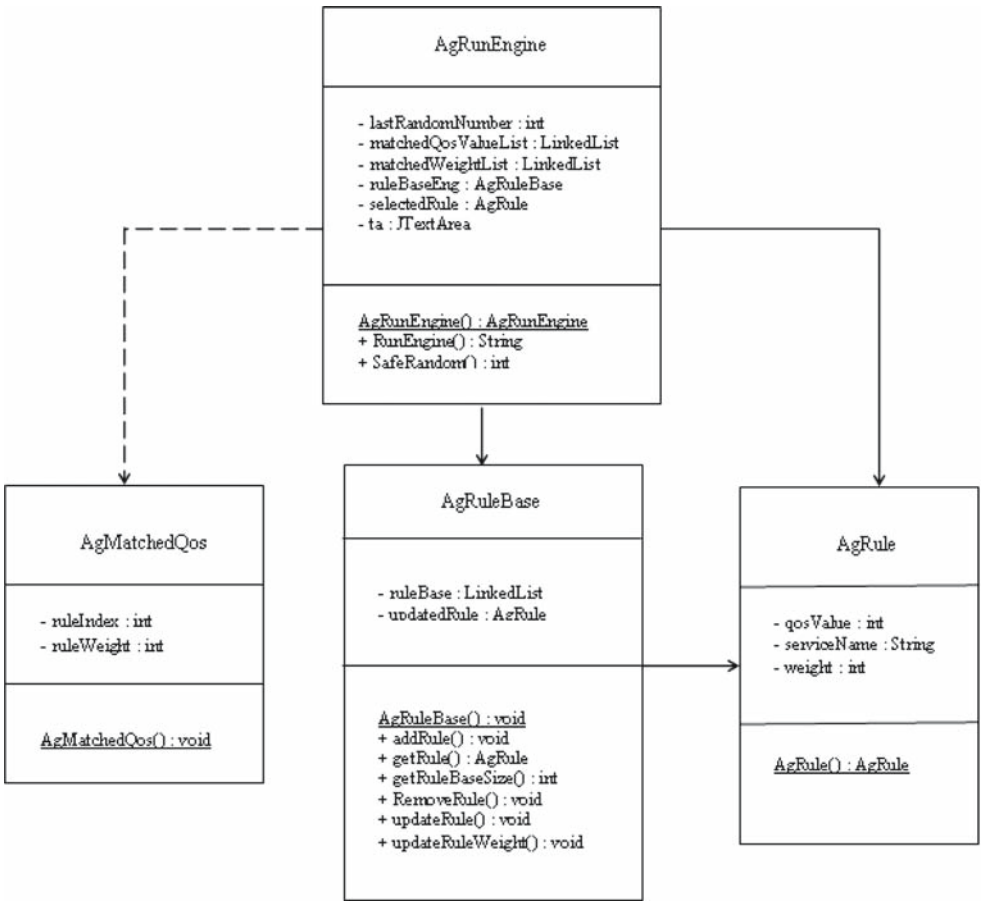

Fig. 4 The adaptor class diagram

The AgRuleBase class has several methods (see Fig. 4). An example of these methods is the updateRuleWeight(). It updates the weight of the selected rule:

```
public void updateRuleWeight(int index){ updatedRule=(AgRule)
ruleBase.get(index);
updatedRule.setWeight(updatedRule.getWeight()+1); }
```

7 Conclusion

A malicious host can deduce mobile agent execution strategy by analyzing its behavior. To prevent this kind of attacks, the proposed approach intervenes at the three levels: the protection protocol, the trust estimation mechanism and the dynamic adaptation process.

The protection protocol is primarily based on the use of the environment key. The latter is protected since it is calculated at the service provider as well as at the customer side. It is used as a symmetric key and enables the protection of the transmitted data.

We introduced a model to enable mobile agents to estimate the host trustworthiness. Moreover, we proposed a technique which provides a reaction based on realistic parameters values. This trust estimation mechanism relies on the contents of the intermediary key (obtained at the preliminary phase of the construction of the environment key). Compared to the current models of trust, our mechanism provides several advantages. The mechanism does not require the estimation of other agent's trustworthiness and it simplifies many complex and costly infrastructures for risk assessment like reputation databases. The major advantage remains in the fact that our estimation of trust is based on concrete values. Thus, in case of an untrusted host deduction, it is possible to locate the exact source of the obtained result. This ability is very important since it enables to avoid an inopportune reaction. Moreover, this model is flexible: the mobile agent owner can modify parameters, trust estimation intervals relying on the

importance of the service, its time limitation (if it is or not out of date) and the risk incurred (or the damage resulted). However, the proposed trust model requests a great aptitude in the inventory of the trust parameters as well as on the estimation of their attributes values.

The dynamic adaptation process allows the adaptation of the mobile agent behavior based on the host trustworthiness, the mobile agent behavior history and the provided QoS. It is supported by the mobile agent reflexive architecture which offers the mobile agent the propriety of flexibility allowing the modification of its behavior. Furthermore, the dynamic adaptation process is based on the degree of degradation of the service and on its ability to propose various alternatives to the same QoS.

The dynamic adaptation process provides some interests since it offers the mobile agent the ability to react with an unexpected behavior. This aptitude prevents the visited host to deduce the followed strategy. Consequently, any behavior analysis attempt becomes difficult and inefficient.

Trust constitutes a method to build host behavior-aware agent and we are now considering the fact that the agent itself could take the initiative to react after the host trust estimation. This can be realized by a dynamic behavioral adaptation. The mobile agent will be able to estimate the host trustworthiness and to modify the trust estimation intervals according to the environment security level. Thus, it will be able to decide which QoS to adopt. This would have the advantage of increasing the mobile agent autonomy in our model.

Appendix

This appendix points out the principal steps of algorithms presented in Tables 1 and 2 (see Sect. 4.2).

Customer side

1. Collect the data :
   Service identifier = Mobi-Scan
   Acronym =Esprit 7
   Validity date = December 31, 2007
2. Apply SHA-1 to each data :
   Service identifier =6654b57274e1bdfc2953324999cbaef0bb470be9
   Acronym =bca3fbec4d267cacddeaa0dcbc091f72d3f32fd7
   Validity date =de01cecfc4fc72d91128210329614afab1ff09ce
3. Concatenate the three digests:
   M=6654b57274e1bdfc2953324999cbaef0bb470be9bca3fbec4d267cacddeaa0dcbc
   091f72d3f32fd7de01cecfc4fc72d91128210329614afab1ff09ce
4. Encrypt M using the RSA (key size = 1,024) public key
   encryption scheme. The obtained result is:
   EM=69710f7d78369be7e6823c00be39174cd1eeb4c8c091ddb616380d02d163ba2
   9a046a68d430fba45bb30b751a81f697d6e254613559f0ccc555e8eea6bfa3e75bb48
   f96b1771a5ac07444497290460b22b98b97c64b61d16b5c7ec95e56c9b5d627f7b2
   53b50448d35ed1fa592cb0d5778fbd8cda2643385ecf5aec7e7f4c764
   We note that the size of EM is 256 characters.
5. Sign EM using DSA algorithm :
   SM=302c021451b1f4af3cd47c252ba00d184176fab9c7d5cfba0214010a4360324f
   91b5fcbc791a80d9ab7bff73fc8d
6. Send EM and SM to the service provider
7. Apply hashing to M :
   D = H(M) =6ec45dd290571c95b03e09037fa628fd9872a7ea
8. Apply D $\oplus$ id (where id is for example 'Ag543hdmk'):
   Ke = D $\oplus$ id =2fa368e6a33f78f8db7f6e364b954099f519e68d

The remaining steps correspond to the verification of SQ and the decryption of EQ using the environment key.

Service provider side

1. Receive EM and SM :
   EM=69710f7d78369be7e6823c00be39174cd1eeb4c8c091ddb616380d02d163ba2
   9a046a68d430fba45bb30b751a81f697d6e254613559f0ccc555e8eea6bfa3e75bb48
   f96b1771a5ac07444497290460b22b98b97c64b61d16b5c7ec95e56c9b5d627f7b2
   53b50448d35ed1fa592cb0d5778fbd8cda2643385ecf5aec7e7f4c764
   SM=302c021451b1f4af3cd47c252ba00d184176fab9c7d5cfba0214010a4360324f
   91b5fcbc791a80d9ab7bff73fc8d
2. Verify SM
3. Decrypt EM and obtain :
   M=6654b57274e1bdfc2953324999cbaef0bb470be9bca3fbec4d267cacddeaa0dcbc
   091f72d3f32fd7de01cecfc4fc72d91128210329614afab1ff09ce
4. Based on the number of trust parameters, the used hashing function (e.g., the digest size is 40 characters) and the position of each parameter, it is possible to obtain the different digests:
   Position1:
   Service identifier = 6654b57274e1bdfc2953324999cbaef0bb470be9
   Position2:
   Acronym = bca3fbec4d267cacddeaa0dcbc091f72d3f32fd7
   Position3:
   Validity date = de01cecfc4fc72d91128210329614afab1ff09ce

The remaining steps enable the estimation of trust degree, the selection of the appropriate QoS and the computation of the environment key. The latter is recomputed in the same way that on the customer side and will be used to encrypt the selected QoS.

References

1. Abdul-Rahman, A., Hailes, S.: Using recommendations for managing trust in distributed systems. In: Proceedings of IEEE Malaysia International Conference on Communication'97 (MICC'97), Kuala Lumpur, Malaysia (1997)
2. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (Im)possibility of obfuscating programs. Advances in cryptology. In: Proceedings of Crypto'2001, Lecture Notes in Computer Science, Vol. 2139, pp. 1–18 (2001)
3. Beaucamps, P., Filiol, E.: On the possibility of practically obfuscating program—towards a unified perspective of code protection. WTCV'06 Special Issue, G. Boufante, I., Marion, J.Y. (eds) J. Comput.Virol. 3(1) (2007)
4. Beimel, A., Burmester, M.: Computing functions of a shared secret. SIAM J. Discrete Math. **13**(3), 324–345 (2000)
5. Bellavista, P., Corradi, A., Frederici C., Montanari R., Tibaldi D.: Security for mobile agents: issues and challenges. In: Mahgoub, I., Ilyas, M. (eds) The Book Handbook of Mobile Computing. CRC, Boca Raton (2004)
6. Bierman, E., Cloete, E.: Classification of malicious host threats in mobile agent computing. In: Proceedings of SACICSIT2002, pp. 141–148 (2002)
7. Borselius, N.: Mobile agent security. Electron. I Commun. Eng. J. IEEE Lond. **14**(5), 211–218 (2002)
8. Braynov, S., Sandhol, T.: Trust revelation in multiagent interaction. In: Proceedings of CHI'02, Workshop on the Philosophy and Design of Socially Adept Technologies, Minneapolis (2002)
9. Cahill, V. et al.: Using trust for secure collaboration in uncertain environment. IEEE Pervasive Comput. **2**(3), 52–61 (2003)
10. Castelfranchi, C., Falcone, R.: Trust is much more than subjective probability: mental components and sources of trust. In: The 32nd Hawaii International Conference on System Sciences—Mini-Track on Software Agents, Maui, Hawaii (2000)
11. Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C., Tsudik, G.: Itinerant agents for mobile computing. Technical Report, IBM T.J. Watson Research Center, NY (1995)
12. D'Anna, L., Matt, B., Reisse, A., Van Vleck, T., Schwab, S., LeBlanc, P.: Self-protecting mobile agents obfuscation report. Network Associates Laboratories Report (2003)
13. Dimitrakos, T.: A service-oriented trust management framework. In: Falcone, R., Barber, S., Korba, L., Singh, M. (eds) Trust, reputation, and security: theories and practice, LNAI 2631. Springer, Heidelberg, pp. 53–72 (2003)
14. Farmer, W.M., Guttman, J.D., Swarup, V.: Security for mobile agents: authentication and state appraisal. In: Proceedings of the European Symposium on Research in Computer Security (ESORICS), pp. 118–130 (1996)
15. Filiol, E.: Strong cryptography armoured computer viruses forbidding code analysis: the bradley virus. In: Proceedings of the 14th EICAR Conference (2005)
16. Filiol, E.: Techniques virales avancees (chapter 8), collections IRIS. Springer, Janvier (2007)
17. Filiol, E.: Malware pattern scanning schemes secure against black-box analysis. EICAR 2006 Special Issue, Broucek I, V., Turnee, P. (eds) J. Comput. Virol. **2**(1), (2006)
18. Gambetta, D.: Can we Trust Trust? Trust: Making and Breaking Cooperative Relations. In: Gambetta, D. (ed) Basil Blackwell, Oxford (1990)

19. Grandison, T., Sloman, M.: A survey of trust in internet applications. IEEE Commun. Surv. Tutor., Fourth Quarter (2000)
20. Gong, L.: Secure java class loading. IEEE Internet Comput. 56–61 (1998)
21. Guessoum, Z., Ziane, M., Faci, N.: Monitoring and organizational-level adaptation of multi-agent systems. In: AAMAS'04. ACM, New York, pp. 514–522 (2004)
22. Hacini, S., Guessoum, Z., Boufaida, Z: Using a trust-based key to protect mobile agent code. In: Transactions on Engineering, Computing and Technology, vol. 16, ISSN 1305-5313, World Enformatika Society, CCIS'2006, Venice, Italy, pp. 326–332 (2006)
23. Hacini, S.: Using adaptability to protect mobile agents code. In: IEEE International Conference on Information Technology ITCC 2005, Las Vegas, USA, pp. 49–53 (2005)
24. Herzberg, A., Pinter, S.S.: Public Protection of Software. Advances in Cryptology: Crypto 85, pp. 158–179. Springer, Berlin (1985)
25. Hohl, F.: Time limited blackbox security: protecting mobile agents from malicious hosts. In: Vigna, G. (ed) Mobile agents and security. Lecture Notes in Computer Science, Vol. 1419, pp. 52–59. Springer, Heidelberg (1998)
26. Jansen, W., Karygiannis, T.: Mobile agent security. NIST Special Publication 800-19, National Institute of Standard and Technology (2000)
27. Josang, A.: Trust-based decision making for electronic transactions. In: The fourth Nordic Workshop on Secure ITSystems (NORDSEC'99), Stockholm University Report 99-005, Stockholm (1999)
28. Josang, A.: A Logic for Uncertain Probabilities. Int. J. Uncertain. Fuzziness Knowl.-Based Syst. **9**(3), 279–311 (2001)
29. Josang, A., Lo Presti, S.: Analyzing the relationship between risk and trust. In: Dimitrakos, T. (ed) The Proceedings of the Second International Conference on Trust Management, Oxford (2004)
30. Karnik, N.: Security in mobile agents systems. PhD thesis, Department of Computer Sciences and Engineering, University of Minnesota, Minneapolis, USA (1998)
31. De Lara, E., Wallach, D.S., Zwaenepoel, W.: Puppeteer: component based adaptation for mobile computing. In: Proceedings of the Third USENIX Symposium on Internet Technologies and Systems, pp. 159–170 (2001)
32. Lin, C., Varadharajan V.: Modelling and evaluating trust relationships in mobile agent based systems. In: Proceedings of First International Conference on Applied Cryptography and Network Security (ACNS03), Lecture Notes in Computer Science, Vol. 2846, pp. 176–190. Springer, Kunming (2003)
33. Lin, C., Varadharajan V., Wang Y., Mu Y.: On the design of a new trust model for mobile agent security. In: The 1st International Conference on Trust and Privacy in Digital Business (TrustBus04), Lecture Notes in Computer Science, Vol. 3184, pp. 60–69. Springer, Zaragoza (2004)
34. Lin, C., Varadharajan, V., Wang, Y., Pruthi, V.: Trust enhanced security for mobile agents. Manuscript (2005)
35. Manchala, D.W.: Trust metrics, models and protocols for electronic commerce transactions. In: The 18th International Conference on Distributed Computing Systems (1998)
36. Manchala, D.W.: E-commerce trust metrics and models. IEEE Internet Comput. 36–44 (2000)
37. Mu, Y., Lin, C., Varadharajan, V., Wang, Y.: On the design of a new trust model for mobile agent security, trust and privacy in digital business. Lecture Notes in Computer Science, Vol. 3184, pp. 60–69. Springer, Berlin (2004)
38. Necula, G.C., Lee, P.: Untrusted agents using proof-carring code. Lecture Notes in Computer Science, Vol. 1419, Springer, Heidelberg (1998)
39. Quin, T.: Cherubim agent based dynamic security architecture. Technical Report University of Illinois at Urbana-Champaign (1998)
40. Reiser, H.: Security requirements for management systems using mobile agents. In: Proceeding of the Fifth IEEE Symposium on Computers and Communications: ISCC 2000, Antibes, France, pp. 160–165 (2000)
41. Riordan, J., Schneier, B.: Environment key generation towards clueless agents. Lect. Notes Comput. Sci. **1419**, 15–24 (1998)
42. Roth, V.: Secure recording of itineraries through cooperating agents. In: Proceedings of the ECOOP Workshop on Distributed Object Security and fourth Workshop on Mobile Object Systems: Secure Internet Mobile Computations, INRIA, France, pp. 147–154 (1998)
43. Roth, V.: Mutual protection of cooperating agents. In: Vitek, J., Jensen, C. (eds) Secure Internet Programming: Security Issues for Mobile and Distributed Objects. Springer, Heidelberg (1999)
44. Rouvrais, S.: Utilisation d'Agents Mobiles pour la Construction de Services Distribues. These de doctorat de l'universite de Rennel, France (2002)
45. Rutkowska, J.: Red pill... or how to detect VMM using (almost) one CPU instruction, 2006. http://invisiblethings.org/papers/redpill.html
46. Sander, T., Tschudin, C.: Toward mobile cryptography. IEEE Symposium Security and Privacy, IEEE Computer Soc. Press, Los Alamitos, pp. 215–224 (1998)
47. Sander, T., Tschudin, C.: Protecting mobile agent against malicious hosts. In: Vigna, G. (ed) Mobile Agents and Security, Lecture Notes in Computer Science, Vol. 1419, pp. 44–60. Springer, Berlin (1998)
48. Smith, S.W., Austel, V.: Trusting trusted hardware: towards a formal model for programmable secure processors. In: The Third USENIX Workshop on Electronic Commerce (1998)
49. Tan, H.K., Moreau, L.: Trust relationships in a mobile agent system. In: Picco, G.P. (ed) Fifth IEEE International Conference on Mobile Agents, Lecture Notes in Computer Science, vol. 2240. Springer, Atlanta (2001)
50. Vigna, G.: Mobile Code Security. Lecture Notes in Computer Science, Vol. 1419. Springer, Berlin (1998)
51. Wang, T., Guan, S., Khoon Chan, T.: Integrity protection for Code-On-Demand Mobile Agents in E-Commerce. J. Syst. Softw. **60**, 211–221 (2000)
52. Wilhelm, U.G., Staamann, S.M., Buttyan, L.: A pessimistic approach to trust in mobile agent platforms. IEEE Internet Comput. **4**5, ISSN: 1089–7801, pp. 40–48 (2000)
53. Wilhelm, U.G., Staamann, S., Buttyan, L.: On the problem of trust in mobile agent systems. In: IEEE Symposium on Network and Distributed System Security, San Diego (1998)
54. Yahalom, R., Klein, B., Beth, T.: Trust relationships in secure systems—a distributed authentication perspective. In: The Proceedings of IEEE Conference on Research in Security and Privacy (1993)
55. Yee, B., Tygar, D.: Secure coprocessors in electronic commerce applications. In: The Proceeding of First Usenix Workshop on Electronic Commerce, Usenix Assoc., Berkeley, pp. 155–170 (1995)