# Evaluating Trust Among Agents

GIACOMO CABRI, LUCA FERRARI, LETIZIA LEONARDI
Dipartimento di Ingegneria dell'Informazione – Università di Modena e Reggio Emilia
Via Vignolese, 905 – 41100 Modena – ITALY
Phone: +39-059-2056190 – Fax: +39-059-2056126
E-mail: {cabri.giacomo, ferrari.luca, leonardi.letizia}@unimo.it

***Abstract –*** *Agent-based applications are more and more exploited in the development of distributed systems, with particular regard to the Internet ones. Even if the development of agent-based applications is not so difficult today – thanks to new paradigms and techniques – security problems are still present. In particular, it is important to deal with security of the data exchanged between agents at runtime. In fact, agents are social, and they interact with other agents in order to carry out specific tasks. Since interacting agents could be developed by different programmers, or provided by different third parties, there is the risk that the interacting counterpart could act maliciously with the received data. In this paper we propose an approach based on the concept of trust, which is more dynamic and adaptable than security, in order to evaluate if an interaction can be done or not.*

***Keywords****: Agents, Roles, Interactions, Trust*

## 1. Introduction

In agent-based applications, interactions among agents are largely exploited in order to use services that they can provide. This situation leads to a continue cooperation between agents developed by different programmers and provided by different vendors, cooperation that often requires a data exchange. Often, the interaction with other agents is crucial for the success of the activities of an agent, so interactions must be carefully considered in agent-based applications.

In a static black and white world, an agent knows a priori whether interacting with another agent or not, while in a dynamic colored world many issues must be considered at runtime. The traditional approach based on security is no longer enough in a very dynamic, uncertain and unpredictable world such as the agents' one. As a first issue, a sure authentication may be not so easy to achieve in a wide environment such as the Internet. Second, the skill of the counterpart can be an important issue to decide whether to perform the interaction or not; an authenticated and secure agent could not provide the exact service needed, or it can provide the service not in the best way. In the evaluation of the skill, previous experiences can help in the decision. These considerations lead to a concept more flexible than security: *trust*. During an interaction between agents, it is important that each involved part can evaluate the trust that the interaction will have.

In this paper, we propose a preliminary study on an evaluation of trust level between two mobile agents, thanks to which agents will be able to start or reject an interaction with more confidence.

Our study is related to mobile agents, since they are an exploited technology in the development of distributed and Internet-based applications today. Furthermore, since interactions between agents are often exploited to carry out a task, and since there are good proposals that model interactions exploiting the concept of role [1, 4, 5, 6], our approach explicitly introduces the trust in the assumed role too.

The paper is organized as follows: section 2 introduces other concepts about trust, section 3 explains how our approach computes the trust level between agents, while section 4 details the Java implementation of our approach. Finally section 5 gives conclusions.

## 2. About Trust

The concept of trust applied to computer science is not new, and in fact we can find other studies in [2, 7, 11]. What these studies emphasize is that trust can be the compound result of trust assigned to different components, thus it is not possible to evaluate the global trust before having evaluated each component. Furthermore, trust depends on not immediately visible properties, and in particular it is based on the capability of predicting these properties and of trusting them. Finally, the trust level of an agent cannot be a fixed property of the single agent, but it depends also on the other agents with which it interacts (or have interacted).

The main difference between *security* and *trust* is that the latter is more subjective and context dependent. In fact, while security is typically set up before the execution of the application, allowing administrator to change rules during the application evolution, trust is decided by the application components themselves. In other words, while security is typically set up *externally* from the application, trust comes from the *inside* of the application, since it is evaluated by the running components themselves. Since trust comes within the application, without requiring external entities (e.g., administrators), this leads to a dynamic situation, where the application can take decisions considering the current environment.

Typically, what happens is that an agent starts an interaction with another agent only if the latter has a trust level greater than a threshold, which usually depends on the goal and the kind of task of the former agent.

It is important to note that building a system based on trust does not mean to simply apply one (or more) threshold to system parameters, since this would lead to a security-based application. To better explain this concept,

M. Baldoni, F. De Paoli, A. Martelli, and A. Omicini (Eds.): *WOA 2004 – Dagli Oggetti agli Agenti, Sistemi Complessi e Agenti Razionali*, Torino, Italy, November 29th – December 1st 2004. Pitagora Editrice Bologna. ISBN 88-371-1533-4. http://woa04.unito.it

imagine that an agent trusts another agent if it owns at least the 80% of a common secret password. This could mean that the first agent trusts the second at the 80%. But at a deeper look, this does not represent a trust threshold, but a security threshold. In fact, in the above situation, it is like if the common password must be shorter than the complete one to allow agents to interact, which means that the security level is lower than the one required with the complete password. Even if the threshold can be changed during the application, the situation can be always reconsidered as a security issue.

From the above example it should be clear that evaluating trust does not mean to simply apply variable thresholds. Trust requires other control mechanisms, and, in particular, the capability to evaluate and change trust levels autonomously during the application evolution. Nevertheless, even being able to evaluate and adapt thresholds during the application does not suffice, and it is for this reason that trust needs also *history*. In fact, only evaluating the trust level over different time instants it is possible to get a very subjective value.

Trust should always be computed dependently on the target of the action the agent is doing, since it is not possible to evaluate trust just related to another agent without considering also the action to perform. This means that there could be different trust levels among the same agents, depending on the actions/interactions they are doing together. Starting from the above considerations, it should be clear that the trust computation should also have a fine grain, depending on the involved agents and interactions.

Furthermore it is important to note that trust should not be considered negatively, but positively. In other words, it is more important to understand which could be the positive consequences of granting trust to a partner, rather than the negative ones (or risks) due to a bad evaluation [7]. In this situation interactions will be promoted, and not rejected due to a not 100% trust level.

The following section shows the formula we propose to evaluate trust level between agents.

## 3. Computing the Trust Level

Since agents are computational entities, they cannot evaluate the trust as humans do (i.e., based on emotions, feelings, intuitions, instincts, and so on). In order to allow agents to evaluate the trust related to other agents or components in a computational way, we propose the following formula:

$$T_{ij} = \frac{(1-S)\cdot A\cdot c_A + S\cdot(c_S + c_I\cdot I) + H\cdot c_H + P\cdot c_P + R\cdot c_R}{(1-S)\cdot c_A + S\cdot(c_S + c_I) + c_H + c_P + c_R}$$

where $T_{ij}$ represents the trust level of the agent *j* computed by the agent *i*. As detailed in section 1, the global trust can be evaluated only if all its components have been evaluated. In the above formula the terms $c_x$ represent weights of the several parameters. They say how much the agent wants to consider a given parameter in the evaluation of the trust. The parameters are the following:

- *S* indicates if the agent is signed or not. The value can be only 0 or 1, depending on the presence of the signature(s) of the agent;
- *A* stands for the authentication of the agent and can embed both *credentials* and *code type*. The former could be, for example, passwords or secrets useful to authenticate the agent or its owner. The latter represents an introspection on the agent code in order to understand, for example, the base classes used to build it, or if it contains dangerous instructions, etc.;
- *I* represents the identity of the signer of the agent (if present);
- *H* represents the *history* of the interactions of the agent *j*. The history is important in order to evaluate in a more subjective way the trust level of *j* perceived by *i*, thus the agent *i* can understand if agent *j* has been a bad agent in the past or not;
- *P* stands for the previous host of the agent. Since this work has been done explicitly in the mobile agent context, we have decided to explicitly insert the previous host parameter in the formula;
- *R* represents the trust of the agent *j* feel by the role assumed by the agent *i*. It can be computed with a formula similar to the above one:

$$R = \frac{(1-S)\cdot A\cdot c_{AR} + S\cdot(c_{SR} + c_{IR}\cdot I) + H\cdot c_{HR} + P\cdot c_{PR}}{(1-S)\cdot c_{AR} + S\cdot(c_{SR} + c_{IR}) + c_{HR} + c_{PR}}$$

where all the terms and the weights have the same meaning described above, even if, as also indicated by the weight subscripts, they are related to the role and not to the agent itself.

Why there is the need of evaluating trust even of the assumed role? First of all it is important to recall the fact that roles are *external* components to agents, which are exploited by them during the execution of the application. The fact that roles are external entities, and the fact that they are usually tied to the local execution environment [3], means that agents have no warranties about the piece of code they are going to exploit. It is for this reason that the trust level must include also the trust about the role (if there is one).

It is important to note that *S* is the only one parameter that can assume a boolean value, depending on the presence of the signature(s), while the other can be between 0 and 1. The weights $c_x$ are between 0 and 10[1], and this means that the final value of $T_{ij}$ will be always less or equal to 1:

$$S \in \{0,1\};\ A, I, H, P, R \in [0,1];\ c_x \in [0,10] \Rightarrow T_{ij} \in [0,1]$$

Please note that, as shown in the first formula, when the agent is not signed (i.e., *S*=0) the identity term *I* is not considered in the computation of the trust level, while when the agent is signed (*S*=1), the term of the authentication *A* is not considered. In fact, since the agent is signed, there is no need to authenticate it, but the signature(s) can be used as authentication as well.

A very important term in the first formula is *S*, which represents the history of the actions of the agent *j*, thus the agent *i* can try to understand if the opponent has been fair or not. The term *S* is not trivial to calculate, due to the fact that is not always simple to keep a track of the history of past actions of each agent, depending on the platform

[1] Please note that the trust level $T_{ij}$ will always be normalized, independently from the range of values the weights $c_x$ belongs to. The choice of the latter range depends on the level of granularity required by the computation.

implementation. What an agent can do, in the case that the platform does not provide support for the history of actions, is to keep a private track of actions/interactions with a specific agent, in order to be able to further evaluate the term *S* when needed. For example, an agent can progressively compute *S* with the following simple count:

$$S = \frac{succesfully\_done\_transactions\_with\_agent\_X}{totally\_done\_transactions\_with\_agent\_X}$$

while the weight $c_s$ should become greater as the successful transactions are recent or not.

All the capitalized terms in the first and second formulas represent parameters that are fixed for all agents, but what makes the formulas subjective is the use of weights $c_x$. In fact, while an agent should not change values of the parameters, it can change values of weights, in order to adapt the computation of the trust level to its execution environment.

### 3.1. Considerations about the Formula

It is important to note that the formula can be used to compute trust even if not all terms are available. For example, in some implementations the history (H) could not be present, thus agents have to compute the trust level with a “partial” formula. Of course, the use of an uncomplete formula will produce less reliable results, since the space of possible result values is shrinked.

Another important thing to note is that, even if “trust” is not the same of “security”, as already written, the formula is partially based on a set of security terms (like A and I), since we believe that first of all trustness should imply also security (but not vice versa).

Finally, please note that the choice of weights is in charge of the agent (and its developer), since the agent must evaluate by itself how much important are the information about the different interacting entity.

## 4. A Java Implementation

In order to ease the use of the first formula we have implemented a set of Java classes that Java agents can exploit. This section gives a presentation of this set of classes as first, and then briefly shows an application developed using IBM Robocode, which exploits the above classes.

### 4.1. Java Classes to Compute the Trust Level

All the classes are contained in a single package, `it.unimo.brain.trust`. It is important to note that, in order to grant a high flexibility in the computation of the trust level, almost all classes are abstract. Nevertheless, in order to give developers a library ready to use, we provided a subpackage, called `impl`, which contains default implementations of the main abstract classes.

Since the $T_{ij}$ is a sum of terms, each one composed of other sums or multiplications of a weight and a capitalized term, we introduced the base class *Term* (see Figure 1), which represents the result of a capitalized term and its weight. In this way it is quite simple to compute the whole formula, since developers have just to add each term, while the terms will compute themselves transparently.

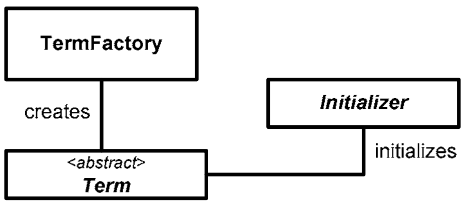

**Figure 1 Main classes of the Java implementation.**

Before it can be used, a term must be initialized, that means it must be able to compute the right value. For this reason we provided an interface, *Initializer*, which has been specialized for each term in order to load the right values. For example, in the case of the computation of *H*, the initializer must contain a table of known host and the values for the trust for each of them. To make all the formulas more flexible, we provided also a factory class that gives the current implementation of each term. The agent is just in charge of calling the method *getTerm* of the factory with the constant that identifies the term. The following piece of code shows an example of the initialization and use of the *S* term:

```
// get a new initializer for F
Initializer init = new SignatureInitializer();
// get the Term from the factory
Term S = TermFactory.getTerm(S_TERM);
// initialize the term
S.initialize(init,agent.getClass().getName());
. . .
// use the term
float weight = ...;
float val_S = S.getValue(weight);
```

It is important to note that the initialization does not provide the weight used in the formula, and this is to obtain a more dynamic system. In fact, since weights adapt the formula to the current context, and since they must be personalized for each agent, they must be provided at the moment of the computation, i.e., when the *getValue* method is called.

### 4.2. Exploiting the Formula in Robocode

In order to prove the usability of the first formula and of its Java implementation, we have tested it in an application developed using the IBM Robocode game platform [8, 9, 10]. Robocode is a Java platform used to implement simple Java games (see Figure 2), where developers can program robots (represented as tanks) that battle each other.

We have chosen this particular platform for two main reasons. First of all the scenario is very similar to the one of agents, and in fact, robots are free to move and interact each other, in a cooperative or competitive way. Furthermore, robots can cheat and can be cheated, and in this situation the computation of trust gain more importance. The second reason is that the use of Robocode gives developers a concrete visible evolution of the application, that means it is possible to understand how robots trust each other simply watching the battle. This is useful in particular in didactic experiences.

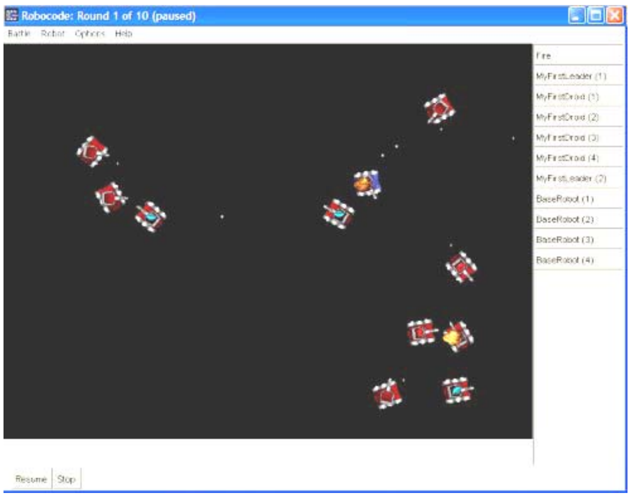

**Figure 2 The Robocode battle-of-trust.**

In the developed application, there is a particular robot that evaluates the trust levels between itself and the other robots, killing those it does not trust. This leads to a situation where only trusted robots survive.

Of course, in this simulation a few parameters of the first formula have just been set to the default values, since they do not have a specific meaning. For example, since the robots execute in the same host, the *H* parameter has been set to a value depending on the team they belong, in order to simulate the provenience from different hosts.

## Conclusions

In this paper we proposed a preliminary study for trust evaluation in agent interactions. Unlike other approaches, ours explicitly takes care of mobility and of the exploitation of roles in interactions.

Our approach is based on a formula, which allows agents to compute the trust level as composed of different components that include the history of previous interactions, in order to allow a complete evaluation. Thanks to the use of weights in the formula, which can be adapted for the current context, the formula is suitable for different situations and agents. This allowed us to develop a set of Java classes which can be exploited to compute the formula value (i.e., the trust level) in Java agent applications. We applied the formula also to other scenarios, similar to those of agents, in order to demonstrate that is quite general and can be easily adapted to different applications.

Future work includes a better evaluation of each component of the formula, in order to understand if they are complete or must be extended. Furthermore, a standardization of the computation of the history will help in the computation of trust.

**Acknowledgments:** Work supported by the Italian MIUR and CNR within the project "IS-MANET, Infrastructures for Mobile ad-hoc networks", and by the MIUR within the project "Trust and law in the Information Society. Fostering and protecting trust in the market, in the institutions and in the technological infrastructure".

## References

[1] D. Baumer, D. Riehle, W. Siberski, M. Wulf, “The Role Object Patterm”, Pattern Languages of Programming conference, 1997, Monticello, Illinois, USA

[2] P. A. Buhler, M. N. Huhns, “Trust and Persistence”, IEEE Internet Computing, April 2001, p. 85-87

[3] G. Cabri, L. Ferrari, L. Leonardi, “The Role Agent Pattern: a Developers Guideline”, in Proceedings of the 2003 IEEE International Conference on System, Man and Cybernetics, 5-8 October 2003, Washington D.C., U.S.A.

[4] G. Cabri, L. Leonardi, F. Zambonelli, “Separation of Concerns in Agent Applications by Roles”, in Proceedings of the 2nd International Workshop on Aspect Oriented Programming for Distributed Computing Systems (AOPDCS 2002), at the International Conference on Distributed Computing Systems (ICDCS 2002), Wien, July 2002

[5] G. Cabri, L. Leonardi, F. Zambonelli, “Modeling Role-based Interactions for Agents”, The Workshop on Agent-oriented methodologies, at the 17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2002), Seattle, Washington, USA, November 2002

[6] G. Cabri, L. Leonardi, F. Zambonelli, “Implementing Role-based Interactions for Internet Agents”, 2003 International Symposium on Applications and the Internet, Orlando (USA), January 2003.

[7] R. Falcone, O. Shehory, “Tutorial 12 – Trust delegation and autonomy: foundations for virtual societies”, Autonomous Agents & Multiagent Systems (AAMAS2002), Bologna, Italy, July 2002

[8] IBM Robocode Ufficial Site http://robocode.alphaworks.ibm.com/home/home.html

[9] IBM Robocode - API Documentation http://robocode.alphaworks.ibm.com/docs/robocode/index.html

[10] Sing Li, “Rock 'em, sock 'em Robocode!”, paper available on line at http://www-106.ibm.com/developerworks/java/library/j-robocode/

[11] J. Seigneur, S. Farrell, C. Jensen, E. Gray, Y. Chen, “End-to-end Trust Starts with Recognition”, Proceedings of the First International Conference on Security in Pervasive Computing, Boppard, Germany, March 2003