Decentralized Reputation Management for cooperating Software Agents in open Multi-Agent Systems

Andreas Grünert, Sebastian Hudert, Stefan König,
Sven Kaffille, and Guido Wirtz

Otto-Friedrich University of Bamberg
sven.kaffille|guido.wirtz@wiai.uni-bamberg.de

Abstract. Multi-Agent Systems (MAS) promise a new advance in distributed computing. In MAS autonomous software agents flexibly cooperate, coordinate and compete to provide the desired function(s) of such a system. If some components of a MAS fail or do not provide the desired functionality, the system is expected to autonomously deal with this situation. It is desirable to reduce occurrences of such situations by selecting trustworthy cooperation partners before cooperating with them. This becomes even more crucial in open MAS in which arbitrary heterogeneous software agents participate. In order to monitor agent behavior and enable selection of trustworthy cooperation partners, trust management services can be applied. As there is no central control in an open MAS and it is completely distributed, these services itself have to be distributed. This paper proposes a fully distributed reputation management service for open MAS based on peer-to-peer technology.

1 Introduction

This paper describes a fully distributed reputation management scheme to establish trust in open Multi-Agent Systems (MAS). Reputation management can be used as a foundation for agents to select trustworthy cooperation partners. For this purpose the paper is organized as follows. This section motivates why distributed reputation management is necessary in open MAS and describes the characteristics of open MAS. The second section exemplifies which requirements reputation management for open MAS must meet, while the third section deals with several aspects of the design of the proposed reputation management. It describes the architecture of our reputation management scheme and because of space limitations explains just the central protocol. Section four places our work in the context of other related work, before the final section summarizes the results of this paper and addresses open issues and future extensions to our approach.

Agents seem to be the next promising paradigm in Software Engineering for complex distributed systems. Currently there is no consensus on a definition what an agent is, but most definitions agree that an agent has to be autonomous,

This is a preprint of the article published in: H. Tanfield (Editor), *International Transactions on Systems Science and Applications*, Volume 1, Number 4, 2006.
http://itssa.xiaglow-research.org.uk/toc/1-4.htm

situated in an environment , must be able to percept, react to, and change this environment [9]. Furthermore a single agent can be reactive or a complex deliberative entity. A MAS is according to [9] ,,a loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each problem solver", where no global control exists, the agents are heterogeneous, and may be self-interested. In an open MAS heterogeneous selfish agents of different developers may enter the system.

For execution of software agents and MAS, distributed runtime environments and services are required (e.g. directory and communication services [6]). In this paper a runtime environment providing such services is called an agent platform (or just platform). A MAS may consist of many different platforms with different owners. To provide an open MAS it must be possible that agent platforms on different machines can be connected to facilitate cooperation between the agents residing on these platforms. It must also be possible that new platforms and new agents can enter the MAS at runtime.

Because of this dynamic nature of open MAS it is reasonable to base a MAS on Peer-to-Peer (P2P) technologies, as these kind of systems provide mechanisms to dynamically handle the arrival and departure of new machines in a network. The services (e.g. discovery services) required by agents are then provided by all the platforms that are connected within the P2P network to equally distribute load. Moreover the whole network becomes scalable as with the arrival of new platforms these platforms will also provide new resources to provide services.

The limited capabilities and knowledge of agents to solve problems on their own makes cooperation necessary, which can be structured in MAS with help of the Cooperative Problem Solving (CPS) process[13]. CPS is divided into four phases: recognition, team formation, plan formation, and team action. As CPS is an iterative process, this phases have not to be processed sequentially.

As it cannot be assumed, that agents are sincere, in order for CPS to be successful in open MAS, agents must have a basis for the decision with which agents to cooperate. This is important in the team formation phase in order to decide which agents should be in the team as they will most likely fulfill their assigned tasks. In phase three (plan formation) it may be helpful to have knowledge about the reliability of agents regarding the tasks they can carry out. In open MAS at the end of the fourth phase of CPS an additional fifth phase, which records the experiences agents make with each other, should be carried out.

One possibility explored in MAS research to identify and exclude agents that behave badly is Trust Management ([5], [14]). Therefore another useful service which should be provided by an agent platform, in such a dynamic environment is a service to manage trust among the agents involved. This service in an open decentralized environment with no global control should also be distributed equally among platforms constituting a MAS. On the one hand distribution of trust management among platforms ensures that there exists no single point of failure or a single entity that can be attacked. On the other hand it accounts for

This is a preprint of the article published in: H. Tanfield (Editor), *International Transactions on Systems Science and Applications*, Volume 1, Number 4, 2006.
http://itssa.xiaglow-research.org.uk/toc/1-4.htm

scalability as argued above. Developing such a service is the central goal of our work.

Trust can be used as a foundation for reasoning about with whom to interact in situations where only partial information about the partner is available and there is a risk that the interaction may be harmful. Trust depends on situation and the context, in which it is validated [3]. Trust between two individuals is not necessarily symmetric. In MAS the context, in which an agent can be trusted, can be defined by using the role an agent plays[4][1].

The primary source used for building trust to someone would be a subjective image [12] one can have of the other party. If such an image is not available, one has to resort to other information as e.g. reputation values provided by a reputation management system. Reputation management [15] facilitates estimation of trust between agents based on a history of interactions.

Reputation management is mostly realized with help of a central trusted entity, which stores reputation values for pairs of agents that have interacted before. If such a central entity is present, an agent can ask it (instead of asking all agents known by it) for all interactions, in which the agent under consideration has been involved, and how other agents rated these interactions.

In order to make reputation interpretable and computable by software it is often represented by numerical values (e.g. real numbers between 0 and 1). The reputation value for a single agent is then computed with help of a so called reputation metric [8].

Reputation information about an agent can be seen by all other agents including the affected agent itself. Therefore the agent can estimate if other agents may have trust in it and how it is rated compared to other agents. This will give the agent an incentive to keep its reputation as high as possible by behaving well in any interaction. Ratings of agents that rated others, but have already left the system are still maintained by reputation management.

According to [12], reputation can be structured in three dimensions: individual, social, and ontological. Our approach assists the social dimension by providing a reputation management service which itself is completely distributed. The ontological dimension can be addressed with help of the roles an agent plays, but is not covered in this paper. In our approach each single agent is responsible for the individual dimension on its own.

2 Requirements on Reputation Management

The reputation management service must enable agents to rate and request the reputation of other agents. Agents must be free to leave and enter the MAS at any time. If an agent leaves and it has gained a certain reputation, it should

[1] In order to provide more information about the context the commitment underlying a cooperation and information about the environment where the cooperation takes place may be incorporated. This has to be examined further and to be integrated in our work in the future.

This is a preprint of the article published in: H. Tanfield (Editor), *International Transactions on Systems Science and Applications*, Volume 1, Number 4, 2006.
http://itssa.xiaglow-research.org.uk/toc/1-4.htm

be able to maintain this reputation, and reclaim it when it reenters the system. This must be possible at any of the distributed agent platforms.

As reputation is context-dependent it is inevitable to save reputation information for different contexts. These contexts are provided by CPS processes that are conducted by the agents and the role(s) agents play during these. Hence, it is necessary to rate the agents with help of the role(s) they play during CPS.

As the MAS is distributed itself, the reputation management service of a platform, should be distributed in a P2P-fashion. There should exist no central global control, so that every participant of a MAS has to provide resources for reputation management. P2P technologies also facilitate an open network, where platforms and agents can join and leave dynamically. By distributing reputation management in a P2P-fashion it is more likely that it scales with the number of participating platforms and agents. In order to facilitate reputation management, agents must be uniquely identifiable, so that they can be rated by other agents. Therefore an identification and authentication mechanism must be available (see 3.2 for details).

In order to assure that agents can only rate others if there has been a cooperation (referred to as *transaction* in the following), the reputation management service of a platform must keep track of transactions with help of transaction identifiers. To ensure that agents do not request a transaction identifier without giving a rating for the requested transaction, a protocol must ensure that ratings are delivered after a transaction. Also a mechanism that motivates agents to leave the MAS in an orderly fashion should be provided. These two mechanisms can be realized with help of a leasing concept for subscription to the system and transactions. This leasing mechanism must be distributed among several platforms. It cannot be realized by the single platform, that hosts the agent for which it manages a lease, as the platform may cooperate with the agent.

To implement these basic requirements some additional requirements have to be imposed. A P2P scalable and load balanced overlay network that provides a means to store reputation data and information about agent identities must be available. For this purpose a distributed hash table (DHT)[2], with additional mechanisms for ensuring data integrity and consistency, can be used. The idea is to use the structure of the DHT to control which platform is responsible for which part of the reputation data. Additionally, no single platform should be responsible for storing a specific chunk of data, as it cannot only manipulate, but also remove data. Therefore reputation data should be distributed among more than one platform.

Hence trustworthiness of platforms themselves has to be tracked with help of a platform reputation. This requires reputation management on the level of agent platforms to make sure that only trustworthy platforms participate in the P2P network. A platform has to be regarded as being trustworthy if it adheres to the protocols required for the reputation management system to function correctly.

Mechanisms must be developed that provide identification and authentication based on well-understood cryptographic concepts. Furthermore mechanisms

This is a preprint of the article published in: H. Tanfield (Editor), *International Transactions on Systems Science and Applications*, Volume 1, Number 4, 2006.
http://itssa.xiaglow-research.org.uk/toc/1-4.htm

to handle agent arrival and departure, where agents provide existing reputation data concerning themselves must be provided. These mechanisms have to ensure validity of the provided data as well. All these mechanisms require that platforms can communicate with each other in order to coordinate their activities. The integrity and reliability of this communication must be secured. To meet these requirements the architecture described in the next section has been developed as a plug-in for existing agent platforms to provide reputation management.

3 Decentralized reputation management design

3.1 Architecture

The plug-in is to be used as a basic service of agent platforms. Thus, its functions are invoked by agent platforms or directly by agents running on that platform via a single interface of the `Trust Service`. By applying the facade pattern here, the complexity of the application can be encapsulated within the plug-in and therefore is not visible to the platform, the agents or their programmers. The reputation management plug-in consists of four different layers. Each layer

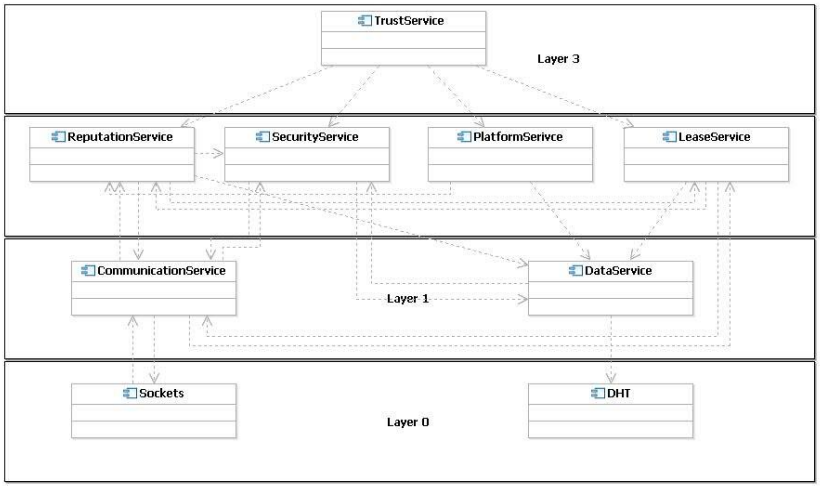

Fig. 1. Architecture of reputation management service.

performs different kinds of tasks and provides its functionality to the layers above via an interface. The basic **layer 0** consists of two different components. One is a DHT, used to store relevant data, such as information about agents and platforms and reputation data. And the other is a socket-based communication component for direct communication between different platforms.

This is a preprint of the article published in: H. Tanfield (Editor), *International Transactions on Systems Science and Applications*, Volume 1, Number 4, 2006.
http://itssa.xiaglow-research.org.uk/toc/1-4.htm

Layer 1 abstracts from the technical basis of layer 0. The `Communication Service` offers means to communicate transparently with other platforms. In order to prevent denial of service attacks or unintentional communication requests from platforms outside the network, the Communication Service is also responsible for communication security issues, e. g. authentication and authorization. The `Data Service` provides operations to store data in and retrieve data from the underlying DHT. This abstraction grants the possibility to replace the employed implementation of DHT. The `Data Service` is also responsible to replicate data within the DHT to facilitate consistency checks. Within the structured overlay network of the DHT each platform is responsible for certain data. The responsibility is determined with help of identifiers associated with data (e.g. identification data of agents/platforms, ratings of agents).

The following services, arranged on **layer 2**, implement the core logic for our reputation management. The `Reputation Service` conducts the process of rating agents and agent platforms. i.e. the service calculates ratings for agents and agent platforms using a given rating metric and a reputation-value representation. The ratings are based on data retrieved with help of the `Data Service`. Neither the rating metric nor the rating representation is hard-wired, but given as a parameter when setting up the system (strategy pattern). Thus, it is possible to use different kinds of rating metrics and rating representations, respectively. This concept allows flexible use of the plug-in in a wide range of scenarios. The metric and reputation value representation used has to be provided when a MAS is created and is the same for all platforms contributing to the MAS. The `Reputation Service` also creates so-called tickets for transactions, that must be signed by all agents participating in a cooperation.

The `Security Service` provides services to secure communication between platforms and data storage in the DHT. This is done utilizing the techniques of signing and ciphering data using symmetric and asymmetric algorithms. The service provides a platform and each agent running on it with a private[2] and a public key. Data stored in the DHT by a participant of the system (agent or platform) has to be signed to ensure integrity. In some cases a second signature from another platform is required to guarantee data integrity. The socket communication between platforms is secured analogously to the Pretty Good Privacy protocol [16]. In order to do so, a means to create temporary session keys for ciphering the communication between different agent platforms is provided. The Security Service also validates signatures and deciphers data.

In order to enter a network, an agent platform invokes the `Platform Service`, which also resides on layer two. The service is used to enter an existing network, create a new network or to leave a network. Moreover, this service addresses data consistency in the DHT by conducting consistency checks. Though platforms are not allowed to store data in the DHT without consent (by signature) of another platform, data might be removed by a platform. In consequence, a platform holding incomplete data will be identified and rewarded with an appropriate rating

[2] How private data of agents and platforms is being secured is the responsibility of the agents and platforms themselves.

This is a preprint of the article published in: H. Tanfield (Editor), *International Transactions on Systems Science and Applications*, Volume 1, Number 4, 2006.
http://itssa.xiaglow-research.org.uk/toc/1-4.htm

by the `Platform Service`. Platforms with very low reputations are finally excluded from the network. Exclusion is implemented with help of the structure of the DHT, when trustworthy platforms remove untrustworthy platforms from their routing tables.

It is not possible for the plug-in to force agents or platforms to use the logout function to leave the network. Thus, it cannot always be determined if an agent is still running. For this purpose and to motivate agents to return the ticket, a Lease Service is introduced. A corresponding lease exists for each agent logging into the plug-in and for each requested ticket (and therefore for each intended transaction). If these leases are not renewed (or ticket handed back respectively) in time, expiration of a lease will result in a bad rating for the corresponding agent. Thus agents leaving without notifying the system will suffer constant downgrading of their reputation values and therefore should be motivated to sign off properly.

Finally **layer 3** consists of the `Trust Service` providing a uniform interface for the whole plug-in.

3.2 Attacks and counter measures

Attacks on the reputation management system can be categorized in *attacks by platforms* and *attacks by agents*. These attacks and appropriate counter measures are discussed in the following.

A platform *does not save or deletes reputation data of an agent* or *manipulates* existing reputation data of an agent. A platform can also *create fake reputation data* for an agent. The whole consistency check concept is solely introduced to reveal platform behaviors like these. Whenever one of them is discovered, the corresponding platform will get a bad rating and, if it continues acting incorrectly, is excluded from the network.

A *platform* tries to make reputation data unavailable by flooding other platforms with *unnecessary requests (Denial of Service)*. The first request to a running platform arrives at a Communication Service. As a general policy, all requests will be discarded, if the Communication Service identifies them as part of a Denial of Service attack by using a simple request count.

A *platform* that has been identified as *not-trustworthy*, as it tried to perform one or more of the attacks described above, is still or again *trying to participate* in reputation management. Participation in the network can occur in two ways: Request-response communication via the Communication Service and saving data in the DHT via the Data Service. The former is handled by the Communication Service of the receiving platform, which will block all requests from non-trustworthy platforms. The latter is prohibited by a mechanism in the DHT, which removes non-trustworthy platforms from the routing tables and only provides trustworthy platforms with access to the DHT for data storage and update.

An *agent pretends to be trustworthy* for a couple of transactions to create a good reputation value and then changes its behavior to exploit others. All ratings are saved, latest rating first, in priority queues. To calculate the overall

This is a preprint of the article published in: H. Tanfield (Editor), *International Transactions on Systems Science and Applications*, Volume 1, Number 4, 2006.
http://itssa.xiaglow-research.org.uk/toc/1-4.htm

reputation value a formula like exponential smoothing can be used with stronger weights for newer ratings [11]. Thus, this attack can be averted because the influence of the initial good ratings will diminish over time. This depends on the metric employed, which as described above can be exchanged and adapted to the application domain as needed.

Agents rate each other based on a *faked transaction*. This attack can not be prevented completely since a plug-in cannot ensure, that a transaction is really taking place. It is only possible to verify ratings by checking the signatures of the ticket, the valuations and the distributed ticket lease. An agent might not fetch a ticket before starting cooperative work so it cannot be rated afterwards. However, other agents can refuse to cooperate if no ticket was passed around.

An agent *cooperates with other agents* to get a high reputation value. To impair the unmeant effects of faked ratings, only one rating per agent pair (rated and rating agent) and role is used to calculate the reputation value. Thus, there are no serious effects on the overall reputation of an agent when two agents give faked (high or low) ratings to each other.

An agent rates another agent with a *bad reputation value without reason*. This attack subsumes two cases: The first one - the ticket of a packet is not signed by all agents involved - is completely prevented. If such a packet comes back to a platform, it is ignored, because not all agents rated in this structure have signed the ticket. The second case occurs if an agent signs a ticket and is rated poorly although it has done its job well. This problem is again out of range of the plug-in, but is also impaired like described in the attack above (faked ratings).

An *agent leaves* because of a bad reputation *and creates a new identity* to get rid of the bad reputation. When leaving the system, an agent takes his cumulated reputation data with it. Therefore it can, in case the reputation is very bad, be tempted to come back with a new ID, which is possible since no other agent or platform would recognize this identity change. To handle this possible attack, every agent registering to the plug-in initially receives a neutral value, not a null value. Thus, every other participant requesting the reputation of this agent can decide individually how to treat such a "new" agent. The agent can also keep its ID and just come back without its reputation data. In this case the lease mechanism will ensure that the agent's reputation will be very low when the agent returns after a certain timespan. Hence it will always be disadvantageous to come back with the old ID but without the latest reputation data structure. In case the registering agent comes back with a reputation data structure belonging to another agent, even if it took over the ID of the other agent ID, it would not be able to log in because of the invalid signatures on the reputation data structure (the returning agent has to sign the reputation data). If the agent also got hold of the other agents private key there is no way to prevent the agent from registering correctly because, the platform cannot proof the identity of the agent. It seems to them that the correct agent registers again. Finally an agent might leave the system once it has a very good reputation. Afterwards it might use this good reputation each time it logs into the system again. Thus, it could behave harmful, get a low reputation, log out and then log in with its high reputation again. To

This is a preprint of the article published in: H. Tanfield (Editor), *International Transactions on Systems Science and Applications*, Volume 1, Number 4, 2006.
http://itssa.xiaglow-research.org.uk/toc/1-4.htm

avoid this, the up-to-date reputation data, given out to every agent, is stored in a queue until the agent returns to enable reputation comparison. Thus, an agent can only register with its up-to-date reputation data structure.

Regardless of how good a reputation management is guarded against attacks, the system cannot fulfill its task if the participating entities are not identified properly. Hence, there must be means to ensure, that at any time any participating entity can be identified unambiguously. In the presented system, this is realized by creating a unique identifier for each participant, so-called `platform identifier` and `agent identifier` which are signed by the participant itself and another network entity to forestall unauthorized identity changes.

If an agent platform wishes to login into an existing network, it first creates a `platform identifier` which contains the unique identifier of the platform, its public key and its network address. The object is then signed with the private key of the platform and sent to a trusted platform which is already part of the network. That platform also uses its private key to sign the `platform identifier` and stores it within the DHT. In case a platform is the very first platform and thus is creating a new network, there is no other platform to sign the first `platform identifier` a second time. But as soon as a second platform joins the network, this is made up for.

A similar procedure is performed if an agent wants to join an existing network. However, the process is slightly more complicated since an agent can assume different roles during its life, which are recorded within the `agent valuation`. The `agent valuation` must not be signed itself, but contains a collection of the current roles of an agent, associated with signed ratings, as well as the `agent identifier`.

The important principle to note is, that no network entity can prove its identity on its own, but always needs a second trusted entity to testify.

3.3 Rating protocol

The most crucial part of reputation management is how ratings are made. Our rating protocol is described in the following.

Given the case, that an agent wants to cooperate with other agents and intends to rate all agents in the role(s) executed by them during the cooperation, the following procedure is executed:

1. The agent that initializes the cooperation (*agent I* in the following) requests a new **ticket** from its **platform** and states an expected duration of the intented cooperative work and its identifier. The ticket is created and signed by the platform, signed by another trusted platform, and then returned to the requesting agent. The expected duration and the identifier are stored within the ticket for later use.
2. *Agent I* passes around the ticket to all agents participating in the cooperative work. This takes place during the CPS process in the team formation phase after the joint commitment[13] has been established. Each agent signs the ticket and thus signals readiness for cooperation and adherence to the joint

This is a preprint of the article published in: H. Tanfield (Editor), *International Transactions on Systems Science and Applications*, Volume 1, Number 4, 2006.
http://itssa.xiaglow-research.org.uk/toc/1-4.htm

commitment. Further it adds a time stamp (of its local time). This time stamp allows ordering the transactions without requiring a global time.

3. After all relevant agents signed the ticket, *agent I* requests a **packet** from its **platform**. The agent therefore passes the signed ticket to the platform, which checks if *agent I* is part of the reputation network and if the ticket is valid. If both checks return satisfactorily, the platform requests **ticket leases** from platforms that are also responsible for storage of data about *Agent I*, (called *platforms L* in the following) for the duration stored within the ticket. If these leases are returned, the platform creates a **packet**, i.e. a data structure composed out of the given ticket and an empty collection where rating values can be entered later. The packet is then returned to *agent I*.
4. The agents plan their joint activity (plan formation) and afterwards the cooperative work takes place (team action).
5. After the cooperation has finished, *agent I* passes around the packet to all participating agents, which store their rating values[3] for each other in the packet. Every agent has to sign its ratings to ensure traceability of who gave which rating to whom and data integrity.
6. After all relevant agents have stored their rating values, the packet is given to a platform. The platform then asks *platforms L*, whether the packet is still valid, which is determined by looking at the corresponding ticket leases. If the packet is still valid, each rating value is checked for a proper signature and if signed properly, stored within the DHT.

The information stored by the reputation management system can be considered during the team formation and plan formation phase of CPS. In the former phase the agent can use the ratings, a potential team member received earlier, to estimate if the agent is cooperative at all. In the latter it can be estimated to what extend an agent is able, disposed, and intending ([5]) to execute its assigned role(s) during CPS if it has been rated in that role(s) by other agents before.

A prototype implementation of our reputation management system as described in this section has been realized in Java. As DHT a Java implementation of chord[10] has been employed and extended with concepts to support our reputation management service (e.g. the join protocol was changed). For a more detailed description of our implementation refer to [7].

4 Related Work

There are two approaches we see in close relation to our approach. The first approach is described in [1]. Instead of `Chord` it uses the decentralized storage method `P-Grid`. In this approach only information on dishonest interactions is considered as relevant and the result of a reputation calculation yields only a

[3] The type of values and their semantics rely on the concrete reputation metric and values employed for reputation management.

This is a preprint of the article published in: H. Tanfield (Editor), *International Transactions on Systems Science and Applications*, Volume 1, Number 4, 2006.
http://itssa.xiaglow-research.org.uk/toc/1-4.htm

result that indicates if an agent is untrustworthy (`0`) or trustworthy (`1`). In our system all interactions are considered relevant to be able to get a more concise impression of agent behavior. In our approach the metric to calculate reputation of an agent is not an integral part of the system. But [1] reveals also some facts in common with our system. The reputation data is of global availability and also the scalability issues are addressed similar to our approach using chord.

The second approach, called AVALANCHE, is based on ideas of the Institute for Computer Science and Social Studies of Freiburg, Germany (see e.g. [11]). One part of this project was implementing a reputation mechanism. This approach has a sanction component that prevents contacting agents with low reputation like our system does with platforms. However in AVALANCHE the metrics are hard-wired where in our work the metrics are exchangeable in AVALANCE it is hard-wired. But the main difference is, that AVALANCHE uses a central trusted entity, while our approach employs a logical central entity, which is physically distributed and therefore our approach has no single point, that can be attacked to disrupt reputation management.

5 Conclusion

This paper presented a fully decentralized scheme for reputation management in MAS based on P2P technology. It provides protocols to store, update, and retrieve reputation data for agents and platforms and to counter measure attacks on reputation management. The application of our reputation management scheme in many domains is supported by its implementation in Java, its modular design, and the possibility to exchange the employed reputation metric and reputation value representation.

Mechanisms to identify and authenticate agents and platforms have been developed. These mechanisms should be extended to rely on the owner of an agent or a platform, so that more sophisticated methods to prevent attacks on reputation management are possible. Knowledge about the owner of an agent or a platform adds extra possibilities to avoid faked ratings and transactions. It also facilitates tracking of agent and platform owner behavior. For this purpose the data structures to identify agents and platforms can be extended to also contain the identity of their owners. The owners should have an identity certified by a trusted third party outside the MAS. To keep track of an owner's reputation it would be possible to treat it as a special agent with the special role owner, that then can be used with the existing mechanisms presented in this paper. Then, it would be possible to prevent new platforms or agents of owners with low reputation to enter the MAS, and this would increase the overall trust in the MAS. To further develop our approach it will be evaluated in different application domains and with different reputation metrics. An additional step is the extension by services supporting the ontological dimension (described by [12]) by providing knowledge about roles and their relationships. These services may be used by all agents to store their experience about the relationships that

This is a preprint of the article published in: H. Tanfield (Editor), *International Transactions on Systems Science and Applications*, Volume 1, Number 4, 2006.
http://itssa.xiaglow-research.org.uk/toc/1-4.htm

exist between roles. This can facilitate the estimation of trust in an agent in a role based on other roles, which are related to that role.

References

1. Aberer, K., Despotovic, Z.: Managing Trust in a Peer-2-Peer Information System (2001), Proceedings of the Ninth International Conference on Information and Knowledge Management (CIKM).
2. Balakrishnan H., Kaashoek M. F., Karger D., Morris R., Stoica I.: Looking Up Data in P2P Systems (2003), in: Communications of the ACM, vol. 46, 43-48
3. Cahill, V., Shand, B., Gray, E., Bryce, C., Dimmock, N.: Using trust for secure collaboration in uncertain environments, in: IEEE Pervasive Computing (2003), vol. 2, 52-61
4. Carter, J., Bitting, E., Ghorbani, A.: Reputation Formalization within Information Sharing Multiagent Architectures (2002), in: Computational Intelligence, vol. 18, 45-64
5. Castelfranchi, C., Falcone, R.: Principles of Trust for MAS: Cognitive Anatomy, Social Importance and Quantification (1998), in: ICMAS '98: Proceedings of the 3rd International Conference on Multi Agent Systems
6. FIPA Abstract Architecture Specification (2002), Foundation for Intelligent Physical Agents
7. Fischer M., Grünert A., Hudert S., König S., Lenskaya K., Scheithauer G., Kaffille S., Wirtz G.: Decentralized Reputation Management for cooperating Software Agents in open Multi-Agent Systems (2006), in: Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik, ISSN 0937-3349 (to be published)
8. Golbeck, J., Hendler, J.: Accuracy of Metrics for Inferring Trust and Reputation in Semantic Web-Based Social Networks, in: Proceedings of 14th International Conference on Knowledge Engineering and Knowledge Management (2004)
9. Jennings N. R., Sycara, K., Wooldridge, M.: A Roadmap of Agent Research and Development, in: Autonomous Agents and Multi-Agent Systems (1998), vol. 1, 7-38
10. Kaffille, S., Loesing, K.: Open Chord version 1.0 - Users Manual Chair of Practical Computer Science and the Distributed and Mobile Systems Group, University of Bamberg, http://open-chord.sourceforge.net
11. Padovan, B., Sackmann, S., Eymann, T., Pippow I.: Automatisierte Reputationsverfolgung auf einem agentenbasierten elektronischen Marktplatz (2001), in: Internationale Tagung Wirtschaftsinformatik 2001, 517-530
12. Sabater, J., Sierra, C.: Social ReGreT, a reputation model based on social relations, in: SIGecom Exch. (2002), vol. 3, 44-56
13. Wooldridge, M. J., Jennings N. R.: Cooperative Problem Solving, in: Journal of Logic and Computation (1999), vol. 9, 563-592
14. Wong, H. C., Sycara, K. P.: Adding Security and Trust to Multiagent Systems. in: Applied Artificial Intelligence (2000), volume 14, 927-941
15. Zacharia, G., Maes, P.: Trust Management Through Reputation Mechanisms, in: Applied Artificial Intelligence (2000), vol. 14, 881-907
16. Zimmermann, P.: The official PGP User's Guide. MIT Press, Cambridge, MA, USA, 1995

This is a preprint of the article published in: H. Tanfield (Editor), *International Transactions on Systems Science and Applications*, Volume 1, Number 4, 2006.
http://itssa.xiaglow-research.org.uk/toc/1-4.htm