OPEN ACCESS
OPUS AUGSBURG

Universität Augsburg
Universitätsbibliothek

Implementing an adaptive higher level observer in trusted desktop grid to control norms

Jan Kantert, Hannes Scharf, Sarah Edenhofer, Sven Tomforde, Jörg Hähner, Christian Müller-Schloer

Angaben zur Veröffentlichung / Publication details:

Kantert, Jan, Hannes Scharf, Sarah Edenhofer, Sven Tomforde, Jörg Hähner, and Christian Müller-Schloer. 2014. "Implementing an adaptive higher level observer in trusted desktop grid to control norms." In Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics, September 1-3, 2014, in Vienna, Austria, edited by Joaquim Filipe, Oleg Gusikhin, Kurosh Madani, and Jurek Sasiadek, 288–95. Setúbal: SciTePress. https://doi.org/10.5220/0005057902880295.

Nutzungsbedingungen / Terms of use: **CC BY-NC-ND 4.0**

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
CC-BY-NC-ND 4.0: Creative Commons: Namensnennung - Nicht kommerziell - Keine Bearbeitung
Weitere Informationen finden Sie unter: / For more information see:
https://creativecommons.org/licenses/by-nc-nd/4.0/deed.de

Implementing an Adaptive Higher Level Observer in Trusted Desktop Grid to Control Norms

Jan Kantert[1], Hannes Scharf[1], Sarah Edenhofer[2], Sven Tomforde[2],
Jörg Hähner[2] and Christian Müller-Schloer[1]
[1]*Institute of Systems Engineering, Wilhelm Gottfried Leibniz University Hanover, Appelstr. 4, 30167 Hanover, Germany*
[2]*Lehrstuhl für Organic Computing, Augsburg University, Eichleitnerstr. 30, 86159 Augsburg, Germany*

Keywords: Adaptive Control Loop, Multi-agent-Systems, Trust, Norms, Desktop-grid System.

Abstract: Grid Computing Systems are examples for open systems with heterogeneous and potentially malicious entities. Such systems can be controlled by system-wide intelligent control mechanisms working on trust relationships between these entities. Trust relationships are based on ratings among individual entities and represent system-wide information. In this paper, we propose to utilise a normative approach for the system-level control loop working on basis of these trust values. Thereby, a normative approach does not interfere with the entities' autonomy and handles each system as black box. Implicit rules already existing in the system are turned into explicit norms – which in turn are becoming mandatory for all entities. This allows the distributed systems to derive the desired behaviour and cooperate in reaction to disturbed situations such as attacks.

1 INTRODUCTION

Organic Computing (OC) (Müller-Schloer, 2004) postulates that the steady growth in complexity of technical systems demands for a paradigm shift. Instead of anticipating all possible system configurations during the design process, a system has been empowered to adjust itself during runtime – design time decisions need to be transferred into the system's runtime components. This architectural paradigm shift allows us to engineer systems that can provide flexible, adaptive, and robust solutions – what we call "life-like" properties.

Typically, OC systems are characterised by multiple autonomous and distributed entities - to which we refer as "agents". Despite their autonomy, which we have to take care about maintaining an efficient, scalable and robust behaviour at system-level, while simultaneously guaranteeing the openness and autonomy. Openness also implies that malicious agents can participate. Therefore, a trust-metric has been developed in previous work that serves as a basis to isolate malicious agents within the overall system (Bernard et al., 2010).

A *Trusted Desktop Grid* (TDG) is a perfect example of such an open system: Agents provide (computing) resources to calculate (parallelisable) jobs for other agents and can make use of the available resources of others to achieve a better performance for their own jobs. Thereby, agents may decide to whom to assign work and from which agents to accept work. As such a *TDG* is an open system by design, malicious agents such as *Freeriders* (meaning they do not work for other agents) or *Egoists* (meaning they return fake results) may also join the system. In order to guarantee a stable system performance and maintain a potential benefit for fair users, these malicious agents have to be handled as attackers and therefore become isolated.

The isolation in such an open system is done using trust values. Therefore, agents rate each other based on their experiences in terms of reliability and trustworthiness. The corresponding trust values are taken into account when deciding about acceptance of work or the assignment of work to others. This paper introduces a novel approach to convert the internal rules of agents (i.e. those leading to trust ratings) into explicit norms. The basic idea is that agents should become aware of how their desired behaviour is. Consequently, they can learn to act in accordance with the overall system goal although they do not have this goal function given in their design. A second major benefit of this approach is that norms can be instantiated in case of disturbed situations, where the normal

Kantert J., Scharf H., Edenhofer S., Tomforde S., Hähner J. and Müller-Schloer C..
Implementing an Adaptive Higher Level Observer in Trusted Desktop Grid to Control Norms.
DOI: 10.5220/0005057902880295
In *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2014), pages 288-295
ISBN: 978-989-758-039-0
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

trust-based cooperation system reaches its limitations. As a result, a change of norms can adapt the overall system to a dynamically changing environment.

The remainder of this paper is organised as follows: Section 2 explains the application scenario. Afterwards, Section 3 introduces our adaptive control loop to control norms in such a self-organised OC system. Section 4 describes the challenges to observe the system state in this application scenario. Based on this system model, Section 5 evaluates different metrics in simulation. Finally, Section 6 summarises the paper and gives an outlook to future work.

2 TRUSTED DESKTOP GRID

In this work, we analyse how to deal with open distributed systems. To understand such systems, we use multi-agent systems and model nodes of the system as agents. Our application scenario is an open distributed Desktop Grid System. We want participating agents to cooperate to gain an advantage. Every agent works for a user and periodically gets a job, which contains multiple parallelisable work units. Its goal is to get all work units processed as fast as possible by requesting other agents to work for it. The performance is measured by the speedup:

$$speedup = \frac{time_{self}}{time_{distributed}} \tag{1}$$

In general, agents behave selfishly and only cooperate if they can expect an advantage. They have to decide which agent they want to give their work to and for which agents they want to work.

Since we consider an open system, agents are autonomous and can join or leave at any time. If no cooperation partners can be found, agents need to calculate their own work units and achieve a *speedup* value equal to one. We do not control the agent implementation, so they may be uncooperative or even malicious and there can not be any assumption of benevolence. Such a system is vulnerable to different kinds of attacks. A *Freerider* can simply refuse to work for other agents and gain an advantage at the expense of cooperative agents.

The global goal is to enable agents, which act according to the system rules, to achieve a good speedup. We measure the global goal either by the average speedup of the well-behaving agents or by the amount of cooperation (eq. 2) combined with the average submit-to-work-ratio of all agents (eq. 3).

$$cooperation = \sum_{i=1}^{n} \sum_{j=1}^{n} ReturnWork(A_i, A_j) \tag{2}$$

$$fairness = \sum_{i=1}^{n} min(submit_i - work_i) \tag{3}$$

To overcome the problems of an open system, we introduced a trust metric (Klejnowski et al., 2010). Every agent gets ratings for every action it takes. This allows us to make an assumption about the general behaviour of an agent based on its previous actions (Klejnowski et al., 2010). In our system, we give agents a good rating if they work for other agents and a bad rating if they reject or cancel work requests. As a result, we can isolate malevolent agents and maintain a good system utility in most cases. We call this system a *Trusted Desktop Grid* (TDG) (Bernard et al., 2010).

We consider the following agent types in our system:

- *Adaptive Agents* - These agents are cooperative. They work for other agents who have good reputation in the system. How high the reputation has to be generally depends on the estimated current system load and how much the queue of the agent is filled up.
- *Freerider* - Such agents do not work for other agents and reject all work requests. However, they ask other agents to work for them. This increases overall system load and decreases the utility for well-behaving agents.
- *Egoists* - These agents only pretend to work for other agents. They accept all work requests but return fake results to other agents, which wastes the time of other agents. If results are not validated, this may lead to wrong results, otherwise, it lowers the utility of the system.
- *Cunning Agents* - These agents behave well in the beginning but may change their behaviour later. Periodically, randomly, or under certain conditions they behave like *Freeriders* or *Egoists*. This is hard to detect and may lower the overall system utility.

We simulate an attack by adding new malicious agents to the system at startup or during runtime. Since these malicious agents distribute their work, the speedup for well-behaving agents decreases. However, those agents get bad ratings such that their reputation in the system is reduced. At this point, other agents stop to cooperate with these isolated agents. We try to minimise the impact and duration of these disturbances, but they still decrease the system utility (Bernard et al., 2011).

One special problem of attacks by *Freeriders* is that they create a large amount of bad ratings in the system. In general, it is easy for agents to detect *Freeriders*, because they do not accept any work. When agents detect a *Freerider*, they refuse

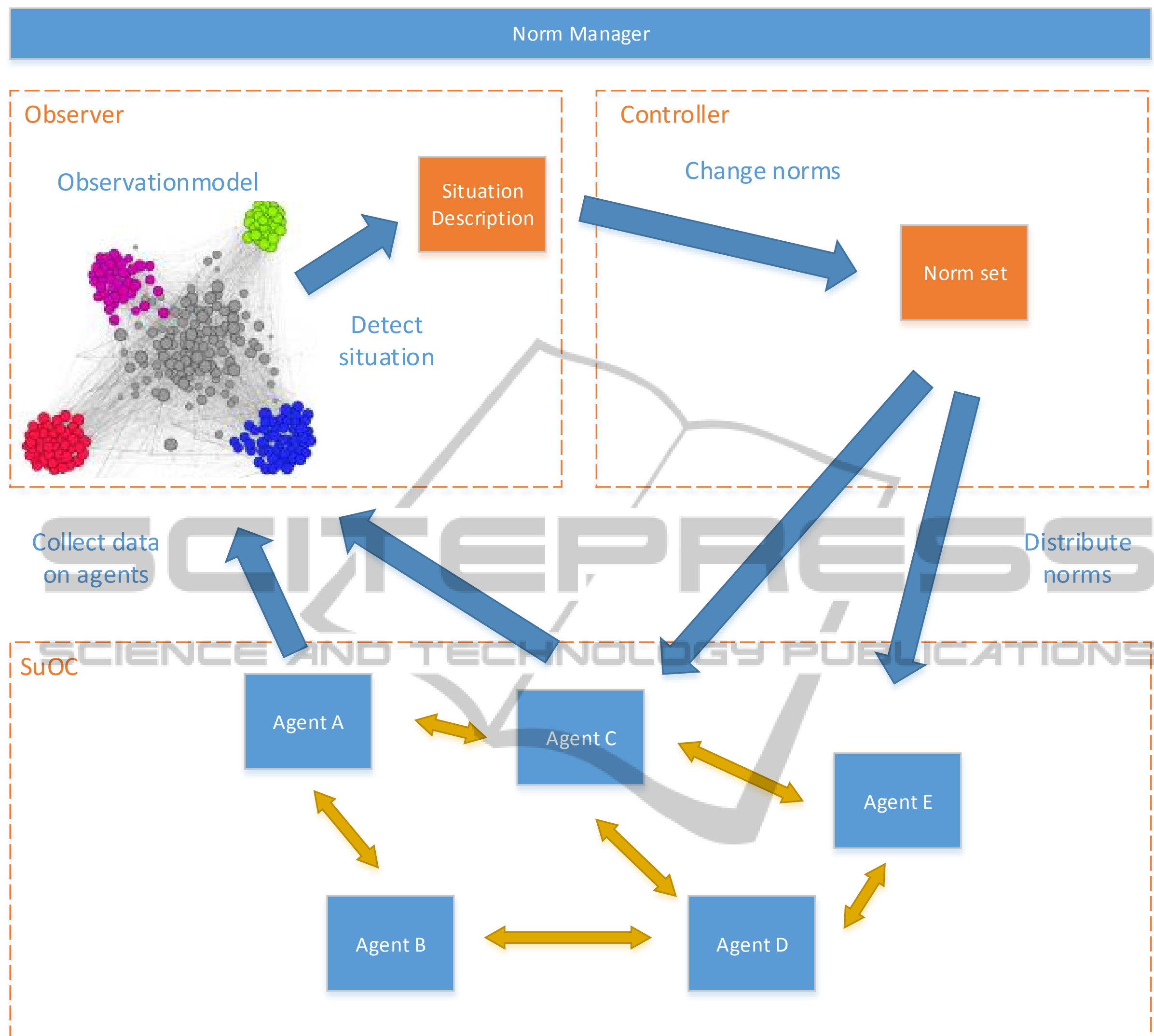


Figure 1: System Overview.

to work for this agent. But the *Freerider* still tries to distribute its work and gives bad ratings to other agents for not cooperating. This leads to a *Trust Breakdown*. Even the reputation of well-behaving agents decreases. It takes usually relatively long time to restore normal system behaviour by isolating all *Freeriders* (Steghöfer et al., 2010).

To prevent problems like *Trust Breakdowns* and increase the robustness of the system, we introduced an agent organisation called *explicit Trusted Communities* (eTCs) (Bernard et al., 2011). When a group of agents notices that they mutually trust each other, they can initialise to form an *eTC*. They elect a leader called the *Trusted Community Manager*, which performs maintenance tasks inside the community. Agents can then reduce security measurements like replication of work units and are able to gain a better speedup. Members of an *eTC* can easier resist attacks, because they can always just cooperate inside the community and ignore the environment.

3 ADAPTIVE CONTROL LOOP FOR NORM MANAGER IN TDG

In our *Trusted Desktop Grid*, different attacks by malevolent agents can occur. We implemented various counter and security measurements to maintain a good utility to well-behaving agents. However, most of these measurements come with some attached costs. Although, we do not benefit from those mechanisms under normal operations. They are essential under attack or at least make recovery from attacks significantly faster.

Additionally, we can configure our reputation system and change the effect of ratings. This may increase or decrease robustness, but it also influences how fast new agents are integrated into the system. Giving larger incentives leads to faster system startup and better speedup when well-behaving agents join the system. However, is gets easier to exploit the system for malevolent agents.

In the *TDG*, a variety of different parameters exist which influence the system behaviour. They must be set before system start. For example, they enable or disable security measures or change the influence of a rating to the reputation system. Some settings result in a better speedup when no attacks occur, but lead to a higher impact on the performance in case of the system being under attack. There is no global optimal value for most of these scenarios. The ideal value or setting depends on the current situation.

To get the best overall performance, we need to change these parameters and settings during runtime according to the current situation. However, we cannot detect global system states like *Trust Breakdown* from a local viewpoint of an agent. It is also not possible to influence agents directly since they are autonomous. There needs to be a higher-level instance which can detect the current system state and a way to indirectly influence all agents in the system.

In Figure 1, we show our concept of the *Norm Manager*, which uses the common Observer-Controller pattern (Tomforde et al., 2011). In the next sections, we describe Observer, Controller and the System under Observation and Control (SuOC).

3.1 Observer

To detect the current system state, the controller monitors work relations of all agents. For this purpose it creates a *work graph* with agents as nodes and edges between agents which have cooperated in the monitored period. The intensity of the cooperation between two agents determines the weight of the edge connecting them. Additionally, the controller creates a *trust graph* with agents as nodes and trust relations as edges. Trust relations between agents can be obtained from the reputation system (Kantert et al., 2013).

Afterwards, we can calculate some common graph metrics for every node. Using statistics, the global system state can be rated. Based on these metrics, we can form clusters and find groups of similar agents. By further classifying these groups, we can achieve an even better understanding about potentially attacks happening. In the end, the observer is able to tell when the system is under attack, tell the type of the attacks and tell how serve the attack is. There will also be an estimation about the accuracy of these informations.

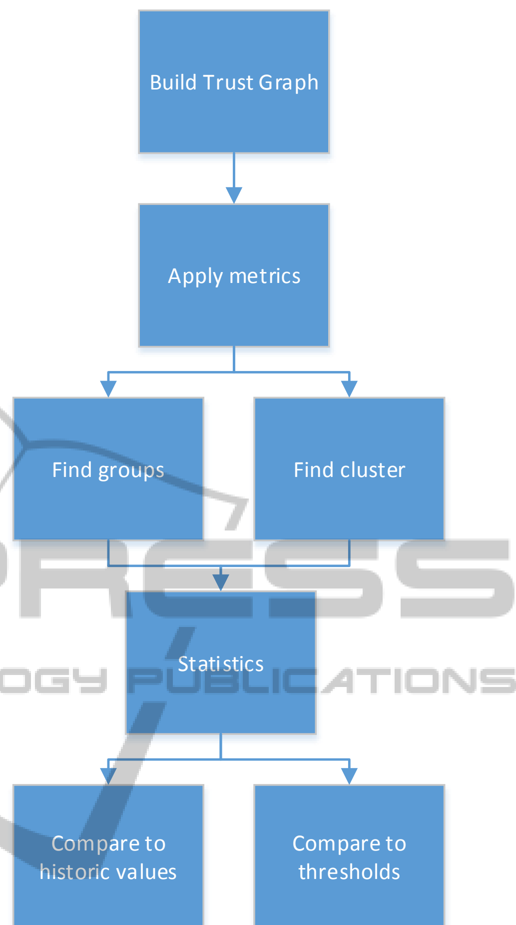


Figure 2: Workflow in observer component.

3.2 Controller

Based on the information obtained by the observer, the controller decides whether the system norms need to be changed. Norms cannot directly influence agents but modify their actions. To be more specific, norms can impose sanctions or offer incentives to actions. To defend against attacks, we can increase sanctions for certain actions under some conditions or we can allow agents to perform security measures, which would lead to sanctions otherwise (Balke et al., 2013).

3.3 SuOC

Agents in the *TDG* need to be able to understand the currently valid norms, which enables them to consider sanctions and incentives in their decision mak-

ing. This allows them to follow short or long-term strategies based on these norms. Since the agents are autonomous and they are free to obey or not, the system still needs to enforce the sanctions and give incentives to agents (Urzică and Cristian, 2013).

The complete control loop implemented by the Observer-Controller component helps to mitigate effects of attacks to the *TDG* and allows a better satisfaction of the system goals. Thereby, it defines an intelligent control mechanism working on system-level. However, if the additional *NM* fails, the system itself is still operational and can continue to run (this refers to the desired OC characteristic of *non-critical complexity* (Schmeck et al., 2010)). When the *NM* is recovered, it can start to optimise the system again.

3.4 Summary

This adaptive control loop allows us to properly configure our system to defend against attacks and perform as well as possible. Most attacks can also be fought off by the agents and the reputation system alone, but this takes more time and leads to worse overall performance. By using security measures only when needed and optimised norms, the system can perform better on average. It can adapt to known and unknown situations, depending on the observations made by the observer. In the next chapter we present our observer, which needs to detect the current state of the system.

4 OBSERVING THE SYSTEM

Since we cannot see the internals or implementation of agents, we need to observe them from the outside. We could monitor interactions between agents, but this may become a bottleneck in larger systems. However, it is easy to monitor the actions indirectly: We can observe the reputation system and use the ratings, which agents give their partners after every interaction.

In Figure 2, we show our general concept of the observer. First, we build a graph of the trust relations between agents. Afterwards, we apply graph metrics to be able to find groups or clusters of similar agents in the next step. We run statistics on every cluster found and compare them to historic or threshold values.

4.1 Trust Graph

To analyse the system state, we build a trust graph. We add all agents as nodes. Afterwards, we fetch the trust relationships between agents from the reputation system and add them as edges to the graph. The weight of the edge represents the amount of trust between the connected agents.

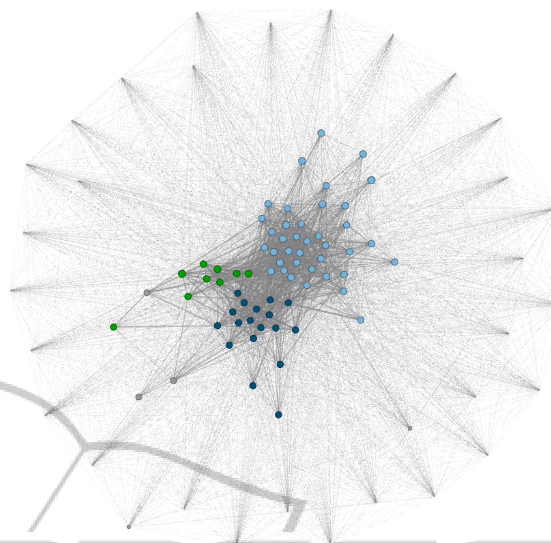


Figure 3: Simple Trust Graph in *TDG* layouted with Force Algorithm.

Figure 3 illustrates an exemplary trust graph from one *TDG* scenario. It has been layouted using a Force Algorithm. In the centre of the graph, agents trust each other and work together. We call this the *core* of the network. All isolated agents are layouted at the border of the graph and have very weak connections. The graph is a complete graph, because there is a trust relation between every pair of agents. However, we omit all edges below a certain weight to improve performance. This also allows us to use algorithms which ignore the edge weight.

4.1.1 Directed

In general, trust has not to be mutual. A *Freerider* F may trust an *Adaptive Agent* A, because A worked for F. But most certainly, A does not trust F, because F refused all work requests of A. We end in a graph with two edges between every pair of nodes. However, after filtering out weak edges, there may be zero or only one edge left.

4.1.2 Undirected

Some algorithms can only work on undirected graphs. Since trust is not mutual, we need to unify both edges. There are different ways to do this:

- Use maximum of weights
- Use sum of weights

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Table 1: Attack size calculated by system size vs attack percentage.

attack %	1%	5%	10%	25%	50%	100%	150%	200%
system size								
50	1	3	5	13	25	50	75	100
100	1	5	10	25	50	100	150	200
250	3	13	25	63	125	250	375	500
500	5	25	50	125	250	500	750	1.000

- Use minimum of weights

We decided to use the minimum of weights, because only two well-behaving agents mutually trust each other. This may also happen during collusion attacks between malevolent agents.

4.2 Metrics

There are plenty of graph metrics in literature. We selected a subset based on (Wasserman, 1994).

4.2.1 Directed

- **Prestige** - This metric counts the incoming edges for a node.
- **Actor Centrality** - This metric counts the outgoing edges for a node.
- **HITS** - Hyperlink-Induced Topic Search algorithm (Kleinberg, 1999). It determines Hubs and Authorities to categorise all nodes in a graph.

4.2.2 Undirected

- **Degree Centrality** - This metric counts all edges for a node
- **Cluster Coefficient** - A metric to measure the degree to which nodes tend to form a cluster.

4.3 Clustering

In this paper, we assume that we can perfectly cluster groups to rate our metrics. Evaluating different clustering algorithms is part of future work. In Figure 3, it can be seen that a Force Algorithm can be used. In our evaluation, we use prior knowledge about agent groups.

5 EVALUATION

In this section, we present some of our results. We show one exemplary experiment for every metric. In our evaluation, we performed several experiments for different system and attack sizes, as shown in Table 1. The system from the shown images consist of 100 agents and an attack percentage of 50%, which means 50 agents enter the system at simulation tick 100k, resulting in a system of 150 agents. We also did separate evaluations for every agent group, which are not shown here.

5.1 Prestige

In Figure 4(a), the Prestige for all agent breeds is shown. It is possible to distinguish between cooperative and non-cooperative agents using this metric: *Adaptive Agents*, *Altruists* and *Cunning Agents* gain a high value. However, on a long term *Cunning Agents* stay below *Adaptive Agents*. *Altruists* grow higher than *Adaptive Agents*. *Egoists* and *Freerider* both stay at a very low value. *Egoists* gain some Prestige at the beginning but loose it again.

5.2 Actor Centrality

Actor Centrality (see Figure 4(b)) behaves similar to Prestige. *Freerider* and *Egoists* stay at a low value. However, they would stayed at zero, if there were no *Altruists* in the system (not shown in the image). *Adaptive Agents* gain a high value. *Cunning Agents* and *Altruists* stay a little lower than *Adaptive Agents*.

5.3 Degree Centrality

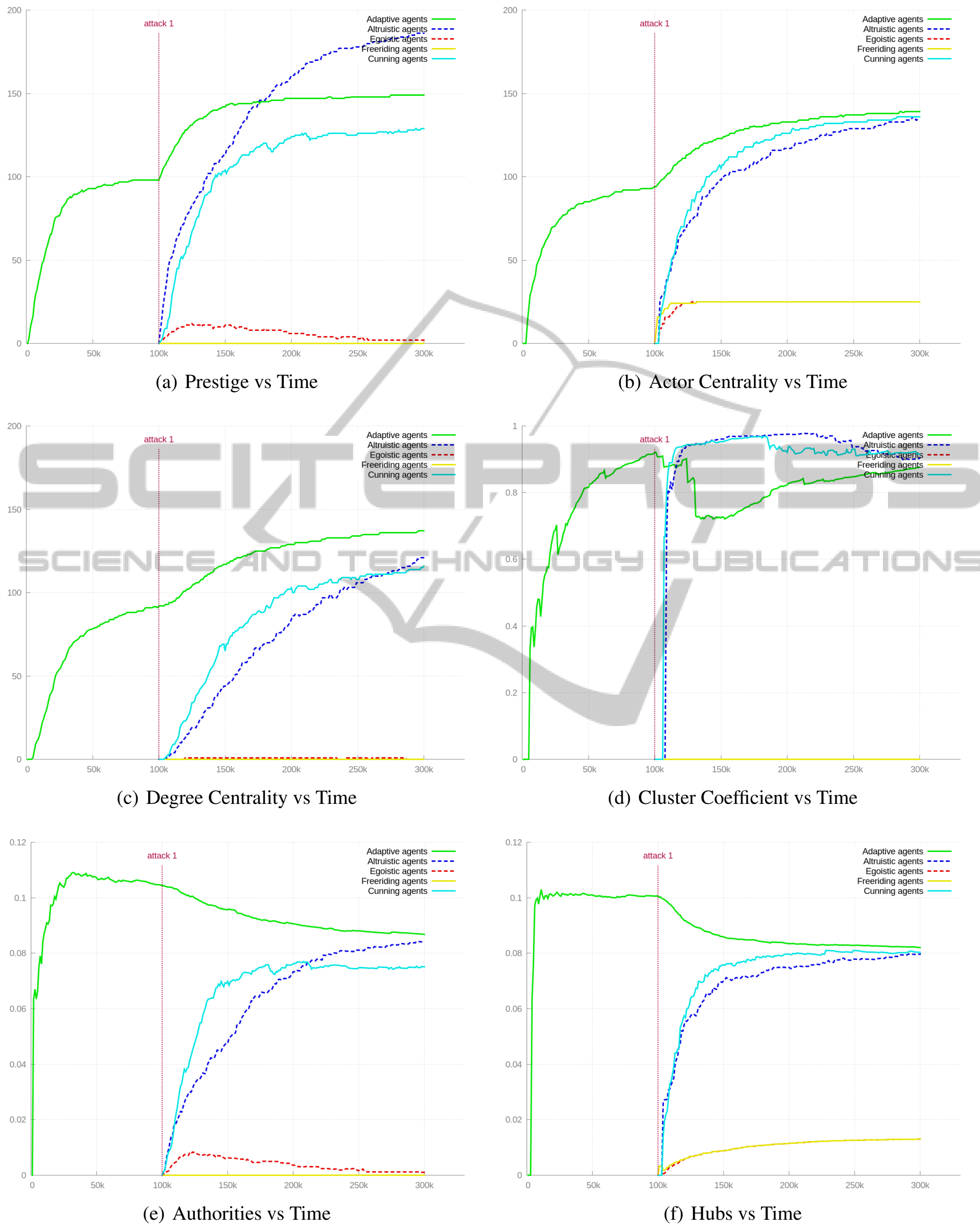
In Figure 4(c), we show the Degree Centrality of the agent groups. This metric can be used to clearly identify non-cooperative agents (*Freeriders* and *Egoists*). *Adaptive Agents* gain a very high value. *Cunning Agents* stay significantly below in a long term. *Altruists* are between the previous two.

5.4 Cluster Coefficient

Cunning Agents and *Altruists* gain a very high Cluster Coefficient value (see Figure 4(d)). *Adaptive Agents* stay below. Non-cooperative agents have a value of zero again.

5.5 Authorities

Authorities calculated by the HITS Algorithm are a good metric to distinguish *Adaptive Agents* and *Altru-*



(a) Prestige vs Time

(b) Actor Centrality vs Time

(c) Degree Centrality vs Time

(d) Cluster Coefficient vs Time

(e) Authorities vs Time

(f) Hubs vs Time

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

Figure 4: Combined simulation results.

ists from *Cunning Agents*. *Egoists* gain a little more Authority at the beginning but start to loose it afterwards. *Freerider* stay at zero all the time (see Figure 4(e)).

5.6 Hubs

Hubs only allow us to categorise agents into a cooperative and a non-cooperative group. Similar to Ac-

tor Centrality, *Egoists* and *Freerider* only get a value higher than zero if *Altruists* are in the system. All cooperative agents reach a high value and stay very close (see Figure 4(f)).

6 CONCLUSION AND FUTURE WORK

Using our metrics, we can reliably identify different types in our system. Non-cooperative agents such as *Freerider* and *Egoists* are very easy to detect. *Altruists* make detection of other agent types harder, because they work for everybody. However, we can still detect all other groups. *Cunning Agents* are also hard to detect, because they behave similar to *Adaptive Agents*. Still, we can identify them on the long-run using Prestige and Authorities.

Our next step is to find good clustering algorithms to find groups of similar agents. Force algorithms as seen in Figure 3 look promising. However, by using multiple clustering metrics, the results should be even better.

ACKNOWLEDGEMENTS

This research is funded by the research unit "OC-Trust" (FOR 1085) of the German Research Foundation (DFG).

REFERENCES

Balke, T. et al. (2013). Norms in MAS: Definitions and Related Concepts. In *Normative Multi-Agent Systems*, volume 4 of *Dagstuhl Follow-Ups*, pages 1–31. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Bernard, Y., Klejnowski, L., Cakar, E., Hahner, J., and Müller-Schloer, C. (2011). Efficiency and Robustness Using Trusted Communities in a Trusted Desktop Grid. In *Proc. of SASO Workshops (SASOW)*.

Bernard, Y., Klejnowski, L., Hähner, J., and Müller-Schloer, C. (2010). Towards Trust in Desktop Grid Systems. *Proc. of IEEE CCGrid*, pages 637–642.

Kantert, J., Bernard, Y., Klejnowski, L., and Müller-Schloer, C. (2013). Interactive Graph View of Explicit Trusted Communities in an Open Trusted Desktop Grid System. In *Proc. of SASO Workshops (SASOW)*, pages 13–14.

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632.

Klejnowski, L., Bernard, Y., Hähner, J., and Müller-Schloer, C. (2010). An Architecture for Trust-Adaptive Agents. In *Proc. of SASO Workshops (SASOW)*, pages 178–183.

Müller-Schloer, C. (2004). Organic Computing: On the Feasibility of Controlled Emergence. In *Proc. of CODES and ISSS'04*, pages 2–5. ACM Press.

Schmeck, H., Müller-Schloer, C., Çakar, E., Mnif, M., and Richter, U. (2010). Adaptivity and Self-organization in Organic Computing Systems. *ACM Trans. on Aut. and Adap. Sys.*, 5(3):1–32.

Steghöfer, J.-P., Kiefhaber, R., Leichtenstern, K., Bernard, Y., Klejnowski, L., Reif, W., Ungerer, T., André, E., Hähner, J., and Müller-Schloer, C. (2010). Trustworthy Organic Computing Systems: Challenges and Perspectives. In *Proc. of ATC 2010*. Springer.

Tomforde, S., Prothmann, H., Branke, J., Hähner, J., Mnif, M., Müller-Schloer, C., Richter, U., and Schmeck, H. (2011). Observation and Control of Organic Systems. In *Organic Computing - A Paradigm Shift for Complex Systems*, pages 325 – 338. Birkhäuser Verlag.

Urzică, A. and Cristian, G. (2013). Policy-Based Instantiation of Norms in MAS. In Fortino, G., Badica, C., Malgeri, M., and Unland, R., editors, *Intelligent Distributed Computing VI*, volume 446 of *Studies in Computational Intelligence*, pages 287–296. Springer Berlin Heidelberg.

Wasserman, S. (1994). *Social network analysis: Methods and applications*, volume 8. Cambridge university press.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS