Defending autonomous agents against attacks in multi-agent systems using norms

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

Angaben zur Veröffentlichung / Publication details:

Kantert, Jan, Sarah Edenhofer, Sven Tomforde, Jörg Hähner, and Christian Müller-Schloer. 2015. “Defending autonomous agents against attacks in multi-agent systems using norms.” In *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART-2015), Lisabon, Portugal, 10-12 January 2015*, edited by Stephane Loiseau, Joaquim Filipe, Beatrice Duval, and Jaap van den Herik, 149–56. Setúbal: SciTePress. https://doi.org/10.5220/0005202101490156.

Defending Autonomous Agents Against Attacks in Multi-Agent Systems Using Norms

Jan Kantert[1], Sarah Edenhofer[2], Sven Tomforde[2], Jörg Hähner[2] and Christian Müller-Schloer[1]
[1]*Institute of Systems Engineering, 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: The Trusted Desktop Grid (TDG) is a self-organised, agent-based organisation, where agents perform computational tasks for others to increase their performance. In order to establish a fair distribution and provide counter-measures against egoistic or malicious elements, technical trust is used. A fully self-organised approach can run into disturbed states such as a trust breakdown of the system that lead to unsatisfying system performance although the majority of participants is still behaving well. We previously introduced an additional system-wide control loop to detect and alleviate disturbed situations. Therefore, we describe an Observer/Controller loop at system level that monitors the system status and intervenes if necessary. This paper focuses on the controller part which instantiates norms as reaction to observed suspicious situations. We demonstrate the benefit of our approach within a Repast-based simulation of the TDG. Therein, the impact of disturbances on the system performance is decreased significantly and the time to recover is shortened.

1 INTRODUCTION

Self-organised volunteer computing shares computing resources of a potentially large set of participants in a distributed manner. Agents can utilise resources of others to get their jobs computed as fast as possible and thereby decrease their processing time. Typically, such a system is realised as open system – meaning that everyone is free to join if following the basic protocol.

Since openness attracts egoistic or even malicious elements, counter-measures are needed to isolate these unwanted elements and provide a performant platform for normal and benevolent users. In previous work, we demonstrated that technical trust is a perfect instrument to achieve this goal – leading to a Trusted Desktop Grid (TDG) (Bernard et al., 2010). However, a fully self-organised solution has drawbacks that become visible in disturbed situations. For instance, an orchestrated joining of malicious elements can cause a trust breakdown that also affects the trust relations of normal agents (Klejnowski, 2014).

In this paper, we present a novel concept to handle these disturbed situations – once they were detected. Since the effects are not visible for isolated agents, a system-wide management has to be established. Previously, we proposed to utilise the Observer/Controller concept as introduced by the Organic Computing initiative (Tomforde et al., 2011) as basis for such a system-wide control and presented an approach to detect such situations. Since agents are autonomous and we cannot control their implementations, a guided self-organisation is aimed for. This is achieved by publishing and controlling norms that have to be followed by agents at runtime. As a result, we establish a system-wide control loop that issues norms in disturbed situations and thereby manages the overall behaviour.

The remainder of this paper is organised as follows: Section 2 describes the TDG as application scenario with a special focus on the agents' goals, the system goal, and the trust mechanism. It also explains the trust breakdown that serves as motivation of this work in more detail. Afterwards, Section 3 introduces the norm-based system control with its observer and controller components as well as the system under observation and control. Section 5 presents our novel concepts and details the approach. In addition, Section 4 puts the approach into the context of related and preliminary work. This is followed by a simulation-based evaluation in Section 6 that demonstrates the benefit of our approach with respect to decreasing the impact of disturbances on the system performance and the time to recover. Section 7 summarises the paper and gives an outlook to current and future work.

2 APPLICATION SCENARIO

In this work, we investigate and improve open distributed systems. To analyse such systems, we model nodes as agents and run a multi-agent system in simulation. Our application scenario is an open distributed Desktop Grid System. Every agent works for a user and periodically gets a job, which contains multiple parallelisable work units. It aims to get all work units done as fast as possible and it accomplishes this by requesting other agents to work for it. Since we consider an open system, agents are autonomous and can join or leave at any time.

2.1 Agent Goal

The performance is measured by the speedup σ. In Equation (1), τ_{self} is the time it would take an agent to calculate a job with multiple work units without any cooperation. $\tau_{\text{distributed}}$ represents the time it took to calculate all work units of one job with cooperation of other workers including all communication times. Speedup can only be determined after the result of the last work unit has been returned.

$$\sigma := \frac{\tau_{\text{self}}}{\tau_{\text{distributed}}} \tag{1}$$

If no cooperation partners can be found, agents need to calculate their own work units and achieve a speedup value equal to one (i.e. no speedup at all). 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 themselves. We do not control the agent implementation, so they may be uncooperative or even malicious. In contrast to other work from state of the art, we do not assume the benevolence of the agents (Wang and Vassileva, 2004). Such an open system is vulnerable to different kinds of attacks. For instance, a *Freerider* can simply refuse to work for other agents and gain an advantage at the expense of cooperative agents.

2.2 System Goal

The global goal is to enable agents, which act according to the system rules, to achieve the best possible speedup. We measure the global achievement og the goal either by the average speedup of the well-behaving agents or by the number of cooperation (4) combined with the average submit-to-work-ratio of all agents (5). $\text{submit}(A_i, A_j)$ represents the number of work units, which agent A_i successfully calculated for agent A_j. Accordingly, $\text{work}(A_i, A_j)$ counts the work units A_i submitted to A_j. $\text{work}(A_i)$ is the number of work units A_i submitted to all other agents (3). Accordingly, $\text{submit}(A_j)$ shows the count of work units an agent calculated for other agents (2).

$$\text{submit}(A_i) := \sum_{j=1, j\neq i}^{n} \text{submit}(A_i, A_j) \tag{2}$$

$$\text{work}(A_i) := \sum_{j=1, j\neq i}^{n} \text{work}(A_i, A_j) \tag{3}$$

$$\text{cooperation} := \sum_{i=1}^{n} \text{work}(A_i) \tag{4}$$

$$\text{fairness} := \sum_{i=1}^{n} \frac{\min\left(\frac{\text{submit}(A_i)}{\text{work}(A_i)}, \frac{\text{work}(A_i)}{\text{submit}(A_i)}\right)}{n} \tag{5}$$

The system solves a distributed resource allocation problem. Since work units can be calculated faster when agents cooperate, we reward and maximise cooperation. Additionally, a high fairness value ensures equal resource distribution (cf. (Jarn, 1996; Demers et al., 1989; Bennett and Zhang, 1996)).

2.3 Trust Metric

To overcome the problems of an open system, where no particular behaviour can be assumed, we introduced a trust metric. Agents receive ratings for all their actions from their particular interaction partners. This allows us to make an estimation about the general behaviour of an agent based on its previous actions. In our system, agents receive 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 (Bernard et al., 2010) (Klejnowski, 2014).

An agent has multiple ratings with a value between -1 and 1 (6). The amount of ratings k is limited to implement oblivion. The global average over all ratings for a single agent is called the reputation ρ_i (7).

$$\text{ratings}_i \in [-1, 1]^k \tag{6}$$

$$\rho_i := \sum_{j=1}^{k} \frac{\text{ratings}_{ij}}{k} \tag{7}$$

2.4 Agent Types

We consider the following agent types in our system:

- *Adaptive Agents* - These agents are cooperative. They work for other agents who earned good reputation in the system. How high the reputation value 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 the 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.

We simulate an attack by adding new malicious agents to the system at start-up 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. Thus, we try to minimise the impact and duration of these disturbances, but they still decrease the system utility (Klejnowski, 2014).

2.5 Trust Breakdown

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 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*, during which even the reputation of well-behaving agents decreases. As a result, well-behaving agents can no longer differentiate between malicious and other well-behaving agents allowing *Freeriders* to exploit all agents. In total, the average *Speedup* decreases for all agents. Usually, a significant period is needed to restore normal system behaviour by isolating all *Freeriders* (Steghöfer et al., 2010).

3 NORM-BASED SYSTEM CONTROL

In our Trusted Desktop Grid (Bernard et al., 2010) (TDG), different attacks by malevolent agents can

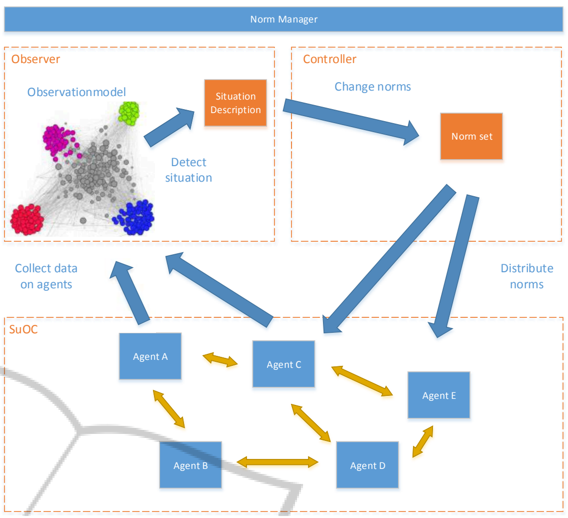

Figure 1: System overview.

occur (for instance, the aforementioned *Trust Breakdown*). We implemented various counter and security measures to maintain a good utility for well-behaving agents. However, most of these measures appear with some attached costs. Although we do not benefit from those mechanisms under normal operations, they are essential under attack or at least lead to a significantly faster recovery from attacks (Kantert et al., 2014b).

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 a faster system start-up and a better speedup when well-behaving agents join the system. However, it also 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 obtain the best overall performance, we need to change these parameters and settings during runtime according to the current situation. For this, a proper system state detection is a prerequisite. 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. Therefore, there needs to be a higher-level instance which can detect the current system state with an option to indirectly influence all agents in the system. We call this higher-level instance Norm Manager (NM).

In Fig. 1, we show our concept of the Norm Manager (NM), which uses the common Observer/Controller pattern (Tomforde et al., 2011). To detect the current system state, the observer 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 observer creates a *trust graph* with agents as nodes and trust relations as edges. Trust relations between agents can be obtained from the reputation system.

Since we cannot see the internals or implementations of agents, we need to observe them from the outside. We could monitor interactions between agents, but this may lead to 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. When we collect those ratings, we can build a trust-graph. Multiple ratings will be merged using an arithmetic mean.

We evaluated the effect of attacks on different metrics. Based on those measurements, we clustered and classified groups of agents based on their behaviour (Kantert et al., 2014b). Using this information, we create a situation description which is can be used by the controller to guide the system. In this paper, we focus on the controller side.

4 RELATED AND PREVIOUS WORK

4.1 Desktop Grid and Trust

Our application scenario is a Desktop Grid System. These systems are used to share resources between multiple administrative authorities. The ShareGrid Project in Northern Italy is an example for a peer-to-peer-based system (Anglano et al., 2008). A second approach is the Organic Grid, which is peer-to-peer-based with decentralised scheduling (Chakravarti et al., 2004). Compared to our system, these approaches assume that there are no malicious parties involved and every node behaves well. Another implementation with a central tracker is the Berkeley Open Infrastructure for Network Computing Project (BOINC) (Anderson and Fedak, 2006).

We model our grid nodes as agents. Agents have a local goal which differs from the global system goal (Rosenschein and Zlotkin, 1994). We consider agents as black boxes. This means, we cannot observe the internal state. Thus, their actions and behaviour cannot be predicted (Hewitt, 1991). Our TDG supports Bag-of-Tasks applications (Anglano et al., 2006).

A classification of Desktop Grid Systems can be found in Choi et al. (2007). A taxonomy can be found in Choi et al. (2008). It is highlighted that there has to be some mechanism to detect failures and malicious behaviour in large-scale systems. Nodes cannot be expected to be unselfish and well-behaving.

In contrast to other state-of-the-art works, we do not assume the benevolence of the agents (Wang and Vassileva, 2004). To cope with this information uncertainty, we introduced a trust metric. A general overview about trust in Multi-Agent Systems can be found in Castelfranchi and Falcone (2010). Another implementation of trust in a Desktop Grid System was evaluated in Domingues et al. (2007).

4.2 Norms

Explicit norms are similar to laws and can be expressed in deontic logic and argumentation. Individuals can reason based on these norms. Since there are multiple actions available, they may use additional factors or preferences (Sartor, 2005). Other approaches use Defeasible logic (DL) to efficiently model (Nute, 1994) and reason (Nute, 1988) about norms. They separate facts and rules, which can be strict rules, defeasible rules and exceptions from defeasible rules (called defeaters). To resolve conflicts between two rules reasoning about the same action, priorities can be specified (Nute, 2003). All reasoning can be done in linear time and is stable even when norms are not consistent (Billington, 1993).

We base our norm format on Urzică and Gratie (2013). The authors developed a model for representing norms using context-aware policies with sanctions. They take reputation into consideration when making decisions based on norms. We use a conditional norm structure as described in Balke et al. (2013). Most of our norms can be characterised as “prescriptions” based on von Wright (1963), because they regulate actions. Our norms are generated by a central elected component representing all agents which classifies them as a “r-norm” according to Tuomela and Bonnevier-Tuomela (1995). By using norms, our agents can reach agreements (Tuomela and Bonnevier-Tuomela, 1995) and express commitments (Singh, 1999). However, the agents can still violate commitments and risk a sanction. Thereby, the agents stay autonomous. Hollander and Wu (2011) present a norm lifecycle including norm creation, enforcement, and adaption.

4.3 Normative Multi-Agent Systems

This work is part of wider research in the area of norms in multi-agent systems. However, we focus more on improving system performance by using norms than researching the characteristics of norms (Singh, 1999). Our scenario is similar to management of common pool resources. According to game theory, this leads to a "tragedy of the commons" (Hardin, 1968). However, Ostrom (1990) observed cases were this did not happen. She presented eight design principles for successful self-management of decentralised institutions. Pitt et al. (2011) adapted these to normative Multi-Agent Systems.

Normative Multi-Agent Systems are used in multiple fields: e.g. Governatori and Rotolo (2008) focus on so-called policy-based intentions in the domain of business process design. Agents plan consecutive actions based on obligations, intentions, beliefs, and desires. Based on DL, social agents can reason about norms and intentions.

Artikis and Pitt (2009) present a generic approach to form organisations using norms. They assign a role to agents in a normative system. This system defines a goal, a process to reach the goal, required skills, and policies constraining the process. Agents directly or indirectly commit for certain actions using a predefined protocol. Agents may join or form an organisation with additional rules.

The normchange definition describes attributes, which are required for Normative Multi-Agent Systems (Boella et al., 2009). Ten guidelines for implementation of norms to MAS are given. We follow those rules in our system.

When norms are involved, agents need to make decisions based on these norms. Conte et al. (1999) argue that agents have to be able to violate norms to maintain autonomy. However, the utility of certain actions may be lower due to sanctions. We did some previous work, presented in (Kantert et al., 2014a).

Normative Multi-Agent Systems can be classified according to five categories (Savarimuthu and Cranefield, 2011): Norm creation, norm identification, norm spreading, norm enforcement, and network topology. We use a leadership mechanism for norm creation and norm spreading. For norm identification we use data mining and machine learning. Our network topology is static. For norm enforcement, we use sanctioning and reputation.

5 APPROACH

In previous work, we focused on detecting attacks during runtime in the observer part of a novel higher level Norm Manager (Kantert et al., 2014b). Using this knowledge, we want the controller part of our NM to change norms in the TDG (see Figure 1).

In this paper, we focus on the controller component. The controller is responsible for guiding the overall system behaviour by applying norms. Such a norm contains a rule and a sanction (Urzică and Gratie, 2013). Agents are still autonomous and can violate norms with the risk of sanctioning.

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 lead to sanctioning otherwise (Balke et al., 2013). Certainly, changed sanctions of incentives only apply to actions which were taken after the change.

To counter attacks of malicious agents the controller utilizes various counter measures: change or create norms; issue incentives or add sanctions. In Table 1, the default norms of our TDG are shown. Agents get positive reputation when they finished the work for other agents. If they reject work they get a bad rating, unless the reputation of the requesting agent A_s is below α.

When an agent enters the system it gets an initial reputation Ψ. To facilitate integration into the system, Ψ is greater 0. However, malicious agents can use this initial trust to exploit other agents. Especially in sibling attacks, where agents have multiple identities, this becomes a big issue. Unfortunately, $\Psi > 0$ is also needed to efficiently integrate well-behaving agents.

Fortunately, the observer can detect such attacks, so, the controller can react based on that knowledge. In our approach, the controller changes α in Norm 1 to a value $\alpha > \Psi$ (see Table 1).

We expect a decrease in the impact of attacks by *Freeriders* and *Egoists* since they will no longer be able to cause *Trust Breakdown* with their initial reputation. This effect can be measured by the time be-

Table 1: Simplified norms for Worker Component in Trusted Desktop Grid.

	Evaluator	Action	Context	Sanction/Incentive
1	Worker	$RejectJob(A_w, A_s)$	$T(A_w, A_s) > \alpha$	$T(A_s, A_w)$ -= $Penalty_{reject}$
			$T(A_w, A_s) \leq \alpha$	-
2	Worker	$ReturnJob(A_w, A_s)$		$T(A_s, A_w)$ += $Incentive_{workDone}$

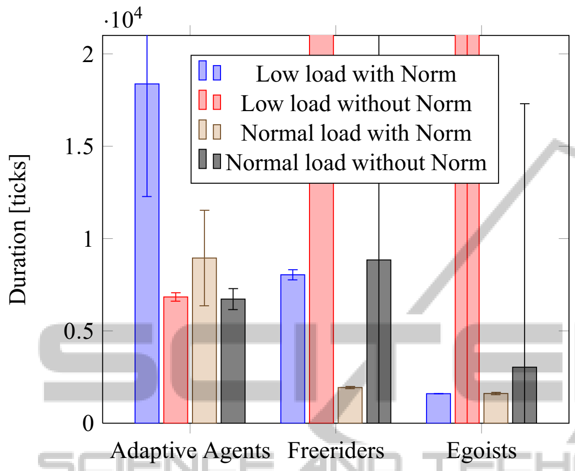

Figure 2: Duration of isolation/integration per agent group.

tween attack start and the point where all attackers are isolated. We want to minimize this duration of isolation. As our approach also affects well-behaving agents, we also measure the increase of duration to integrate them into the system.

6 EVALUATION AND DISCUSSION

We consider attacks by *Freeriders* and *Egoists* and evaluate both attacks by adding each 100 attacker agents to a system of 200 *Adaptive Agents*. To measure the effect on well-behaving agents we repeat the experiment with 100 *Adaptive Agents* entering the system. The described norm change (see Section 5) is performed at the beginning of the attack. Additionally, we run a reference experiment without norm change for all agent types. Since isolation and integration of agents is slower during low load situations, we added this as a scenario. Every experiment is repeated one-thousand times - resulting in 12.000 experiments.

After the attack starts at τ_{start}, we periodically calculate the speedup σ (defined in Equation (1)) for the attacking agents. $\tau_{\text{end,isolation}}$ is defined to be the smallest value with $\tau_{\text{end}} > \tau_{\text{start}} \wedge \sigma \leq 1$ (8). The duration of isolation $\delta_{\text{isolation}}$ is then determined as the difference of τ_{end} and τ_{start} (9).

$$\tau_{\text{end,isolation}} := \min\{\tau : \tau > \tau_{\text{start}} \wedge \sigma_\tau \leq 1\} \quad (8)$$

$$\delta_{\text{isolation}} := \tau_{\text{end,isolation}} - \tau_{\text{start}} \quad (9)$$

For *Adaptive Agents*, we similarly calculate the duration of integration $\delta_{\text{integration}}$ (11). In a reference experiment without norm change, we determine the final speedup after integration σ_{ref}. $\tau_{\text{end,integration}}$ is then defined to be the first time after attack where $\sigma \geq \sigma_{\text{ref}}$ (10).

$$\tau_{\text{end,integration}} := \min\{\tau : \tau > \tau_{\text{start}} \wedge \sigma_\tau \geq \sigma_{\text{ref}}\} \quad (10)$$

$$\delta_{\text{integration}} := \tau_{\text{end,integration}} - \tau_{\text{start}} \quad (11)$$

In Figure 2, we present our results for three agent types. For *Freeriders* and *Egoists*, the graph shows $\delta_{\text{isolation}}$. In contrast, for *Adaptive Agents*, it illustrates $\delta_{\text{integration}}$. Full results with standard deviation are listed in Table 2.

The results show that isolation of malicious agents greatly improved when norms were changed, especially, in low load situations. For *Freeriders* the duration decreased by 78% under normal load. Under low load, *Freeriders* were not fully isolated before. However, this changed with our approach: The system did properly isolate the attackers in all experiments. Since isolation did not work in the reference case we limited the length of that experiment. Therefore, the value for low load without norms in Table 2 has no variance at all and the relative gain cannot be calculated.

Our approach is very effective when dealing with *Egoists*. With changed norms during attack they get isolated after calculating their first job (duration of a job is 1600 ticks). Without the change they did not get isolated in most cases under low load and it took about twice as long under normal load.

However, well-behaving agents are also affected by the norm change: *Adaptive Agents* need 33% longer under normal load and 169% longer under low load. Integration still worked in all experiments and can be considered stable.

Our results show that changing norms reduces the impact of attacks by *Freeriders* and *Egoists*. However, this change cannot become the default because it also affects the integration of well-behaving agents. Nevertheless, by using our NM we can change norms when the observer detects an attack by *Freeriders* or *Egoists*.

Critical to the success of this method is fast detection of such attacks. After isolation of the attackers

Table 2: Duration of isolation/integration per agent group.

Agent	Low Load without Norm	Normal Load without Norm	Low Load with Norm	Normal Load with Norm
Adaptive Agents	6837.1±228.06	6722.1±568.06	18375.3±6098.97	8945.7±2585.81
Freerider	145000±0	8841.6±17597.08	8037.9±275.77	1930.8±57.37
Egoists	41178.4±64102.20	3034.1±14268.11	1600±0	1609.3±62.47

the norm changes can be reversed since isolation of those two groups is permanent remains the TDG. Isolation is performed using Trust and Reputation mechanism of the TDG. We chose this approach to keep maximal autonomy for the agents.

Our implementation is currently limited to systems below some 50k agents since our graph metrics used in the observer to create the situation description have quadratic runtime. However, we plan to enlarge this limit by distributing the NM and using algorithms with better runtime.

Currently, the NM poses a single point of failure for norm creation and changes. However, norms are not crucial to the system since the TDG can run without them. If this happens agents can simple elect a new NM. This can be referred to as *non-critical complexity* (Schmeck et al., 2010). Nevertheless, we plan to add some redundancy here and want to implement a more distributed approach with multiple NMs.

7 CONCLUSION AND FUTURE WORK

This paper presented a novel concept to manage an open, self-organised, and trust-based agent community by utilising norms. As application scenario, we investigated a Trusted Desktop Grid system, where agents can share their computation resources in order to increase their performance. We introduced a system-wide control loop that follows Organic Computing's generic Observer/Controller concept; this loop is responsible for detecting and alleviating disturbed system states. Thereby, this paper focused on the controller part which instantiates norms as reaction to observed suspicious situations. Our simulation-based evaluation demonstrates that this system-wide control loop is able to counter disturbed situations such as the trust breakdown. Using our approach, the impact of the disturbances can be decreased significantly and the time to recover in the case of failures is shortened dramatically.

Current and future work deals with questions closely related to issuing norms. For instance, an automatic generation of new norms that can be instantiated to counter previously unknown situations is of interest. This will be coupled with an online learning component that estimates the achieved impact of the applied norms and, therefore, improves the performance of this control loop at runtime. Finally, a combination with a self-organised surveillance of norm compliance in combination with a self-managed sanctioning mechanism will be applied to the system to complete the control loop and combine distributed and centralised elements.

REFERENCES

Anderson, D. and Fedak, G. (2006). The Computational and Storage Potential of Volunteer Computing. In *Proc. of CCGRID 2006*, pages 73–80. IEEE.

Anglano, C., Brevik, J., Canonico, M., Nurmi, D., and Wolski, R. (2006). Fault-aware Scheduling for Bag-of-Tasks Applications on Desktop Grids. In *Proc. of GRID 2006*, pages 56–63. IEEE.

Anglano, C., Canonico, M., Guazzone, M., Botta, M., Rabellino, S., Arena, S., and Girardi, G. (2008). Peer-to-Peer Desktop Grids in the Real World: The ShareGrid Project. *Proc. of CCGrid 2008*, pages 609–614.

Artikis, A. and Pitt, J. (2009). Specifying Open Agent Systems: A Survey. In Artikis, A., Picard, G., and Vercouter, L., editors, *Engineering Societies in the Agents World IX*, volume 5485 of *LNCS*, pages 29–45. Springer.

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.

Bennett, J. and Zhang, H. (1996). WF2Q: Worst-case Fair Weighted Fair Queueing. In *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, volume 1, pages 120–128.

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

Billington, D. (1993). Defeasible Logic is Stable. *Journal of Logic and Computation*, 3(4):379–400.

Boella, G., Pigozzi, G., and van der Torre, L. (2009). Normative Systems in Computer Science - Ten Guidelines for Normative Multiagent Systems. In Boella, G., Noriega, P., Pigozzi, G., and Verhagen, H., editors, *Normative Multi-Agent Systems*, number 09121 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

Castelfranchi, C. and Falcone, R. (2010). *Trust Theory: A Socio-Cognitive and Computational Model*, volume 18. John Wiley & Sons.

Chakravarti, A. J., Baumgartner, G., and Lauria, M. (2004). Application-Specific Scheduling for the Organic Grid. In *Proc. of GRID 2004 Workshops*, pages 146–155, Washington, DC, USA. IEEE.

Choi, S., Buyya, R., Kim, H., and Byun, E. (2008). A Taxonomy of Desktop Grids and its Mapping to State of the Art Systems. Technical report, Grid Computing and Dist. Sys. Laboratory, The University of Melbourne.

Choi, S., Kim, H., Byun, E., Baik, M., Kim, S., Park, C., and Hwang, C. (2007). Characterizing and Classifying Desktop Grid. In *Proc. of CCGRID 2007*, pages 743–748.

Conte, R., Castelfranchi, C., and Dignum, F. (1999). Autonomous Norm Acceptance. In Müller, J., Rao, A., and Singh, M., editors, *Intelligent Agents V: Agents Theories, Architectures, and Languages*, volume 1555 of *LNCS*, pages 99–112. Springer.

Demers, A., Keshav, S., and Shenker, S. (1989). Analysis and Simulation of a Fair Queueing Algorithm. In *Symposium Proceedings on Communications Architectures & Protocols*, SIGCOMM '89, pages 1–12, New York, NY, USA. ACM.

Domingues, P., Sousa, B., and Moura Silva, L. (2007). Sabotage-tolerance and Trustmanagement in Desktop Grid Computing. In *Future Gener. Comput. Syst. 23, 7*, pages 904–912.

Governatori, G. and Rotolo, A. (2008). BIO Logical Agents: Norms, Beliefs, Intentions in Defeasible Logic. *Autonomous Agents and Multi-Agent Systems*, 17(1):36–69.

Hardin, G. (1968). The Tragedy of the Commons. *Science*, 162(3859):1243–1248.

Hewitt, C. (1991). Open Information Systems Semantics for Distributed Artificial Intelligence. *Artificial intelligence*, 47(1):79–106.

Hollander, C. D. and Wu, A. S. (2011). The Current State of Normative Agent-Based Systems. *Journal of Artificial Societies and Social Simulation*, 14(2):6.

Jarn, R. e. a. (1996). Fairness, Call Establishment Latency and Other Performance Metrics. *ATM-Forum*, 96(1173).

Kantert, J., Bernard, Y., Klejnowski, L., and Müller-Schloer, C. (2014a). Estimation of Reward and Decision Making for Trust-Adaptive Agents in Normative Environments. In Maehle, E., Römer, K., Karl, W., and Tovar, E., editors, *Architecture of Computing Systems – ARCS 2014*, volume 8350 of *LNCS*, pages 49–59. Springer.

Kantert, J., Scharf, H., Edenhofer, S., Tomforde, S., Hähner, J., and Müller-Schloer, C. (2014b). A Graph Analysis Approach to Detect Attacks in Multi-Agent-Systems at Runtime. accepted for publication at SASO 2014.

Klejnowski, L. (2014). *Trusted community : a novel multiagent organisation for open distributed systems*. PhD thesis, Leibniz Universität Hannover.

Nute, D. (1988). Defeasible Reasoning: A Philosophical Analysis in Prolog. In *Aspects of Artificial Intelligence*, pages 251–288. Springer.

Nute, D. (1994). Defeasible Logic. *Handbook of Logic in Artificial Intelligence and Logic Programming*, 3:353–395.

Nute, D. (2003). Defeasible Logic. In *Proceedings of the Applications of Prolog 14th International Conference on Web Knowledge Management and Decision Support*, INAP'01, pages 151–169, Berlin, Heidelberg. Springer.

Ostrom, E. (1990). *Governing the Commons: The Evolution of Institutions for Collective Action*. Cambridge university press.

Pitt, J., Schaumeier, J., and Artikis, A. (2011). The Axiomatisation of Socio-Economic Principles for Self-Organising Systems. In *Self-Adaptive and Self-Organizing Systems (SASO), 2011 Fifth IEEE International Conference on*, pages 138–147.

Rosenschein, J. S. and Zlotkin, G. (1994). *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press, Cambridge.

Sartor, G. (2005). *Legal Reasoning: A Cognitive Approach to Law*. Springer.

Savarimuthu, B. T. R. and Cranefield, S. (2011). Norm Creation, Spreading and Emergence: A Survey of Simulation Models of Norms in Multi-Agent Systems. *Multiagent and Grid Systems*, 7(1):21–54.

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.

Singh, M. P. (1999). An Ontology for Commitments in Multiagent Systems. *Artificial Intelligence and Law*, 7(1):97–113.

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.

Tuomela, R. and Bonnevier-Tuomela, M. (1995). Norms and Agreements. *E. J. of Law, Philosophy and Computer Science*, 5:41–46.

Urzică, A. and Gratie, C. (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.

von Wright, G. H. (1963). *Norms and Action: A Logical Enquiry*. Routledge & Kegan Paul.

Wang, Y. and Vassileva, J. (2004). Trust-Based Community Formation in Peer-to-Peer File Sharing Networks. In *Proc. on Web Intelligence*, pages 341–348.