# Detecting and Isolating Inconsistently Behaving Agents using an Intelligent Control Loop

Jan Kantert[1], Sarah Edenhofer[2], Sven Tomforde[2], Jörg Hähner[2] and Christian Müller-Schloer[1]
[1]*Institute of Systems Engineering, Leibniz University Hannover, Appelstr. 4, 30167 Hanover, Germany*
[2]*Lehrstuhl für Organic Computing, Augsburg University, Eichleitnerstr. 30, 86159 Augsburg, Germany*
*{kantert, cms}@sra.uni-hannover.de,*

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

Abstract: Desktop Computing Grids provide a framework for joining in and sharing resources with others. The result is a self-organised system that typically consists of numerous distributed autonomous entities. Openness and heterogeneity postulate severe challenges to the overall system's stability and efficiency since uncooperative and even malicious participants are free to join. In this paper, we present a concept for identifying agents with exploitation strategies that works on a system-wide analysis of trust and work relationships. Afterwards, we introduce a system-wide control loop to isolate these malicious elements using a norm-based approach – due to the agents' autonomy, we have to build on indirect control actions. Within simulations of a Desktop Computing Grid scenario, we show that the intelligent control loop works highly successful: these malicious elements are identified and isolated with a low error rate. We further demonstrate that the approach results in a significant increase of utility for all participating benevolent agents.

## 1 INTRODUCTION

Desktop Computing Grid systems are self-organised collectives of resources provided by a potentially large set of participants. For each computer, an agent is responsible for getting the user's jobs processed by others as fast as possible. Simultaneously, it manages which other agents are allowed to utilised the own resources. The resulting system is characterised by openness and heterogeneity - everyone is free to join if following the basic protocol.

Openness inherently defines drawbacks. For instance, egoistic or even malicious behaviour of agents is possible. Therefore, counter-measures are needed to isolate unwanted elements and provide a performant platform for normal and benevolent users. By introducing technical trust, most of these negative effects can be controlled. One major challenge that is addressed in this paper is to identify agents with inconsistent behaviour. This means that an agent tends to exploit the system as soon as it has established stable trust relationships within the system – and returns to benevolent behaviour if others detected the exploitation strategy. In our concept, we extend previous work for the Trusted Desktop Grid (TDG) (Klejnowski, 2014) with an intelligent control loop. This loop works at system-level and follows the Observer/Controller concept (Tomforde et al., 2011) as known from Organic Computing (Müller-Schloer, 2004). Since agents are autonomous, the control loop works on publicly available information, i.e. trust and work relationships. We follow a graph-based analysis strategy and show that even this challenging behaviour of so-called *Cunning Agents* can be identified.

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 system-wide control loop that establishes a norm-oriented management cycle to identify and isolate negative elements. Afterwards, Section 3 describes the approach to detect and isolate *Cunning Agents* in detail. The mechanism is evaluated in Section 4 by simulations of our Trusted Desktop Grid. Therein, we show that the normative approach is highly successful and demonstrate that an isolation is achieved with low failure rate. Section 5 compares the concept to the current state-of-the-art. Finally, Section 6 summarises the paper and gives an outlook to current and future work.


Kantert J., Edenhofer S., Tomforde S., Hähner J. and Müller-Schloer C..
Detecting and Isolating Inconsistently Behaving Agents using an Intelligent Control Loop.
DOI: 10.5220/0005548402460253
In *Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2015), pages 246-253
ISBN: 978-989-758-122-9

## 2 APPLICATION SCENARIO

As a possible application scenario, we investigate open grid computing systems which can host numerous distributable workloads, e.g. distributed rendering of films. The system is considered *open* since there is no central controlling entity, all communication is performed peer-to-peer and agents are free to join. Worker nodes belong to different administrative domains, thus, good behaviour cannot be assumed. Nodes participate voluntarily to submit work into the system and, thereby, increase the speedup of their jobs. However, they also have to compute work units for other submitters.

### 2.1 Agent Goal

To analyse such systems, we model nodes as agents and run a multi-agent system in simulation. Every agent works for a user and periodically receives a job, which contains multiple parallelisable work units. It aims to accomplish all work units as fast as possible by requesting other agents to work for it. Since we consider an open system, agents behave autonomously, and can join or leave at any time. The system performance is measured by the speedup $\sigma$. In Equation (1), $t_{\text{self}}$ is the time it would require an agent to compute a job containing multiple work units without any cooperation. $t_{\text{distributed}}$ represents the time it took to compute all work units of one job with cooperation of other workers including all communication times. The actual speedup $\sigma$ can only be determined after the result of the last work unit has been returned.

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

If no cooperation partners can be found, agents need to compute 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 give their work to and for which agents they work themselves. We do not control the agent implementation, so they might behave uncooperatively or even maliciously.

### 2.2 Worker and Submitter Component

Each agent consists of a worker and a submitter component. The submitter component is responsible for distributing work units. When an agent receives a job containing multiple work units, it creates a list of trusted workers. It then requests workers from this list to cooperate and compute work units, until either no more work units or no more workers are left. If all workers were asked, but unprocessed work units remain, the agent computes them on its own. The worker component decides whether an agent wants to work for a certain submitter. When the agent receives an offer, it computes its rewards for accepting or rejecting the job. There are different strategies based on reputation, workload and environment. If the reward of accepting the job prevails, the agent accepts the job. It may cancel the job later on, but typically it computes the job and returns the results to the worker (Klejnowski, 2014).

### 2.3 Open Systems and Benevolence

In contrast to other state-of-the-art work, we do not assume the benevolence of the agents (Wang and Vassileva, 2004). In such an open system we cannot control the implementation of agents and, therefore, the system is vulnerable to different kinds of attacks. For instance, a *Freerider* could simply refuse to work for other agents and gain an advantage at the expense of cooperative agents. Another attacker might just pretend to work and return wrong results. Also, combinations of both or alternating behaviour are possible. Additionally, attacker can collude to exploit the system.

### 2.4 Trust and Norms

To overcome these problems of an open system where no particular behaviour can be assumed, we introduce a trust metric. Agents receive ratings for all their actions from their particular interaction partners. This allows others to estimate the future behaviour of a certain agent based on its previous actions. To perform this reasoning a series of ratings for a certain agent can be accumulated to a single reputation value using the trust metric. Autonomous agents need to become aware of the expected behaviour in the system. Therefore, we influence the desired actions by norms. These norms are valid for an *Action* in a certain *Context* and, thereby, guide the agents. To enforce the behaviour, they impose a *Sanction* if violated or offer an *Incentive* if fulfilled.

In this scenario, good trust ratings are used as an *Incentive* and, in the opposite, bad trust ratings impose a *Sanction* (agents with higher reputation values have a higher chance to get their work units computed). Based on the norms, 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, the society isolates malevolent agents and maintains a good system utility in most cases. We call this system a Trusted Desktop Grid (TDG) (Klejnowski, 2014).

Since agents are considered as black boxes they cannot be controlled directly from the outside. Each agent is autonomous and selfish. However, we want to influence the system to optimise and make it more robust. Therefore, we introduce norms to change the incentives and sanctions for all agents.

## 2.5 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 input queue of the agent is filled up.
- *Freeriders* - 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.
- *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.
- *Altruistic Agents* - Such agents will accept every job. In general, this behaviour is not malicious and increases the system performance. However, it hinders isolation of bad-behaving agents and impacts the system goals.

## 2.6 Higher-level Norm Manager

In Figure 1, we present our concept of the Norm Manager (NM), which uses the common Observer-Controller pattern (Tomforde et al., 2011). The complete control loop implemented by the Observer-Controller component helps to mitigate effects of attacks to the TDG and allows a better fulfilment of the system goals. Thereby, it defines an intelligent control mechanism working at system-level. However, if the additional NM fails, the system itself is still operational and can continue to run (this refers to

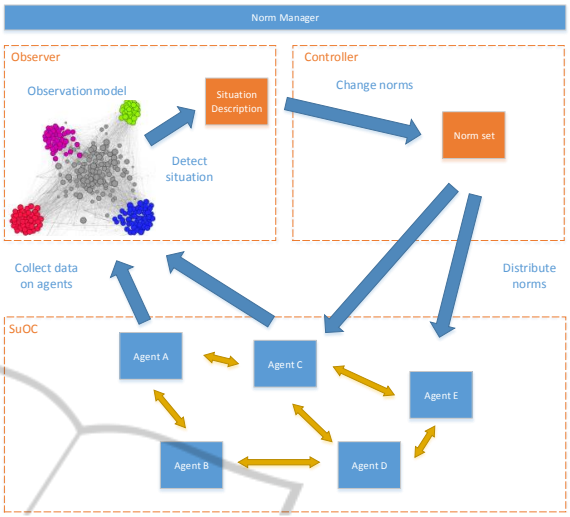


Figure 1: System Overview of the Norm Manager consisting of an Observer and a Controller which control the System under Observation and Control (SuOC) using norms.

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.

### 2.6.1 Observer - Detect Current Situation

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. In this context, the intensity is determined according to the number of shared work packages. 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 and define as how trustworthy agents estimates each other within an interval between 0 (not trustworthy at all) and 1 (fully trustworthy, i.e. self-trust). Since we cannot see the internals or implementation of agents, we need to observe them from the outside. We could monitor all 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.

Afterwards, the NM calculates certain common graph metrics for every node (i.e. *DegreeCentrality*, textitPrestige or *ClusteringCoefficient*; see (Kantert et al., 2014)). Using statistics, the global system

state is determined. Based on these metrics, our algorithm forms clusters and finds groups of similar agents. By further classifying these groups, the Observer achieves an even better understanding about potentially happening attacks. In the end, it is able to classify whether the system is under attack, categorise the type of the attack, and rank attacks according to their severity. This is accompanied by an estimation of how accurate this information is. From a methodical point of view, the observer works as follows: First, it builds graphs for trust and work relations between agents. In a second step, it applies graph metrics to be able to identify groups or clusters of similar agents in the next step. Afterwards, it runs statistics on every cluster found and compares them to historic or threshold values. Clusters are tracked over time to detect tendencies and predict future values.

#### 2.6.2 Controller - Change Norms

The controller is responsible for guiding the overall system behaviour by applying norms. Such a norm contains a rule and a sanction or an incentive (Urzică and Gratie, 2013). Agents are still autonomous and can violate norms with the risk of being sanctioned. Based on the information obtained by the observer, the controller decides whether the system norms need to be changed. Norms can not 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 certain conditions we can allow agents to perform security measures, which would lead to sanctions

## 3 APPROACH

In previous work (Kantert et al., 2015) we demonstrated that clearly malicious agents (i.e. *Freeriders* and *Egoists*) can be isolated using norms and a higher-level observer. In this Section, we present an approach to also detect *Cunning Agents* which is significantly harder to realise due to their exploitation strategy. Afterwards, normative control is performed to isolate these agents.

### 3.1 Detecting and Isolating Cunning Agents

*Cunning Agents* cannot be easily detected and isolated locally in a distributed system. Therefore, they pose a permanent threat to the TDG. However, a higher-level NM can detect them and then can change norms to isolate them.

To detect *Cunning Agents* in the Norm Manager (NM) we cluster groups using the MCL (Van Dongen, 2001) and the BIRCH (Zhang et al., 1996) algorithm on the *trust graph* and classify them using a decision matrix. For the purpose of this paper the matrix can only identify groups of *Cunning Agents*. The metrics used for that purpose were presented in (Kantert et al., 2014). In particular, we use the value of *Authorities* and *DegreeCentrality* to detect groups of *Cunning Agents*.

Once the NM detected groups of *Cunning Agents* it has to isolate them. Unfortunately, it cannot influence agents directly but it can do this using norms. Therefore, it introduces a new norm which allows agents to refuse work from *Cunning Agents*. We propose to detect them based on their inconsistent behaviour.

### 3.2 Inconsistent Behaviour

In trust-based distributed system such as the TDG, we condense a series of trust rating $R$ to a single reputation value $\tau$ (see Equation 3). However, this does not take into account how consistent those ratings are. Normally, agents make their decisions based on the aggregated value $\tau$.

$$r \in [-1,1] \Rightarrow R \in [-1,1]^k \tag{2}$$

$$\tau := \mathfrak{T}(R) := \frac{\sum_{r\in R} r}{\sum_{r\in R} |r|} \tag{3}$$

However, *Cunning Agents* behave strategically regarding their reputation value: They behave well until they reach a certain threshold $\tau_{\text{upper}}$ and then stop to cooperate with other agents until they fall below a threshold $\tau_{\text{lower}}$. Therefore, their reputation $\tau$ is between those two values most of the time and they adjust their $\tau_{\text{upper}}$ and $\tau_{\text{lower}}$ to be considered as well-behaving by other agents based on the reputation (see Equation 4).

$$\tau_{\text{lower}} \leq \tau \leq \tau_{\text{upper}} \wedge \tau_{\text{lower}} \geq \tau_{\text{wb}} \tag{4}$$

Since those agents intentionally exploit the trust metric they cannot get detected and isolated using the reputation value. However, they receive very inconsistent ratings $r$ because of their changing behaviour and we can leverage that to detect them. Therefore, we define the consistency $\kappa$ based on the standard deviation:

$$\kappa := 1 - \frac{\sum_{r\in R, r>0} r}{\sum_{r\in R} |r|} \cdot (\tau-1)^2 + \frac{\sum_{r\in R, r<0} r}{\sum_{r\in R} |r|} \cdot (\tau+1)^2 \tag{5}$$

We expect a very low value for *Cunning Agents* and a value close to one for *Adaptive Agents*. Other agents

such as *Freeriders* or *Egoists* also should get a value of approximately one since they behave consistently maliciously.

### 3.3 Changing Norms

The NM cannot directly influence agents or force them to perform any actions. However, it can change norms and, thereby, change incentives and sanctions for certain actions. To cope with *Cunning Agents* while maintaining the autonomy of agents we choose to allow them to reject jobs from inconsistently behaving agents. Therefore, agents can decide on their own if they want to work for *Cunning Agents* since they will still receive an incentive for that.

In Figure 2, we show the changed norm. The threshold $\delta$ for the reputation $\tau$ is smaller than $\tau_{\text{lower}}$ and threshold $\gamma$ for consistency is 0.8 in our experiment. `requester.reputation` is calculated using the trust metric $\tau$ and `requester.consistency` is determined by $\kappa$. The sanction results in a rating in $R$ for the working agent.

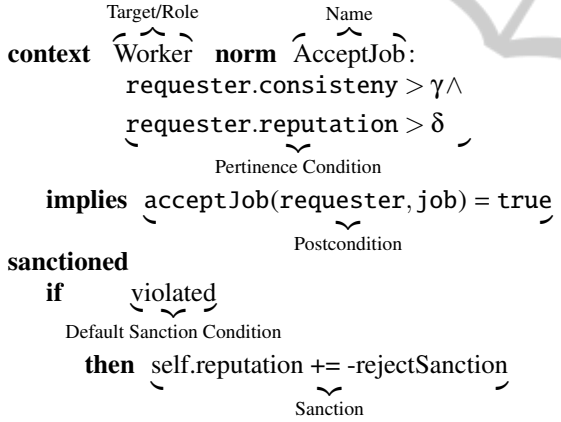


Figure 2: Changed norm used in the evaluation to isolate *Cunning Agents*. It sanctions allows agents to reject jobs from inconsistently behaving agents.

## 4 EVALUATION

In this section we provide experimental validation of our approach.

### 4.1 Setup

The setup in the evaluation consists of 100 *Adaptive Agents* and additional 50 *Cunning Agents* for all experiments with attackers. Each experiment ran for 300k ticks and was repeated 100 times. We performed three series of experiments: (E1) without attacker; (E2) with attackers; (E3) with attackers and our norm changes.

### 4.2 Detection of Cunning Agents

The detection of *Cunning Agents* in (E2) by the NM can be observed between tick 15k and 50k and was successful in all experiments. In average the NM detected the attackers at tick 24920 with a standard deviation of 15301. The NM can introduce the norm at this point. However, to make the impact of the norm change more comparable we set the time $t_{\text{change}}$ to 100k for subsequent experiment in (E3).

### 4.3 Consistency Values

We measured the consistency values for *Adaptive Agents* and *Cunning Agents* in (E2) at the end of the experiment. Both values turned out to be as expected: *Adaptive Agents* have a value of $0.9999 \pm 0.00053$ which is very close to one. In contrast *Cunning Agents* have a value of $0.0611 \pm 0.01338$ which is next to zero.

### 4.4 Influence of Norm Change

We measure the influence of the norm change using the speedup $\sigma$ for *Adaptive Agents* and *Cunning Agents* in experiments (E1), (E2) and (E3) (see Figure 4). In Figure 3, we show one single exemplary experiment of (E3) with speedup over time. In the beginning *Adaptive Agents* and *Cunning Agents* achieve a similar but varying speedup. After the NM introduced the norm change at tick 100k the speedup for *Cunning Agents* falls below one and those agents no longer gain any advantage from participating in the system. In contrast, *Adaptive Agents* gain a stable high speedup again. In the undisturbed experiment (E1), we measured that *Adaptive Agents* can achieve a speedup of $11.74 \pm 0.73$ when there is no attack. When a heavy attack of *Cunning Agents* is added in (E2) the speedup decreases to $5.29 \pm 1.26$. With norm change by the NM the speedup increases again to $6.75 \pm 1.24$ in (E3) which is less than in Figure 3 at the end of the experiment because the value is averaged over the complete experiment. *Cunning Agents* achieve a speedup of $5.23 \pm 1.38$ when they exploit the system in (E1). At the same time they work significantly less than *Adaptive Agents*. However, when the NM changes the norm the speedup of *Cunning Agents* decreases to $2.49 \pm 0.33$. Again, this is averaged over 300k ticks and as shown in Figure 3 the *Cunning Agents* are isolated with a speedup below one at the end of all experiments.


Figure 3: Exemplary simulation run with norm introduced at tick 100k. A speedup of less than one means that agents no longer have an advantage from participating in the system.

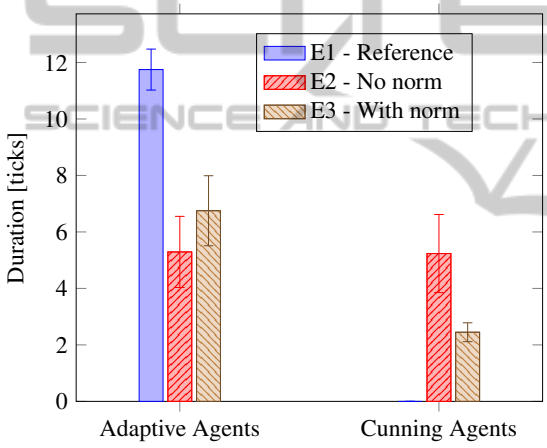


Figure 4: Speedup for *Adaptive Agents* and *Cunning Agents* for every experiment series with 100 Experiments each. In E3 the norm was introduced at tick 100k. All experiments lasted 300k ticks.

# 5 RELATED WORK

Our application scenario is a Trusted 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 each node behaves well. Another implementation with a central tracker is the Berkeley Open Infrastructure for Network Computing project (BOINC) (Anderson and Fedak, 2006). All those systems solve a distributed resource allocation problem. Since work units can be computed faster when agents cooperate, they reward and, thus, maximise cooperation. Additionally, a high fairness value ensures equal resource distribution (cf. (Jain et al., 1996; Demers et al., 1989; Bennett and Zhang, 1996)). We model our grid nodes as agents. Agents follow a local goal which differs from the global system goal (Rosenschein and Zlotkin, 1994). We consider agents as black boxes which means that we cannot observe their internal state. Thus, their actions and behaviour cannot be predicted (Hewitt, 1991). Our Trusted Desktop Grid 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 emphasised there 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).

## 5.1 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 (Ostrom, 1990) observed cases where this did not happen. She presented eight design principles for successful self-management of decentralised institutions. Pitt et al. (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 reason about norms and intentions.

In (Artikis and Pitt, 2009), the authors 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 to certain actions using a predefined protocol. Agents may join or form an organisation with additional rules. The "norm-change" 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. According to (Savarimuthu and Cranefield, 2011), Normative Multi-Agent Systems can be divided into five categories: 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. For norm enforcement, we use sanctioning and reputation. Our network topology is static. Related approaches use Normative Multi-Agent Systems for governance or task delegation in distributed systems (Singh et al., 2013).

# 6 CONCLUSION AND FUTURE WORK

This paper presented a novel approach to identify and isolate agents with inconsistent and partly malicious behaviour within open, self-organised systems such as Desktop Computing Grids. Therefore, we established an intelligent system-wide control loop to guide the self-organised behaviour of heterogeneous agents that are free to join the system if they follow the basic protocol. In general, we identified different stereo-type agent behaviours that also consider exploitation strategies and malicious behaviour. Based on the system-wide identification approach, we proposed to establish an observer/controller loop that issues norms as response to the currently observed conditions. The concept consists of an observer part that derives an appropriate situation description from externally monitorable information, i.e. interactions between agents, and a controller part that generates norms, i.e. demanded behavioural guidelines that are augmented with incentive and sanctioning strategies. For the identification task of exploiting agents, we introduced a graph-based method to detect suspicious agents or groups of agents at runtime.

The evaluation is based on a simulation of a Trusted Desktop Grid in which several classes of stereo-type agent behaviour are considered. The results demonstrated that the system-wide control loop works successful in terms of issuing appropriate norms. These norms are followed by benevolent agents – and as a result, malicious agents are isolated, i.e. they do not find cooperation partners any more. Future work will focus on refining the control loop and improving the controller part. Therefore, we investigate how norms can be generated automatically that a suitable for certain situations. In addition, we currently develop distributed and fully self-organised strategies to monitor whether agents comply to norms or not.

# REFERENCES


Anderson, D. P. and Fedak, G. (2006). The Computational and Storage Potential of Volunteer Computing. In *Proc. of CCGRID 2006*, pages 73–80, Singapore. 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, Singapore. 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*, 0:609–614.

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, Saint-Etienne, FR.

Bennett, J. C. 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, San Francisco, CA, USA. IEEE.

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, pages 1–21, 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, Chichester, UK.

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, Rio de Janeiro, Brazil. IEEE.

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, Paris, France.

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. *Future Generation Computer Systems*, 23(7):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.

Jain, R., Babic, G., Nagendra, B., and Lam, C.-C. (1996). Fairness, Call Establishment Latency and Other Performance Metrics. *ATM-Forum*, 96(1173):1–6.

Kantert, J., Edenhofer, S., Tomforde, S., Hähner, J., and Müller-Schloer, C. (2015). Defending Autonomous Agents Against Attacks in Multi-Agent Systems Using Norms. In *Proceedings of the 7th International Conference on Agents and Artificial Intelligence*, pages 149–156, Lisbon, Portugal. INSTICC, SciTePress.

Kantert, J., Scharf, H., Edenhofer, S., Tomforde, S., Hähner, J., and Müller-Schloer, C. (2014). A Graph Analysis Approach to Detect Attacks in Multi-Agent-Systems at Runtime. In *2014 IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems*, pages 80–89, London, UK. IEEE.

Klejnowski, L. (2014). *Trusted Community: A Novel Multiagent Organisation for Open Distributed Systems*. PhD thesis, Leibniz Universität Hannover.

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

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

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, Michigan, US. IEEE.

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

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. Auton. Adapt. Syst.*, 5:10:1–10:32.

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

Singh, M. P., Arrott, M., Balke, T., Chopra, A. K., Christiaanse, R., Cranefield, S., Dignum, F., Eynard, D., Farcas, E., Fornara, N., Gandon, F., Governatori, G., Dam, H. K., Hulstijn, J., Krueger, I., Lam, H.-P., Meisinger, M., Noriega, P., Savarimuthu, B. T. R., Tadanki, K., Verhagen, H., and Villata, S. (2013). The Uses of Norms. In *Normative Multi-Agent Systems*, volume 4 of *Dagstuhl Follow-Ups*, pages 191–229. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany.

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, Basel, CH.

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, Calabria, Italy.

Van Dongen, S. M. (2001). *Graph clustering by flow simulation*. PhD thesis, Utrecht University.

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, Beijing, China. IEEE.

Zhang, T., Ramakrishnan, R., and Livny, M. (1996). Birch: an efficient data clustering method for very large databases. In *ACM SIGMOD Record*, volume 25, pages 103–114, Montreal, Canada. ACM.