Influence of Norms on Decision Making in Trusted Desktop Grid Systems
Making Norms Explicit

Jan Kantert, Lukas Klejnowski, Yvonne Bernard and Christian Müller-Schloer
Institute of Systems Engineering, Wilhelm Gottfried Leibniz University of Hanover, Hanover, Germany

Keywords: Multi-Agent-Systems, Norms, Decision Making, Desktop-grid System, Trust.

Abstract: In a Trusted Desktop Grid agents cooperate to improve the speedup of their calculations. Since it is an open system, the behaviour of agents can not be foreseen. In previous work we have introduced a trust metric to cope with this information uncertainty. Agents rate each other and exclude undesired behaving agents. Previous implementations of agents used hardcoded logic for those ratings. In this paper, we propose an approach to convert implicit rules to explicit norms. This allows learning agents to understand the expected behavior and helps us to provide an improved reaction to attacks by changing norms.

1 INTRODUCTION

In Organic Computing (Müller-Schloer and Schmeck, 2011) we develop new methods to cope with growing complexity of today's computing systems. Since embedded and mobile devices are getting cheaper and more powerful, new software design paradigms are required. With systems consisting of multiple distributed devices we need to take care of efficiency and robustness, as well as maintaining openness and autonomy. We model devices as agents in a multi-agent system. Since we are in an open system we cannot assume a well-defined behaviour from an agent. To cope with this information uncertainty we introduced a trust metric (Bernard et al., 2010). Agents will cooperate based on trust and can exclude misbehaving agents to ensure high performance.

Our scenario is an open Trusted Desktop Grid (TDG), where agents have parallelisable jobs and need to cooperate to achieve better performance. Agents need to decide for whom they want to work and whom they want to give their work to. Since TDG is an open system, malicious agents like freeriders, which will not work for other agents, or egoists, which will return fake results, may also join the systems. To improve the overall performance those attackers need to get isolated, which can be done by using trust. Agents will give each other ratings based on their actions and will use the corresponding trust value to make their decisions.

In this paper, we convert our internal agent rules, which lead to trust ratings, to explicit norms. This allows other learning agents to understand the expected behaviour in our system. Additionally, it allows us to change norms to adapt to a changed environment. In Chapter 2 we present our motivation to use explicit norms. Afterwards we describe more details about our application scenario in Chapter 3. In Chapter 4, we picture the challenges of decision-making in TDG. After understanding the current behaviour, we propose new explicit norms in Chapter 5. We discuss related and future work in Chapter 6 and finish with a conclusion in Chapter 7.

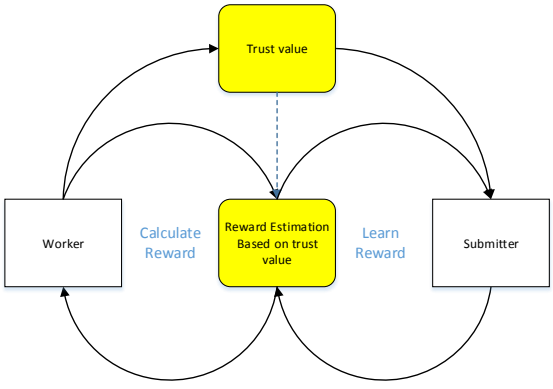

Figure 1: Block diagram of the decision process with worker and submitter.

2 MOTIVATION

In a distributed system with multiple agents it is hard to find a perfect global solution using a central ob-

Kantert J., Klejnowski L., Bernard Y. and Müller-Schloer C..
Influence of Norms on Decision Making in Trusted Desktop Grid Systems - Making Norms Explicit.
DOI: 10.5220/0004918002780283
In *Proceedings of the 6th International Conference on Agents and Artificial Intelligence* (ICAART-2014), pages 278-283
ISBN: 978-989-758-016-1

serving instance, because there are too many parameters to optimise. By giving agents more autonomy we want them to find better and more flexible solutions for job distribution in a changing environment. But since agents are selfish we need to enforce the system goals. Giving them a set of norms allows them to understand the expected behaviour and act according to it. The central instance will create and modify norms to reach the global system goal. However, sometimes agents may offend a norm to perform better and improve the overall system performance.

Agents need to take decisions in a self-referential fitness landscape (Cakar and Müller-Schloer, 2009). Every action taken will influence decisions of other agents, because it will influence trust relations and the amount of work in the system. In our past implementations, decisions were made based on hard limits on the trust value of other agents, and the outcome of actions was known at design time. For example, after working for another agent the agent would get a positive trust rating. In case of missing a deadline or rejecting a work unit the agent would get a negative rating. This worked out very well in most situations. Unfortunately, in extreme situations like collusion attacks or overload those static outcome of actions could lead to trust breakdowns and bad performance (Castelfranchi and Falcone, 2010).

In our current agent implementation we have a static decision mechanism which leads to positive emergent behaviour of the system. However, our Trusted Desktop Grid is an open system, which other agents may join at any time and which does not have a dependable definition of expected behaviour for other agents. This leads to attacking agents being able to exploit the system without breaking any implicit or explicit rules (Bernard et al., 2012). To obtain a clear understanding of the expected system behaviour we want to formulate explicit norms from the implicit behaviour. We want to analyse the norms and improve robustness by adjusting norms afterwards. This will allow us to perform better detection and mitigation of malicious behaviour and thus improved system performance.

3 APPLICATION SCENARIO

Our application scenario is an open Trusted Desktop Grid System. Agents generate a bag of tasks at evenly distributed random times and need to compute these tasks as fast as possible. They can cooperate with other agents and try to maximise their *speedup* by working for each other.

When computing the job on their own it will take

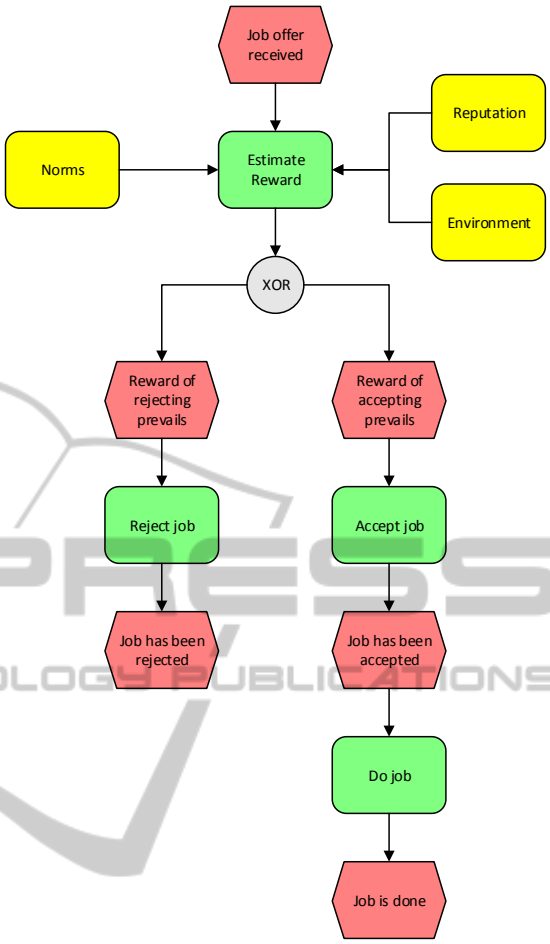

Figure 2: event driven process chain of the worker in Trusted Desktop Grid.

an agent $time_{own}$. When cooperating it will take only $time_{distributed}$, which should be smaller than $time_{own}$. The $speedup$ can be calculated with those two times:

$$speedup = \frac{time_{own}}{time_{distributed}} \quad (1)$$

New agents can join the system at any time. Existing agents can leave the system and will do so if they are unable to achieve speedup greater than one. Every agent needs to decide for which agent it wants to work and to which agent it wants to give its work. Since the TDG is an open system agents will try to cheat to gain an advantage. It is unknown in advance whether an agent will behave according to the rules. To cope with this information uncertainty we introduced a trust metric $T(from,to)$ which describes the trust of agent A_x in relation to agent A_y:

$$x \neq y, -1 \leq T(x,y) \leq 1 \quad (2)$$

The trust value is based on previous interactions between the agents and predicts how well an agent will

behave according to the rules in the system. If an agent A_x does not have sufficient experiences with another agent A_y it will use the global reputation R_y of the agent to rate it:

$$R_y = \sum_{i=1}^{n} \frac{T(i,y)}{n} \quad (3)$$

Every agent has the primary goal to maximise the speedup on calulating its jobs. This is the only reason why it will participate in the system. To give all agents the opportunity to gain a good speedup our system has two goals:

- Maximise cooperation
- Ensure fairness

Since agents can only achieve a speedup when they cooperate the first goal is parallel to the goal of all agents. Cooperation is measured by the sum of all returned work units in the system:

$$cooperation = \sum_{i=1}^{n} \sum_{j=1}^{n} ReturnWork(A_i, A_j) \quad (4)$$

The second goal is needed to prevent exploitation of agents, which would make them leave the system. We measure the fairness of an agent by the submit/work ratio which should be about one. Afterwards we aggregate the difference between ratio and 1 for all agents and try to minimise this value.

$$fairness = \sum_{i=1}^{n} 1 - \frac{submit_i}{work_i} \quad (5)$$

4 DECISION MAKING

Agents have a submitter and a worker component as can be seen in Figure 1. The worker component needs to decide whether it wants to work for another agent and will gain reputation by accepting and completing jobs. The submitter component is responsible for finding the best cooperation partners when a job is available. In general, the submitter will be more successful if the agent has a high reputation.

Every agent needs to make a decision for two questions:

- For which agent do I want to work? (Submitter)
- Which agents will I give my work to? (Worker)

As seen in Figure 2, if an agent gets asked whether it wants to work for another agent it needs to estimate its own reward for rejecting or accepting the job. Rejecting a job will give the agent a negative trust rating of $Penalty_{reject}$. Accepting and returning a job will give it a positive rating of $Incentive_{workDone}$. The worker can also accept and later cancel the job, which leads to a negative rating of $Penalty_{canceled}$, which is usually higher than the penalty for just rejecting the job and therefore will not be a good option in most cases.

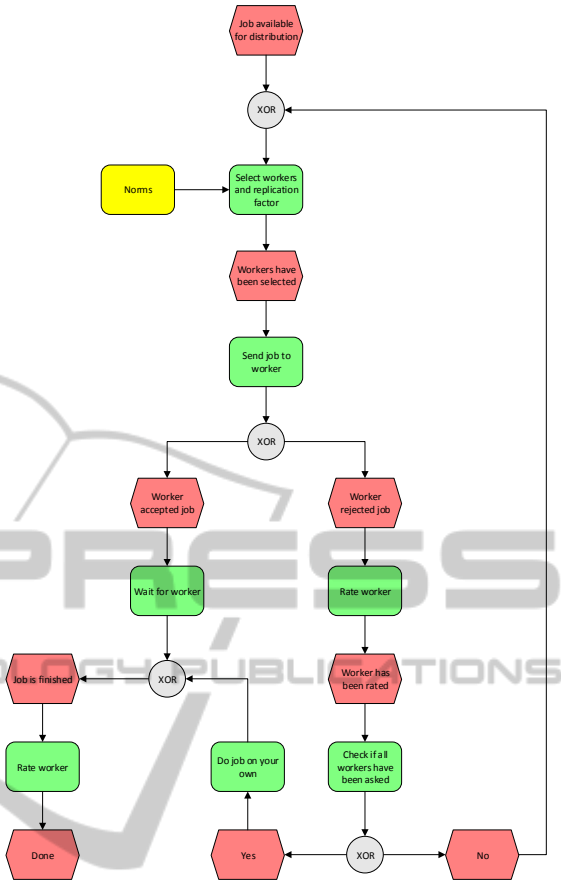

Figure 3: event driven process chain of the submitter in Trusted Desktop Grid.

In the past we had a fixed threshold table for the acceptance of jobs. If one agent A_x got a request to work for another agent A_y it would get the reputation R_y of the requesting agent. Based on it's own reputation R_x it would do a lookup in the threshold table and would only accept the job if R_y is higher than the threshold in the table. Agent A_x will also reject the job if its workqueue is full.

We made this more flexible by using a reward estimator in (Kantert et al., 2013), which calculates the expected reward for a certain reputation. This allowed an agent to estimate the reward for accepting or rejecting a job in advance. When evaluating the new decision making we found out that our previous agents were not always choosing the optimal action to optimise their own reward, which should be their pri-

Table 1: Implicit norms in the Trusted Desktop Grid.

	Evaluator	Action	Context	Sanction/Incentive
1	Worker	$RejectJob(A_w,A_s)$	$T(A_w,A_s) > 0$ $T(A_w,A_s) \leq 0$	$T(A_s,A_w) = T(A_s,A_w) - Penalty_{reject}$ -
2	Worker	$ReturnJob(A_w,A_s)$		$T(A_s,A_w) = T(A_s,A_w) + Incentive_{workDone}$
3	Worker	$CancelJob(A_w,A_s)$		$T(A_s,A_w) = T(A_s,A_w) - Penalty_{canceled}$
4	Submitter	$GiveJobTo(A_s,A_w)$	$T(A_s,A_w) \geq T_{SuitableWorker}$	Speedup

mary goal. Those agents did exclude agents with a low reputation by not working for them, which is generally good for the overall system performance, but the incentive to work for a potential malicous agent is the same as the incentive to work for a well behaving agent with high reputation.

The submitter of an agent will try to distribute jobs to other agents to leverage parallel processing. As seen in Figure 3 it will select a list of workers and orders them by the time they promise to finish the job. Consequently it asks them if they are willing to accept the job. If no worker was found the agent will execute the job on its own. After a job is done the speedup gets calulated and the agent will know how well he performed. We use this value to create statistics and to improve the reward estimator.

In the current implementation we will only select workers with a reputation higher than $T_{SuitableWorker}$. If this did not work the submitter will replicate the job (not shown in Figure 3 for simplicity) and ask multiple workers to execute the work, which will improve the chance that at least one worker will finish the work. Afterwards it will do the calulation on its own. This behaviour is hardcoded and agents do not replicate jobs if the reputation of the worker is high. However there is no mechanism to prevent an extensive use of replication and a learning agent would just replicate every job to improve the reliablility.

With our previous agent implementation we had a set of hardcoded rules, which lead to a good overall system performance. However when we made our agents more autonomous and selfish it turned out that our rating system and rules have some loop holes. Since our TDG is an open system a targeted attacker or learning agent could easily exploit the system to perform better while deteriorating the overall system performance.

We decided to look further into the implicitly expected behaviour of agents in our system. By using hardcoded rules we prevent our agents from performing certain actions, but since there is no limitation other implementations may abuse them. We want to make the rules and expectations in the system more explicit by using norms in the next chapter.

5 NORMS

A norm describes the expected behaviour of one agent in a group agents. Depending on the context a certain action will trigger a sanction or incentive. Because there can no longer be a static decision mechanism we introduced a learning reward estimator to find the best suitable actions for a given environment (Kantert et al., 2013). Since agents will always try to optimise their own reward we need to influence the reward given to an agent. Unfortunately we cannot directly do this, because the reward is based on the speedup reached while computing work units. However agents depend on the collaboration of other agents to achieve a high speedup, and other agents will base their decision to cooperate on the reputation of the agents. We want to exploit this to influence the decision making by norms and indirectly modify the reward an agent gets from the system.

5.1 Representation

We describe a norm by a tuple:

$$Norm = \langle Evaluator, Action, (Policy_1 \dots Policy_n) \rangle \tag{6}$$

$$Policy = \langle Context, Sanction \rangle \tag{7}$$

$$Norm = \langle Evaluator, Action, (\langle Context, Sanction \rangle, \langle Context, Sanction \rangle \dots) \rangle \tag{8}$$

The *Evaluator* is either the worker or submitter part of an agent. Both have different *Actions* they can perform:

- Worker
  - $AcceptJob(A_w,A_s)$ - Agent A_w accepts a job from agent A_s
  - $RejectJob(A_w,A_s)$ - Agent A_w rejects a job from agent A_s
  - $ReturnJob(A_w,A_s)$ - A_w returns the correct calulation for job to A_s
  - $CancelJob(A_w,A_s)$ - A_w cancels job of A_s
- Submitter

Table 2: Proposed explicit norms for Trusted Desktop Grid.

	Evaluator	Action	Context	Sanction/Incentive
1	Worker	$RejectJob(A_w,A_s)$	$T(A_w,A_s) > 0$ $T(A_w,A_s) \leq 0$	$T(A_s,A_w) = T(A_s,A_w) - Penalty_{reject}$ -
2	Worker	$ReturnJob(A_w,A_s)$	$T(A_w,A_s) > 0$ $T(A_w,A_s) \leq 0$	$T(A_s,A_w) = T(A_s,A_w) + Incentive_{workDone}$ $T(A_s,A_w) = T(A_s,A_w)$ $+Incentive_{workDoneLowTrust}$
3	Worker	$CancelJob(A_w,A_s)$		$T(A_s,A_w) = T(A_s,A_w) - Penalty_{canceled}$
4	Submitter	$GiveJobTo(A_s,A_w)$	$ReplicationFactor \geq 1$	Speedup Speedup; $T(A_w,A_s) = T(A_w,A_s) - Penalty_{replication}$

- $AskForDeadline(A_s,A_w)$ - A_s asks worker A_w for the deadline for a job
- $GiveJobTo(A_s,A_w)$ - A_s asks worker A_w to do a job
- $CancelJob(A_s,A_w)$ - A_s cancels a job A_w is working on
- $ReplicateJob(copies)$ - Will copy a job multiple times and use $GiveJobTo()$ on them

A norm may have multiple *Policies*, which consist of a *Context* and a *Sanction*, which can also be an incentive. The *Context* contains one or multiple conditions which must be true to trigger a certain *Sanction*. Since all agents want to achieve a maximal speedup, it is not possible to give a direct reward to an agent and we can only increase or decrease the speedup indirectly by varying the reputation of an agent. The *Sanction* may also influence more indirect values, which can influence the success of an agent:

- Incentive
  - Reputation is increased
  - Monetary incentives
- Sanction
  - Reputation is decreased
  - Loss of monetary incentives
  - Agent gets excluded from Trusted Communities (Bernard et al., 2011)

5.2 Implicit Norms

In Table 1 we listed all implicit norms of our current system. The column *Evaluator* contains the entity, which performs the *Action* in the second column. There are two actors: A_s is the submitting agent and A_w is the working agent. The column *Context* contains the conditions for a certain outcome in the column *Sanction*.

To summarise the implicit norms: An agent always has to accept and return every job if the requesting agent has a positive reputation. An agent should give a job to another agent with at least a reputation of $T_{SuitableWorker}$ or otherwise replicate it. Especially the last part might be difficult: The agents cannot decide to behave differently and accept a sanction. Those are hard rules and we want to change them to give agents more autonomy. When we create norms from an observing instance, we only have a limited system overview. Sometimes it may be better for an agent and the system to violate a norm for improving the overall performance.

5.3 Improved Norms for TDG

To improve norms in the system to enforce the same behaviour as our hardcoded rules we need to solve the following issues:

1. Isolation of not cooperating agents - There should be a motivation for agents to exclude not cooperating agents.
2. Replication should only be used when necesary - There should be a penalty when using replication to limit its usage.

To cope with the first challenge there should be no incentive or a small sanction for working for agents with a low reputation. The easiest solution would be to remove the incentive for working for agents with a low reputation. However, this would prevent the recovery after a trust breakdown (Castelfranchi and Falcone, 2010). If agents do not trust each other, there will not be an incentive for an agent to work for others than itself. A better solution would be to lower the incentive for working for agents with low reputation. Therefore, we introduce a new $Incentive_{workDoneLowTrust}$ with the constraint that $Incentive_{workDoneLowTrust} < Incentive_{workDone}$. In Table 2, Norm 1, we already allow agents to reject a job from agents with a low reputation without any sanctions. We extended Norm 2 to lower the incentive for accepting jobs from agents with low reputation.

One way to solve the second challenge would be to put a small trust sanction on the use of replication. This would lower the reputation of the agent and make it slightly harder for the agent to find cooperation partners. If the agent only makes use of replication infrequently, this will not have a big impact on the overall

reputation. It is still a challenge to enforce this norm since detection of replication is not trivial in a distributed system. We changed Norm 4 in Table 2 to impose a small penalty for agents replicating jobs.

6 RELATED AND FUTURE WORK

This work is part of wider research in the area of norms in multi-agent systems. However, our focus is more on improving system performance by using norms than researching the characteristics of norms (Singh, 1999). We use the same widely acknowledged 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).

Assuming we could detect extreme situations, we want to improve the system behaviour by changing the decision-making during runtime using norms. This would allow us to motivate agents to cooperate in case of a trust breakdown (Castelfranchi and Falcone, 2010) by giving a larger incentive to do so. We could also encorage agents to work with existing peers and temporarily ignore newcomers by lowering the incentive to work with newcomers. However, we do not want to limit our agents too much to allow them to keep their autonomy.

To improve fairness in the Trusted Desktop Grid, it may be useful to have a monetary component in addition to the reputation for every agent (Huberman and Clearwater, 1995). Agents would get a monetary incentive for every finished job and would need to pay other agents for the calculation of their jobs. Trust would be used to prevent malicious behaviour and allow better money exchange.

7 CONCLUSIONS

Making norms explicit helped us to understand the needed behaviour for our system to perform well. It allowed us to detect and to fix potential loopholes, which could be exploited by attackers. Additionally, it gives us the ability to change the expected behaviour at runtime to react to collusion attacks. We plan to experiment with different incentives to adjust the norms to fit the system goals.

REFERENCES

Balke, T., Pereira, C. d. C., Dignum, F., Lorini, E., Rotolo, A., Vasconcelos, W., and Villata, S. (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., Bluhm, D., Hähner, J., and Müller-Schloer, C. (2012). An Evolutionary Approach to Grid Computing Agents. In *Italian Workshop on Artificial Life and Evolutionary Computation*.

Bernard, Y., Klejnowski, L., Cakar, E., Hahner, J., and Muller-Schloer, C. (2011). Efficiency and Robustness Using Trusted Communities in a Trusted Desktop Grid. In *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2011 Fifth IEEE Conference on*.

Bernard, Y., Klejnowski, L., Hähner, J., and Müller-Schloer, C. (2010). Towards Trust in Desktop Grid Systems. *Cluster Computing and the Grid, IEEE Int. Symposium on*, 0:637–642.

Cakar, E. and Müller-Schloer, C. (2009). Self-Organising Interaction Patterns of Homogeneous and Heterogeneous Multi-Agent Populations. In *Self-Adaptive and Self-Organizing Systems, 2009. SASO '09. Third IEEE Int. Conference on*, pages 165–174.

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

Huberman, B. A. and Clearwater, S. H. (1995). A multiagent system for controlling building environments. In *Proceedings of the First International Conference on Multiagent Systems*, pages 171–176.

Kantert, J., Bernard, Y., Klejnowski, L., and Müller-Schloer, C. (2013). Estimation of reward and decision making for trust-adaptive agents in normative environments. accepted at ARCS2014.

Müller-Schloer, C. and Schmeck, H. (2011). Organic Computing - Quo Vadis? In *Organic Computing - A Paradigm Shift for Complex Systems*, chapter 6.2, page to appear. Birkhäuser Verlag.

Singh, M. P. (1999). An ontology for commitments in multiagent systems. *Artificial Intelligence and Law*, 7(1):97–113.

Tuomela, R. and Bonnevier-Tuomela, M. (1995). Norms and agreements. *European Journal of Law, Philosophy and Computer Science*, 5:41–46.

von Wright, G. H. (1963). *Norms and action: a logical enquiry*. Routledge & Kegan Paul.