Trust-aware Agents
for Self-organising Computing Systems

Von der Fakultät für Elektrotechnik und Informatik der Gottfried Wilhelm Leibniz Universität Hannover zur Erlangung des akademischen Grades Doktor-Ingenieur (abgekürzt: Dr.-Ing.) genehmigte Dissertation

von M.Sc. Yvonne Bernard

geboren am 28. März 1982 in Hannover

2014

Referent: Prof. Dr.-Ing. Christian Müller-Schloer

Korreferent: Prof. Dr. Theo Ungerer

Tag der Promotion: 08.10.2014

Danksagung

Ich möchte meinem Doktorvater Prof. Christian Müller-Schloer für die hervorragende Betreuung und Unterstützung danken. Im Rahmen meiner Promotion und Tätigkeit an seinem Fachgebiet konnte ich mich dank seiner vertrauensvollen Führung fachlich wie persönlich entfalten und weiterentwickeln.

Ich danke meinem Korreferenten Prof. Theo Ungerer für seine kontinuierliche konstruktive Unterstützung, nicht nur meiner Dissertation, sondern auch im Rahmen der DFG-Forschergruppe OC-Trust.

Allen meinen ehemaligen Kolleginnen und Kollegen am Fachgebiet System- und Rechnerarchitektur sowie an der "Außenstelle SRA" danke ich herzlich für die gemeinsame Zeit, die von viel Spaß und gegenseitiger Unterstützung in fachlicher wie menschlicher Hinsicht geprägt war. So viele tolle Menschen namentlich zu erwähnen würde den Seitenrahmen sprengen.

Ein besonderer Dank gilt meiner Familie, vor allem meinen Eltern, die immer an mich geglaubt haben, meinem Partner Daniel für sein Verständnis und seine Motivationskraft und meinem Bruder Marcel für seine humorvolle Unterstützung.

Zusammenfassung

In der heutigen Zeit liegt der Herausforderung im Systemdesign darin, Systeme einerseits trotz ihrer immensen Vernetzung beherrschbarer zu machen, andererseits jedoch auch, die Offenheit in solchen komplexen Systemen zu berücksichtigen. Offenheit ist in komplexen Systemen Chance und Risiko zugleich: Werden neue Komponenten (z.B. neue Agenten in einem Multiagentensystem) Teil eines Systems, sinkt die Belastung der einzelnen Subsysteme, da ein zusätzlicher Knoten seine Resourcen zur Verfügung stellen kann. Ein geschicktes Management offener Systeme kann somit zu Performancesteigerungen führen. Andererseits muss das System jedoch sicherstellen, dass neue Komponenten sich vertrauenswürdig verhalten: Handelt die neue Komponente so, wie es von ihr erwartet wird? Sind die ihr übergebenen Daten dort sicher und werden übernommene Aufgaben innerhalb von Deadlinegrenzen zuverlässig korrekt fertiggestellt? Ohne die Implementierung der neuen Komponente zu kennen, muss das Gesamsystem rein aus dem Verhalten einer Komponente erkennen, ob es sich um einen fehlerhaften oder bösartigen Knoten handelt und ggf. Gegenmaßnahmen einleiten.

In dieser Arbeit wird eine adaptive Agentenarchitektur vorgestellt, die eine verteilte Erkennung und Ausgrenzung bösartiger Agenten unter Verwendung von Vertrauenswerten ermöglicht. Basierend auf dieser Agentenarchitektur werden Instanzen verschiedener Agentenklassen vorgestellt. Diese unterscheiden sich in der Fähigkeit, Informationen aus ihrer Umgebung wahrzunehmen (Awareness) und im Verhalten, das aus dieser Wahrnehmung resultiert. Je mehr Informationen ein Agent zur Verfügung hat, desto komplexer können seine Entscheidungen ausfallen.

Die vorgestellten Agenten zeichnen sich durch die Fähigkeit zur selbstorgansierten Bildung von Vertrauensgemeinschaften (Trusted Communities), verschiedene Verfahren zum Lernen optimaler Verhaltensweisen, Vorhersagemechanismen und Anpassungsverfahren für Situationsentwicklungen sowie die Berücksichtigung hierarchischer Erkenntnisse in lokalen Entscheidungen aus.

Anwendungsszenario zur Evaluation der Konzepte in dieser Arbeit ist ein Desktop Grid and Volunteer Computing System, in dem die teilnehmenden Knoten mit einer Agentenkomponente versehen sind, die in der Lage sind, Kooperationsentscheidungen bezüglich der paralllen Nutzung freier Rechenressourcen im System zu treffen und Vertrauenswerte zu erheben. Ziel dieser Agentenimplementierungen ist es, trotz der Offenheit die Performanz und Robustheit des Systems aufrechzuerhalten.

Schlagworte: Organic Computing, Social Organic Computing, Vertrauen, Multiagentensysteme, offene verteilte Systeme, Vertrauen, Reputation, Desktop Grids

Abstract

Nowadays, the focus of systems design is to cope with the interconnectedness of components, but at the same time management of the openness of such systems. The openness of complex systems is benefit and risk at the same time: If new components (here: agents) enter the system, they can reduce the workload of the other system components, because they can take over certain tasks. Therefore, a well-planned self-organised management of open systems can lead to performance improvement.

On the other hand, the overall system needs to make sure that the new components act in a trustworthy manner: Does the new component act as it is expected to? Is data safely stored and will tasks assigned to the component be completed correctly and timely?

Without knowing the underlying implementation of the new components, the overall system needs to detect from their actions, if the node is faulty or even malicious and, if necessary, initiate countermeasures.

This thesis proposes an adaptive agent architecture, which allows for a distributed detection and isolation of misbehaving agents by introducing trust. Instances of agent classes, which differ in the ability to observe information (awareness) and their behaviour based on the observed parameters are presented. The more information is available, the more complex the decision can become.

The agents introduced here are equipped with the ability to self-organise Trusted Communities, different learning techniques to optimise their behaviour at runtime, prediction of and adaptation to situation changes as well as the consideration of hierarchically perceived information in local agent decisions.

The evaluation scenario used to evaluate the concepts in this thesis is a desktop grid and volunteer computing system, in which participating nodes are equipped with an agent component, which is able to make cooperation decisions regarding the parallel usage of idle computing resources and to collect trust information from interactions.

It will be shown, how agent-based mechanisms are able to improve the performance and robustness of such an open complex system.

Keywords: Organic Computing, Social Organic Computing, multi-agent systems, open distributed systems, Trust, reputation, Desktop Grids

List of Abbreviations

AOM	adaptive observation model
BOINC	Berkeley Open Infrastructure for Network Computing
CPR	Common Pool Resource
eTC	explicit Trusted Community
iTC	implicit Trusted Community
JML	Java Modeling Language
MAS	Multi-agent System
OCL	Object Constraint Language
OMG	Object Management Group
P2P	Peer-to-Peer
SuOC	System under Observation and Control
TC	Trusted Community
TDG	Trusted Desktop Grid
UML	Unified Modeling Language

List of Figures

List of Definitions

Contents

Chapter 1

Introduction

1.1 Motivation

System designers nowadays are required more than ever before to cope with the increasing complexity of systems. Decreases in the size, power consumption and prices of embedded devices as well as increasing computational power and multiple communication channels have led to a variety of large systems of interconnected embedded devices. For instance, the complex networks within modern cars have to deal with not only permanent subsystems, but also the driver's smartphone (car-to-x) and even other cars (car-to-car). System designers must recognise this increasing openness. On the one hand, the abilities of the system are being continuously extended and optimised. For instance, the system receives more computational power or functionalities through volatile devices, which increases the possibilities available to the user

for using the system. On the other hand, these volatile nodes can pose a threat to the open system: they could be malfunctioning or even maliciously intending to harm the system.

The overall aim of system designers in such open, complex systems is to find a global system behaviour that is simultaneously efficient and robust with respect to misbehaving agents.

In this context, robustness is regarded as the ability of the system to return to a stable state after a disturbance has occurred—e.g. when a misbehaving agent joins the system.

As a way of dealing with unknown components in open systems, we use trust as a computational concept to measure the expected behaviour of a node. Trust, in general, is an expectation value derived from the rated former interactions. The definition used here describes trust as an aggregation of direct interaction results and reputation, where reputation is the aggregation of rated interactions from other agents. This trust value counters the information uncertainty in open systems. If an agent within such an open system has to decide whether or not to cooperate with another agent, its decision will generally be based on its own interaction history with this agent. If it had no former interaction or if the interaction took place a long time ago, the agent will instead rely on the ratings of interactions that other agents have had with this agent. Thus, the agent has more information available in uncertain situations and is more likely to make beneficial decisions. Therefore, a trust and reputation system that aggregates private knowledge from one's own direct interaction and community knowledge from other agents into an aggregated trust value, is introduced and used as part of the cooperative decision input of the agents developed in this thesis. This trust-enhanced cooperation strategies make agents as well as the overall system robust regarding misbehaving agents and thus increase the system performance. In the literature, no similar approach, which combines trust consideration and adaptive self-organisation algorithms, is present.

The remainder of this chapter is organised as follows. In Section 1.2, the application scenario, which is used for the evaluation of the techniques developed in the thesis, is introduced. Section 1.3 shows which aspects of multi-agent systems are applied and extended in this thesis. The thesis extends various techniques from the organic computing domain as described in Section 1.4. The key component of trust-enhanced self-organisation is introduced in Section 1.5, and the classification, overview and scientific focus of this thesis is presented in Section 1.6.

1.2 Application Scenario Trusted Desktop Grid

The application scenario used in this thesis is an open desktop grid and volunteer computing system [1]. Agents represent users of desktop PCs and act on their behalf (delegation [2]). If the user needs to calculate large computational *bag-of-tasks* applications like video rendering or face recognition, he has the option of letting the agent distribute this task over the network. The agent uses the idle computational power of other PCs in the system. Large tasks are split into smaller work units and given to volunteer agents representing the PCs of other users. In return, these agents can use the idle resources of this agent.

One example of a desktop grid and volunteer computing system is the Berkeley Open Infrastructure for Network Computing (BOINC [3]), where users can add their PC's computing resources to a community. Since projects (e.g. SETI@home [4]) that submit their jobs to BOINC need to be registered, BOINC uses a central scheduling service that can be a single point of failure. We consider applications that produce jobs on the users' machine—e.g. video rendering; therefore, we expect all agents to submit and compute jobs. Thus, we rely on decentralised scheduling and matchmaking. Each agent can assume two roles: submitters give work units to other agents and try to find the best match for their tasks; workers decide whose work units to accept for computation. This completely decentralised trust-enhanced scheduling algorithm is, in contrast to BOINC, more robust regarding partial failures of the network as well as attacks caused by misbehaving agents.

Such open, volatile systems have the ability to increase the users' performance using parallelisation. But they also include threats due to their openness. System designers cannot foresee each situation that the system might be in, especially with regard to misbehaving agents. Misbehaviour can be induced by external factors of the agents—for instance, loss of internet connection, the PC being switched off, or the user's limiting the resources available to the agent to conduct calculations. Moreover, misbehaviour can be a strategic decision made by the agent in order to exploit other agents and, consequently, the overall system. Here, trust is a promising mechanism to counter the information uncertainty.

A selfish agent would try to optimise its benefit by submitting many work units but not computing for others. There are two types of agents showing selfish misbehaviour: *free-riders* do not accept other agents' work units for computation and *egoists* have a high probability of sending no, incomplete or wrong results. Usually, grid systems use replication (sending the same work unit to several agents for computation) and majority voting to determine the correct result. This safety measure

is very inefficient due to the large overhead of calculating the same work units several times. Therefore, in our system, this kind of misbehaviour is coped with by using a trust and reputation mechanism. Each interaction between agents is rated: a positive interaction like the computation of a work unit is rated positive, unsuccessful interactions like declining, wrong or incomplete computation of work units are rated negative. These trust values are used in both, *submitter* and *worker* roles. Agents in the submitter role are able to submit work units to all agents, but can decide to request for only the most trustworthy ones. Therefore, they are more likely to find a cooperative interaction partner that increases their efficiency. In the worker role, agents will only accept work units from agents with a minimal trust value. The coupling of worker behaviour and submitter success leads to a self-organised exclusion of uncooperative agents. If agents are uncooperative as workers, they are rated as untrustworthy and will not find cooperation partners in the worker role. Thus, the computation of their jobs will take longer, and the uncooperative agent's benefit from being part of the system will be minimised. Hence, the uncooperative agent will either leave the system or change its behaviour and become more cooperative.

The set of agents where all members are cooperative and mutually trust each other is called a *Trusted Community*. We call our open desktop grid and volunteer computing system, which is enhanced by trust-based self-organisation, the *Trusted Desktop Grid (TDG)*. Similar approaches that use trust or adaptation for scheduling and matchmaking are H-Trust [5] and the Organic Grid [6]. However, the adaptivity and self-organisation abilities of these approaches are not as far-reaching as in the TDG, especially regarding disturbances caused by misbehaving agents. We will compare our concepts and results to these approaches later in this thesis.

The TDG is a specialisation of the general class of *common pool resource (CPR) problems* [7]. Computational power is a resource shared among the community of agents. All agents use this resource, but the times at which the resources are provisioned and consumed are heterogeneous and depend on the users and on their applications generating jobs—e.g. a user requests the rendering of a video each time he finishes a scene using his 3D creation tool. An overuse of computational power within a desktop grid would lead to full queues, and agents would see lower benefits due to long waiting times. This might motivate agents to leave the open volunteer grid, which would destroy the grid system over time. This 'tragedy of the commons' in our application scenario is prevented by the trust-based self-organisation mechanisms.

The main reason for using the TDG as an application scenario is that it represents

both, open systems and CPR problems. Moreover, it is a scenario with practical relevance that can be used both, as a simulation and on top of a middleware for grid-based systems as developed in the OC-trust project.

The system is distributed without central control. The considered applications produce bag-of-task jobs—i.e. tasks that are independent of each other. Such an open desktop grid is suited for scenarios where most clients run applications that produce grid jobs and thus have high demand for computing resources—e.g. video rendering.

According to the taxonomy of Choi [8], we classify the agents of this desktop grid system as: egoistic, volatile, distributed over the internet, dynamic, faulty and heterogeneous.

Clients have the capabilities to be both submitters and workers, which is described below in detail. The clients are assumed to be heterogeneous in terms of administrative domains, machine resources, usage patterns, volatility, etc. Such a grid is suitable for scenarios where most clients run applications that produce grid jobs and, thus, have high demand for computing resources.

In the TDG, agents become *submitters* whenever a user application on their machine produces a grid job. These jobs are split into single work units that are distributed among available worker clients. The *workers* process them and return the results to the submitters, which validate the results. However, these systems are exposed to threats by clients that plan to exploit or damage the system. A worker can, for example, return an incorrect result or not return a result at all. Workers can also refuse to accept a WU. In the area of volunteer desktop grid systems, such cheating behaviour is a serious issue [9].

Usually in such systems, information uncertainty regarding the behaviour of agents from other administrative domains is coped with using replication. A work unit is given to several agents, which compute and return the result. Majority voting then determines the best result. This is very inefficient due to the tremendous overhead caused by a work unit being computed several times. The payload in the system is then decreased accordingly.

Agents can act as worker and submitter at the same time.

Here, trust mechanisms can help the agents to estimate the future behaviour of other agents. Trust-considering agents are able to overcome the information uncertainty in open systems and improve their performance. By extending each client with an agent component and modelling the relations between the agents with a trust mechanism, we expect to counter these threats, thereby increasing the efficiency of

such a system. If, for instance, an agent chooses only those workers that it already had good experiences with, the expected outcome is better.

Agents use trust information to overcome the information uncertainty in open systems in the submitter and worker components, in which they make the following decisions:

- Submitter: To which agent will I give my jobs? (Submission)
- Worker: For which agents will I work? (Acceptance)

In order to include trust information in these decisions, the agents need a trust and reputation system as described above. Therefore, the following types of misbehaviour during agent interaction are rated negative:

- Work unit cancelling: The owner needs to find a new worker, and the calculation time until cancelling is the calculation overhead.
- Delayed computation (e.g. because resources have been used for own purposes): The owner needs to resubmit the work unit.
- Wrong results (e.g. the work unit has been completed, but the result is wrong/-faked): This can happen due to hardware failures or be done on purpose by malicious agents that try to harm the system.
- Free riding: Generally rejecting the work units of other agents.

Agents showing one of these behaviours receive negative ratings. Thus, they are considered less trustworthy and are less likely to be regarded as desirable cooperation partners.

Positive behaviour, on the other hand, leads to good ratings:

- Returning a correctly computed work unit in time
- High reliability (often online)

Agents showing positive behaviour are rated positive and are desirable cooperation partners because the possibility of reaching a good performance increases if an agent cooperates with them.

Therefore, trust helps agents to find suitable cooperation partners and, therefore, to increase their own performance as well as the overall system performance. Due to the negative consequences of misbehaviour (bad ratings and, thus, decreased own

performance due to control loops), agents are incentivised to change their behaviour and become more cooperative.

Moreover, misbehaving agents are identified in a distributed fashion and excluded from the implicit Trusted Community (iTC, Section 5.2.2).

1.3 MAS

We model the TDG as a multi-agent system (MAS) in concordance with Foster [10]. The TDG agents can be used in a simulation environment as well as on top of a middleware for grid systems. In this thesis, we concentrate on the evaluation of the trust-adaptive agent mechanics using the TDG simulation. We extend the general grid functionality (and, thus, system performance) by adding the adaptivity of an agent-based approach. Agents act on behalf of their user and make autonomous decisions.

1.3.1 Agent Architecture Enabling Proactive Decisions

The *adaptive agent architecture* presented in this thesis enables agents to adapt proactively to changing situations. This is done by decoupling the complex decision mechanisms into different decision levels. Using forecasting techniques, agents are able to detect an upcoming change in situations or other agents' behaviour. Based on this, they are able to modify their behaviour in order to minimise the effects of the future situation change by adapting even before the situation occurs. This goes beyond the reactive definition of classical agents. Moreover, we extend the awareness of the agent as well as its decision space by adding long-term considerations and more flexibility to their behaviour.

1.3.2 Awareness

Moreover, the adaptive agent architecture enables agents to control their behaviour depending on the needs of the current situation. In a critical situation, for instance, an agent needs more information within shorter time intervals in order to make the best decision. In non-critical, stable system states, agents can save communication efforts by observing less information less often and still making reasonable decisions. This is especially useful in situations involving limited bandwidth—e.g. over the internet. Apart from information about the environment, agents also need information regarding their possible cooperation partners: their availability, workload and trust values are important parameters for its own cooperation decisions. The *adaptive*

observation model presented in this thesis enables agents to change the parameters that they collect and analyse at run-time to the current situation's needs. Therefore, we extend the current notion of awareness used in the MAS community to adaptive awareness in order to enable the agent to make the best decision possible. Furthermore, the adaptive observation model enables agents to save communication effort by only observing and evaluating the parameters that are currently relevant to them.

1.3.3 Institutions

Additionally to the self-organisation mechanisms, we propose mechanisms to counter situations that cannot be foreseen by single agents. A trust breakdown, for instance, is a situation where agents lose trust in other agents—e.g. due to overload situations or other internal or external disturbances—and, therefore, do not find cooperation partners. This situation cannot be observed through the local view of an agent. Therefore, to counter such a situation, we need an *institution* that is able to observe the situation and legislate norms which lead out of such undesired system situations. We propose a mechanism within the adaptive agent architecture, which enables the agents to 'listen' to such institutional norms and to take these norms into consideration during behaviour selection reasoning. Thus, we are able to resolve global situations using a hierarchical component that reacts only if necessary, and leaves the system to the general control of the local self-organisation process whenever possible.

1.4 Extension of Organic Computing Techniques

In this thesis, different techniques from the Organic Computing initiative are used and extended in a general, reusable fashion. The adaptive agent architecture extends the *Observer/Controller pattern* [11]. The idea of this pattern is that a system under observation and control (SuOC) is continuously supervised by a higher-level observer analysing the situation. As a consequence, the SuOC is parameterised by the controller, which acts as soon as the observer detects a situation change. We extend this pattern in four ways:

1. **OC Extension 1: Prediction** The observer not only analyses current situations, but also tries to forecast the future. Accordingly, the controller is able to react based on this forecast with the suited adaptation of current behaviour parameters.

2. **OC Extension 2: Norms** The controller is able to receive norms from a higher-level institution outside the agent and to reason whether or not it is worthwhile to obey this norm.

3. **OC Extension 3: Learning** In order to further enhance this self-organisation process, we analyse and adapt suitable learning techniques (cf. Section 4.1.1) that optimise the agents' behavioural decision-making process.

4. **OC Extension 4: Adaptive Observation Model** We not only use the well-known forward link between observer and controller, but also introduce a backward link: the controller is able to define which parameter it currently needs the observer to observe and analyse. This novel addition enables agents to save communication effort; thus, it enhances the bandwidth for the payload.

Figure 2.1 in the following section shows how these OC extensions are embedded in the architecture of this thesis.

1.5 Self-organised Trust-based Agent-Society Coordination

In this thesis, we show how agents must be designed in order to adapt to changing situations and optimise system states in a distributed fashion.

On the agent level, this is realised by the adaptive agent architecture introduced in Section 1.3, which leads to a new class of adaptive, trust-aware learning and proactive agent implementations.

One of the key self-organisation processes on the system level is the formation of *trusted communities (TCs)*. Agents form TCs in order to achieve better efficiency and robustness for the overall system, which is reached by maximising the agents' own efficiency. This is carried out by interacting with the most trustworthy cooperation partners.

Such TCs can be explicit (*explicit Trusted Communities (eTC)*), which means they have a dedicated TC manager (TCM)that is elected as a hierarchical component. The membership information is available to all agents. TCs can also be implicit (*implicit Trusted Communities (iTC)*), which means there is no explicit membership function, and each agent has its own view defining who is part of the iTC.

iTCs are groups of agents formed by the exclusion of misbehaving agents. This is achieved through each agent locally by ranking the available agents using trust

values. The formation of iTCs is a purely self-organised mechanism and will be analysed in this thesis.

1.6 Classification and Scientific Focus

This thesis is classified into the field of agent architectures in the context of self-organising and adaptive systems, MAS (in general) and volunteer desktop grids. We study how agents can be designed to self-organise in a way that maximises system efficiency and robustness. This goes beyond the standard definition of agent architectures and their dynamics (e.g. BDI, ContractNet). These general techniques are evaluated in an open system, and the TDG is an instance of the class of CPR problems. The agent techniques developed in this thesis are not limited to the desktop grid domain; rather, are applicable to all instances of CPR and similar problem classes.

The thesis is organised as follows. In Chapter 2, we will introduce the main research questions that the thesis will address. In Chapter 3, the related work will be presented and we will argue, in which way the techniques presented in the thesis differ from the existing approaches.

In Chapter 4, the techniques from the state of the art, which have been adapted in order to be used in this thesis, are discussed. The system view of trust feedback loops and self-organising TCs is shown in Chapter 5. Chapter 6 will introduce the adaptive agent architecture, which is the general framework of the agent implementations presented in this thesis, and a hierarchy of the different classes of agents in this thesis. Chapter 7 presents the evaluation results of trust-adaptive agents and TCs, and compares them to the state of the art. The last chapter summarises the developed techniques and their effects.

Chapter 2

Research Questions and Aim of This Thesis

The general aim of this thesis is to develop agents that are able to adapt their behaviour to the current situation. This is carried out in the context of an open complex system, which means that agents can join and leave the system at any time. Thus, the agent cannot foresee the behaviour of other agents in the system; it has incomplete information. *Trust* as a computational concept based on rated former interactions can help in this respect. In order to cope with incomplete information, agents use trust values that, as an aggregation of own direct trust and *reputation* (the aggregated trust values of other agents) hint at how the unknown agent might behave in the future. Agents then decide, based on trust values (and other information like workload), with which agents cooperation is worthwhile. Furthermore, we evaluate how machine-learning techniques can be used to optimise the agents' trust-based behaviour decisions.

In the TDG, cooperation decisions are made with respect to two roles. In the submitter role, the agent searches for the most trustworthy available agent that is able to calculate its work unit. In the worker role, the agent decides whose work units to accept for computation if it is asked for this type of cooperation. Due to the trust value representing the other agents' experiences with an agent, an agent will only be successful in a submitter role if it has been cooperative in a worker role. This is because agents adapt their behaviour based on trust values; therefore, they are only cooperative with cooperative agents.

Apart from current trust information, we extended the agents' awareness of long-term information. Agents are able to track previous situation parameters and predict future situations. These predictions enable agents to act proactively: they adapt to

changing situations before the situation occurs.

The amount of adaptivity must correspond to the needs of the current situation. If there is no dynamism in the system, the agent does not need to predict situation changes (because they do not exist in this situation and thus are trivial). But as soon as the situation is dynamic, predictions are worthwhile to consider to improve agent performance.

Similarly, trust is necessary only if there are misbehaving or, at least, unknown agents in the system. If all agents are known as cooperative agents that never change their behaviour, the trust and reputation mechanisms are just overheads. Therefore, it is important that agents adapt their behaviour as well as their awareness to the current situation. Agents only have to observe those parameters that are important for their decisions in the current situation (*adaptive cognition*). By adapting the set of observed parameters to the situation, agents can save communication overheads without sacrificing the quality of their behaviour decisions and, thus, can maximise their performance.

We introduce an architecture that decouples long-term decisions from short-term ones. The long-term decision parameterises the decision space of the short-term behaviour decision. For instance, if the long-term analysis predicts that the workload is going to increase, the long-term decision process will constrain the set of possible parameters of the short-term behaviour decision to only those parameters that are cooperative, thereby enabling the agent to cope with the high future workload. The short-term behaviour decision is able to select a behaviour within the preselected parameter space.

Moreover, the agent is able to consider norms from an institution (e.g. the manager of an eTC) by 'listening' to the legislated norms and incorporating them in their long-term decision-making process. Thus, agents are able to resolve global unwanted situations that could not be observed using local knowledge alone.

This thesis aims to show how an agent architecture that allows for all these aspects, can be designed. Moreover, we propose a hierarchy in which all aspects of the agent's awareness and abilities are considered as agent classes. For each of these agent classes, we present an implementation and evaluate the benefits of this class in the TDG scenario.

Figure 2.1 gives an overview of the building blocks of this thesis including the OC extensions introduced in Section 1.4.

The application scenario TDG (khaki) has already been introduced in Section 1.2.

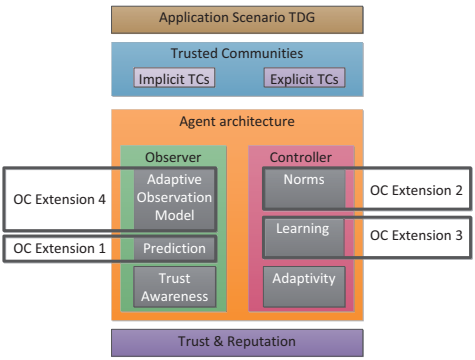


Figure 2.1: Overview of the building blocks of this thesis

Chapter 5 introduces the system model used in this thesis. This relates to trust-based interactions, which represent the main mechanics of the 'Trust & Reputation' (purple) building block, as well as to the self-organising TCs (blue), which includes both, implicit and explicit TCs. Chapter 6 will present the Agent Architecture (orange), which includes agent implementations showing the research aspects on the Observer (trust awareness, adaptive observation model) and controller (norms, learning, adaptivity) levels.

Chapter 7 presents the evaluation results of trust-adaptive agents and TCs, and compares them to the state of the art. The last chapter will summarise the developed techniques and their effects.

In the following chapter, the related work and state of the art of the aspects of these building blocks, especially regarding grid systems and trust, will be presented.

Chapter 3

Related Work

In this chapter, the related work is presented and analysed.

First, we will provide a deeper understanding of the application scenario used in this thesis. Therefore, we classify this volunteer desktop grid system and the types

of applications for which it is suited. We will analyse similar approaches regarding the system architecture and the already existing scheduling and matchmaking techniques.

This thesis deals with trust and reputation systems, in general, and places particular focus on a trust and reputation mechanism tailored to the agents developed in the thesis and used in the application scenario. Therefore, we analyse the related work of trust and reputation systems regarding the aspects necessary for our needs.

This chapter focuses on competing approaches, therefore, we analyse why the existing technologies are not sufficiently suited for our application scenario and thus justify the development of the approaches presented in this thesis.

3.1 Grid Computing

This section aims to provide an overview of the related work pertaining to the application scenario of the Trusted Desktop Grid (TDG). First, possible jobs and applications within the TDG will be introduced and defined. The difference between centralised and peer-to-peer systems will be pointed out, and we will compare the TDG mechanics to the related work of decentralised volunteer grid systems.

3.1.1 Data and Task Composition in Grids

Distributed computing has a broad variety of applications. For instance, this can include computer-aided biology, weather forecasts, particle physics, and astronomy [12]. Well-known projects within this area are gravitation wave research (Einstein@home), protein folding (Folding@home) [13], and pattern recognition in radio telescope data (SETI@home) [14]. The projects are either data or computational intensive; sometimes both, as the two are not always completely distinguishable. Therefore, in this thesis, we abstract from the classification of the job-generating application and concentrate on the bag-of-tasks application. This class of application defines jobs as a set of smaller work units that do not depend on the results of previous work units or jobs.

3.1.2 Bag-of-tasks Applications

Definition 1. Bag-of-tasks applications:

Bag-of-tasks applications are applications that produce jobs containing causally independent work units.

Thus, each work unit of a job can be computed without communication with other work units or processes. Therefore, bag-of-tasks jobs are 'embarrassingly parallel'; they can be distributed independently with ease. This parallelisation can be used to reduce the computation time of large jobs within distributed systems. In order to reach the best *speedup* (time taken for serial computation on the owner's PC divided by the time taken for parallel computation in the grid), it is essential for an agent to find enough trustworthy, reliable workers that will use the parallel computation to its best advantage. Therefore, we enhance the scheduling and matchmaking of our agents in order to find the 'best matches' in a given situation.

3.2 Volunteer Desktop Grid Systems

A grid can be described as a pooling of geographically distributed computer systems. We differentiate between institutional grid systems and desktop grid systems. Institutional grids consist of clusters that are interconnected, but from the same or related administrative domains that also fund and maintain the grid. In contrast, desktop grids usually contain PCs from different administrative domains. In volunteer desktop grids (VDG), private users offer their computing resources to other users. Using idle resources is a cost- and energy-efficient method of desktop computing, but also requires new ways of dealing with the maintenance of these systems. The scheduling in such VDGs can be either centralised or decentralised as pointed out in the following.

3.2.1 Centralised Desktop Grid Systems

One of the most popular approaches used by volunteer desktop grids is the Berkeley Open Infrastructure for Network Computing (BOINC) [14]. Developed at the Space Sciences Laboratory of U.C. Berkeley, BOINC became a well-known project for scientifically distributed supercomputing on private volunteer PCs over the internet. In order to participate, each grid user installs a client software on his PC, which sends performance data to a centralised server. This central component then organises the scheduling: It decides which client uses which work unit. Abstract scheduling in grid systems is a problem that, depending on the model prerequisites, is either NP-hard[15] [16] or NP-complete[17] [18]. The difference between both classifications is that the authors defining the problem as NP-complete define the problem itself within NP complexity, whereas others do not define the problem itself into the set of non-polynomial decision problems. Therefore, both definitions can be regarded

as valid, especially as we are mostly interested in the conclusion that scheduling in grid systems has a high complexity and make heuristics a worthwhile candidate for consideration. The main drawback of centralized grid management systems like BOINC is that the central scheduler is a single point of failure. Since volunteer grids are open systems, system designers need to cope with the risks of attacks not only from outside, but also from inside the system by malicious agents. For instance, corrupted clients can be spread over the internet or a user might develop his own malicious grid client.

Apart from the security issues, the central server of centralised grids like BOINC can also form bottlenecks if the system becomes too large to handle all requests. Decentralised scheduling overcomes these drawbacks. Therefore, however, it needs to find heuristics in order to make good scheduling solutions at run-time.

3.2.2 Peer-to-peer Desktop Grids

As pointed out above, centralised desktop grid systems have drawbacks in terms of reliability (of the central component) and scalability (of the central entity). Therefore, several decentralised solutions for grids have been developed. A prominent example is Gnutella [19], which enables users to share files and data with each other. The grid management tasks are distributed among the membersand PCs with higher computational capacity, and/or better bandwidth is chosen to manage more traffic than weak nodes in a distributed fashion. The grid client software is responsible for these management tasks, thereby ensuring that the user does not have to worry. Similarly, users could install a grid client software that incorporates the agent software. The basic idea that grids and MAS can be combined was introduced in Ian Foster's *Brain meets Brawn* [10]. Foster's idea was to enhance the robustness mechanics of decentralised grids with the adaptivity and scalability of MAS. In this thesis, the TDG is implemented as an instance of the combination of peer-to-peer desktop grids and MAS. We additionally introduce trust here because a combination of volunteer grids and MAS needs to cope with the openness of volunteer grids in a context of local, autonomous agent interactions.

3.3 Requirements of Grid Systems

The classification of grid systems given in the previous subsections has shown that there are several different grid systems available. Nonetheless, there are certain requirements a grid system needs to fulfill in order to be suited for our application

scenario. Their requirements are introduced in this section and then, in the following two sections, applied to the related work in grid systems, scheduling and matchmaking mechanisms.

1. First of all, the application scenario needs to be suited to process **bag-of-tasks** applications. The large computation jobs need to consist of smaller work units that do not depend on the results of previous work units or jobs. Therefore, the distribution of the work units does not have to deal with causal dependence.

2. Due to our aim to build scalable systems, we want the organisation of the grid system to be **decentralised**. Therefore, we are able to enhance local agent algorithms and thus reach a bottom-up global effect.

3. Moreover, we want the decentralised management of the system to be **scalable** as well, which does not necessarily have to be the case in all decentralised systems. Therefore, the system management algorithms need to deal with large system sizes as well as high dynamics within the overall systems.

4. Despite the complexity of large systems, we also optimise the abilities of the system to deal with **openness**. The system is volatile, agents can join and leave at any time. Additionally, the system designer cannot control each agent implementation, agents might be malfunctioning, selfish, unreliable or even malicious. This can be broken down to the following two requirements of systems being either trust-based or adaptive regarding different system situations and agent behaviour implementation. The most promising, but also most complex approach would be a combination of trust consideration and agent adaptivity.

5. **Trust-based** systems are able to overcome the information uncertainty of open systems by using trust information. This trust information helps agents to make cooperation decisions although they have not enough own experience with other agents. Therefore, trust can help to increase system performance as well as robustness regarding attacks from misbehaving agents.

6. **Adaptivity** in grid systems can be applied in different ways: Agents can adapt to changing environmental conditions, changing agent compositions in the overall system and changing user requirements. The requirement of adaptivity in this analysis therefore focuses on adaptivity regarding agent behaviour in order to improve the system's performance and robustness.

Based on these requirements, in the following sections we will analyse grid systems (Section 3.4) as well as scheduling and matchmaking mechanisms (Section 3.5) regarding their applicability as application scenario.

3.4 Trust-based or Adaptive Approaches to Grid Systems

This section presents the state of the art of adaptive and trust-based desktop grid systems and compared to the requirements we determined in the precious section. Especially, approaches that show aspects of either adaptivity or trust are introduced in this section.

3.4.1 QGrid

The QGrid framework [20] is an adaptive system for resource management in grid systems that consist of PCs of different users over the internet. Based on a q learning [21] strategy and trust factors, the system aims at reducing the benefits enjoyed by uncooperative users. The underlying strategy is a market-based one that uses a virtual warranty. A resource provider decides according to a bid and trust value of a consumer whether to provide the requested resources. The framework is implemented within the CROWN grid, which is based on a two-tier architecture. The upper layer is a high performance server that collects and maintains information about the member's nodes. Although it uses CROWN, QGrid is a centralised system; therefore, it is not a suitable alternative for our decentralised application scenario..

3.4.2 RIDGE

RIDGE [22] is based on the BOINC architecture, but replaces the distribution strategy with a reputation-based algorithm. The members are rated based on their former calculation results and speed. In order to validate a work unit result and thereby determine the set of reliable members, the work units are replicated and rated using M-first voting [1]. Due to the centralised BOINC architecture 3.2.1, RIDGE also uses central administration and scheduling. Thus, it is not suitable for our application scenario.

[1] "'In M-first voting, each workunit is replicated into at least M tasks and a workunit is said to have completed successfully as soon as M results match."' [22]

3.4.3 WaveGrid

WaveGrid [23] is a peer-to-peer based desktop grid approach. The general idea is to utilise the idle cycles of PCs around the world. Due to the geographic distribution and different night phases in different geographic time zones, idle times can be seen as a wave rotating around the globe in 24 hours. Usually, users need the computing resources at daytime, and PCs (or clusters, servers, etc.) idle during the night. In order to overcome this imbalance, WaveGrid aims at computing jobs in time zones where most PCs idle. In particular, WaveGrid builds an overlay network organising available nodes across different time zones. Thus, this peer-to-peer-based system reaches a load balancing based on worldwide time zones. This approach is interesting, but does not include trust information or other techniques to cope with misbehaving agents.

3.4.4 A Pure Peer-to-peer Desktop Grid Framework with Efficient Fault Tolerance

A pure peer-to-peer desktop grid framework is presented in [24]. It uses an unstructured peer-to-peer model in which the worker and submitter roles are connected, and both roles are assumed simultaneously. Based on *IamAlive* messages, the state of a note is broadcasted to all members of the grid at predefined intervals. Time-outs then show that a node has left the system. Requests regarding resources are also broadcasted to all agents.

Although the authors argue about parallel and serialised distribution as well as different interaction modes, information about the scheduling and matchmaking mechanisms are not presented in detail. Due to the purely broadcast-based system, this approach has a lack of scalability and therefore does not fulfill the requirements of our application scenario.

3.4.5 H-Trust

H-Trust [25] is a trust- and reputation-based desktop grid approach. The system makes use of the Hirsch index [26], which is a function used to quantify the scientific contribution of an individual. Therefore, the number of publications as well as the number of citations of a person is evaluated. In the case of H-trust, the calculation of trust and reputation is very similar. The calculation is based on three tables that are stored locally by each grid member. In order to compute local trust, a local service history table (LSHL) is used, in which the last experiences with other agents

are stored. These events are rated and stored in a local trust rating table (LTRT). Due to the limited storage, a kind of forgiveness is implemented and old experiences vanish over time. Thus, agents that improve their behaviour have the opportunity to become active members of the system even if they have been avoided in the past. Since not all members can own local trust information about every other member (there might be some with whom they had no personal experience), a reputation system is used to overcome this lack of information. Therefore, a credibility table is used which includes the credibility of all other members. If a submitter A does not possess any own trust information about an agent B, it asks its known agents for their trust value for B. The recommendations that it receives for B from agents with a high credibility are stored and ordered by their trust value. According to their rank in the table, agents are chosen and their cooperation sought. If a cooperation attempt ends with a negative experience (e.g. work unit not completed), agent A lowers the credibility of all agents that recommended B to it. If the cooperation is successful, the credibility of all recommending agents is increased.

This trust-based approach works in a completely decentralised manner and is suited for decentralised scheduling of bag-of-tasks applications. Despite the usage of trust, the lack of adaptivity might lead to management overhead which influences the scalability of the approach. Therefore, H-Trust is not an approach which completely fulfills all criteria, but, due to its closeness to our requirements, a candidate for comparison with the mechanisms developed in this thesis.

3.4.6 Organic Grid

A further decentralised approach is the Organic Grid [6]. The overlay network structure of this grid is a tree that is built by each agent in a submitter role. The submitter itself is the root node. It orders the other agents according to their calculation performance. In order to build such a tree, each agent needs a list of known grid members as a starting point.

The work unit distribution in the Organic Grid is in the pull mode. If an agent is available as a worker, it advertises its computing resources and asks for work units to compute. In contrast, all other adaptive or trust-based grid systems in the related work use the push mode, where an agent in a submitter role needs to ask other agents for computation.

The Organic Grid uses mobile agents: if a worker asks for work units, it receives a copy of a mobile agent from a submitter, which then performs the task allocation. The worker becomes a child node of the submitter. If the submitter has a work

unit computation task for the worker, the mobile agent receives the task. A worker can forward work units to its sub-nodes by copying a mobile agent to them. The tree overlay is built in this way. A computed work unit is then delivered from the executing worker back to the submitter node along the network.

If, for a given task, a set of jobs comprising several work units has been completed, the submitter sends a termination command to all child nodes, which then forward this command to the leaves of the tree. An agent that receives such a termination message destroys its copy of the mobile agent. In some situations, however, termination cannot be guaranteed—for instance, if a message is lost.

For example, agent A receives a termination message for a job J, whereas agent B does not due to a communication error. Agent A destroys the mobile agent as requested and tries to find a new mobile agent by offering its resources to the other agents. Agent B receives this advertisement and sends a new copy of the mobile agent for job J. Thus, agent A could become a sub-node of agent B for a task that should have already been terminated. But this is prevented by a rule that sub-node building is possible only if there are work units from Job J still waiting to be distributed. Instead, agent B will terminate its mobile agent for J because the attempt to recruit A as a sub-node failed.

Similarly, the Organic Grid is able to detect failed nodes. A list contains parent nodes and, eventually, even their parent nodes. If there is no communication from the parent node after a certain amount of time, the worker node contacts the parent node, which then contacts its parent node and so forth. A communication problem can be detected as can an incomplete termination.

The adaptivity in this approach is included in the adaptive tree overlay structure. Each submitter has several child nodes that requested work units for calculation (pull mode). In addition, each submitter owns a list of active workers and a list of potential workers. Potential workers are nominated by other workers using child propagation, and each worker sends the information about its best performing sub-node to its parent node within a given time interval. The submitter uses active as well as potential workers and rates them based on the time they needed to compute r results. The performance estimation is the average number of results r in R result computation intervals. Therefore, not only the worker node itself, but the whole tree (including all its sub-nodes) is rated for its performance. If a potential worker returned the minimum required result, it is added to the list of active workers. Based on the rating of workers regarding the average duration of $r \cdot (R+1)$ results, the tree is restructured using the fastest nodes near the submitter root node. This mechanism continuously

minimises the delay between submitters and the best-performing workers.

Due to the limited size of the active node list, the slowest child is deleted after each performance rating and added to the list of former child nodes. This node is ignored for a certain time interval, which means that pulls for work as well as child propagation messages are ignored. This minimises the threat of sudden structure changes or even change loops that might occur if two nodes are similarly slow and would continuously replace each other in the active node list.

Due to the adaptivity and pull mode strategy, the Organic Grid is a promising candidate for the decentralised scalable scheduling of bag-of-tasks applications. Nonetheless, Organic Grid misses the notion of trust and therefore its suitability in open systems with misbehaving agents has to be further evaluated. Therefore, Organic Grid fulfills most, but not all of the requirements we have for our application scenario. Therefore, in Section 7.7.2 we will compare Organic Grid to the approaches presented in this thesis.

In order to compare Organic Grid to our approaches, some adaptations have to be made in order to translate the mobile agents concept as well as the pull mode into the environment of our application scenario.

3.5 Scheduling and Matchmaking

While the previous section presented different desktop grid approaches in terms of adaptivity or trust aspects, this section now concentrates on the different matchmaking approaches in such grid systems. We will analyse, in which way the techniques and mechanisms introduced here are suited for the requirements introduced in Section 3.3.

Scheduling and matchmaking are used in different ways in the literature. In [27], *scheduling* is defined as the assignment of jobs or work units to resources. [28] presents different scheduling strategies that refer to the processing of jobs in a scheduler. This definition of scheduling concentrates particularly on the temporal order of work units on the resource. In contrast, *matchmaking* defines which work units are given to which agent. In this thesis, we use the definition of [28], and define the question of which work units are given to which agents, as matchmaking.

Definition 2. Scheduling:

Scheduling is the temporal organisation of the computation of work units on a resource.

Definition 3. Matchmaking:

Matchmaking is the allocation of work units to resources.

Matchmaking can also be a part of a scheduler component. The two mechanisms are not always clearly distinguishable.

Due to the different definitions of scheduling and matchmaking, we will present the matchmaking strategies that we are interested in and briefly introduce scheduling mechanisms that contain interesting matchmaking aspects. Possibly, we could use certain aspects from the literature to improve our trust-enhanced self-organising mechanisms.

3.5.1 Scheduling Mechanisms

In this section, scheduling mechanisms are presented and analysed with respect to their embedded matchmaking strategies and regarding our requirements introduced in Section 3.3.

Application-specific Scheduling for the Organic Grid

In [29], the scheduling mechanisms of the Organic Grid (Section 3.4.6) are presented. In addition to bag-of-tasks applications, this paper now also uses matrix operations as jobs with causal dependencies. In this type of application, a causal dependence between work units requires further communication among the members of the desktop grid. In order to fit the Organic Grid to the special needs of this class of applications, and in addition to the tree overlay network, a torus overlay ensuring efficient data access for the matrix operations is introduced. This scheduling extension goes beyond the requirements of the bag-of-tasks applications considered in this thesis, but the basic scheduling mechanisms of the Organic Grid without extensions are a worthwhile field of comparison for our results.

Agent-Based Autonomous Scheduling Mechanism Using Availability in Desktop Grid Systems

[30] introduces a scheduling mechanism that includes the availability information of the members in order to achieve efficient mapping between tasks and workers. The overlay network is a content addressable network (CAN)[31]. organised by the logical distance of the resources. This distance is the response time between coordinator and worker. The coordinator is an instance that mediates between submitters and

workers. A work unit is transferred to a worker as a mobile agent that performs the calculation using the worker's resources. The workers are classified by their reliability, and the replication of work units is adapted to the expected reliability. Adapting the replication factor to the agents' reliability is an interesting idea, but there might still be room for improvement given the possibility that agents might misbehave during scheduling. due to the missing consideration of trust, this approach does not fulfill our requirements.

An Adaptive Decentralized Scheduling Mechanism for Peer-to-peer Desktop Grids

Based on a framework presented in [24], [32] introduces a scheduling mechanism that ranks the available workers by their attributes and compares these attributes to the requirements of the work units. The requirements are derived from statistical analysis of former tasks on a resource. Trust is not considered in this approach, which makes its suitability for completely open systems questionable. However, this approach is currently not well-suited to the requirements of our system.

A Scheduling and Certification Algorithm for Defeating Collusion in Desktop Grids

[33] presented a scheduling approach that recognises malicious and even colluding nodes based on replication and result verification. As it is based on the centralised BOINC (Section 3.2.1) system, this approach is not suited for our decentralised application scenario.

Fault-tolerant Dynamic Job-scheduling Policy

[34] presents a dynamic approach to fault-tolerant scheduling. This approach uses a hierarchical organisation of schedulers on N levels. On the top level N, a grid super scheduler is used; on the lowest level 0, several local schedulers are implemented. Level 1 is used to distribute jobs and the levels on top are used for load balancing. As the grid scheduler is a centralised component in the system, this scheduling mechanism does not fulfill our requirement of decentralised management.

Result Verification and Trust-based Scheduling in Peer-to-peer Grids

Zhao [35] presents a trust-based approach to scheduling that includes result verification techniques. Each grid member owns a task queue, which includes work units

that need to be processed, a scheduler, and a verification system. Work units are given to the most trustworthy agents. The work units are usually replicated and, additionally, a quiz is distributed. The quiz is a 'test work unit' that cannot be distinguished from other work units, but that can be validated for the submitter. Based on the results of the quiz calculations, the trust value is updated, and the replication is adapted to the trust value in order to minimise replication overhead. The scalability of this approach is only given if there are not too many test packages in the system. As soon as there are many potentially misbehaving agents, both the replication rate and the number of test packages increase, which leads to an increased workload. Although the combination of trust and adaptivity is promising, the scalability issues make this scheduling approach suboptimal regarding our requirements.

Cluster Computing on the Fly: P2P Scheduling of Idle Cycles in the Internet

The scheduling of WaveGrid [36] works as a CAN-based system that supports different classes of applications (work piles, point-of-presence applications, and daytime-based bag-of-tasks scheduling [37]). It is not suited for the scheduling needs of our application scenario due to the lack of trust consideration and doubts regarding the scalability of the overall system using this scheduling approach.

Intelligent Agent-based Scheduling Mechanism for Grid Service

An approach based on grid services is presented in [38]. Similar to [34], this approach uses hierarchical schedulers, but concentrates on service-oriented architecture (SOA). Due to the different application class (SOA instead of bag-of-tasks), this approach is not suited for our needs.

Supporting Self-organization for Hybrid Grid Resource Scheduling

An interesting hybrid approach combining institutional and desktop grids is presented in [39], but is not relevant for our application scenario which requires a decentralised solution.

A Hybrid Policy for Job Scheduling and Load Balancing in Heterogeneous Computational Grids

A scheduling and load balancing strategy for grid systems is presented in [40]. The grid system here is decomposed into sites of homogeneous clusters, and the job distribution concentrates on load information. Due to the centralised aspects and the classification into homogeneous subsystems, which is not possible in our adaptive, dynamic application scenario, this approach is not suited for our purposes.

Having provided this overview of scheduling mechanisms, which include matchmaking aspects, we will now present 'pure' matchmaking strategies for desktop grids.

3.5.2 Matchmaking Mechanisms

This section presents the state of the art of matchmaking mechanisms, which concentrate on the allocation of work units to workers. These mechanisms are analysed regarding our requirements 3.3. Here, we concentrate on the matchmaking strategies that are suited for bag-of-tasks applications. Therefore, we leave out matchmaking for SOAs.

Matchmaking with Limited Knowledge of Resources on Clouds and Grids

Melendez [41] presents a matchmaking strategy based on the reservation of resources in order to fulfil deadlines. Based on the start time S_i, deadline D_i and calculation time E_i, the laxity $L_i = D_i - S_i - E_i$ is calculated. According to the Any-Schedulability theory, which is presented in the paper, a set of requests is distributable if the laxity for each request i is larger than or equal to the worst case delay of other overlapping requests. Due to the central resource broker necessary for the matchmaking, this approach is not suited for our requirements.

A Matchmaking Algorithm for Resource Discovery on Grid

[42] and [43] present a tree-based matchmaking approach. The tree structure is based on the attributes of the available nodes. The left part of the tree stores idle nodes, the right one occupies resources. Attributed are, for instance, the architecture (Intel, AMD) and the operating system (Windows, Linux, Solaris). The nodes of the tree store table IDs, which point to tables including the resources that fulfil the attributes.

If a worker is needed, the left part of the tree is processed according to the attributes necessary for the job and its work units. The table of available suitable

workers is then retrieved. The search time for d levels, n nodes and r table entries is $O(d+n+r)$.

The presented approach is interesting for resource matching, but the actual resource selection (which criteria are used to select the most suitable agent from the table delivered by the mechanism) is not presented. In addition to that, the approach is not trust-based and therefore, the ability to cope with misbehaving agents is questionable.

Matchmaking: Distributed Resource Management for High Throughput Computing

Based on a central matchmaker, Raman [44] presents a strategy that matches the work units' resource requirements and available resources. Due to the central component, this approach is not suited for us.

Distributed Grid Resource Discovery with Matchmakers

In [45], a grid is viewed as a set of distributed matchmakers. Each of these matchmakers provides its resources for calculations. All local matchmaking requests are handled by the local matchmaker, which requests information about its neighbours' availability. If the local resource is insufficient for a calculation request, the request is forwarded to neighbouring matchmakers.

Since the presented approach lacks the handling of misbehaviour in open systems(e.g. using trust), it is not a suitable matchmaking strategy for our application scenario.

A Peer-to-peer Approach to Resource Location in Grid Environments

A strategy for request forwarding is presented in [46]. In particular, four different strategies for forwarding are compared. These include random forwarding, learning, best neighbour, and a combination of learning and best neighbour. The idea of immediate updates for resource information changes is interesting, but as there is again no model of misbehaviour handling, it is not a suitable approach for our needs.

Condor Distributed Scheduler

The scheduling strategy of Condor, including matchmaking strategies, is presented in [47]. It is based on a central manager for resource management and matchmaking.

Due to this central component, it is not a suitable candidate for our application scenario.

Similarly, the job distribution based on a centralised broker presented in [48], is unsuitable due to its central instance.

Centralized versus Distributed Schedulers for Multiple Bag-of-task Applications

Beaumont [49] presents four scheduling approaches. In addition to a simple first-come first-served heuristic, two bandwidth-based models that prioritise the work units according to the bandwidth between submitter and worker, are presented. The coarse-grain bandwidth-centric (CGBC) approach accumulates smaller work units to macro tasks and sends these to workers with a high bandwidth connection. The parallel bandwidth-centric approach reduces communication by ensuring that only threads of the same task class communicate with each other and then build a tree structure in order to reach fairness in task allocation with respect to bandwidth.

The fourth heuristic is called data-centric. It uses the bandwidth-based approach for data-intensive jobs.

In general, a bandwidth-based model for communication and data intensive tasks is a promising idea. In our application scenario, however, the general focus is not on bandwidth usage alone, but on a fair and efficient scheduling in spite of misbehaving nodes, which is not covered by this approach.

Towards Bidirectional Distributed Matchmaking

[50] presents a distributed matchmaking strategy that is based on a bidirectional process. In most classical grid strategies, workers wait passively for work units whereas submitters search actively for workers. In contrast, [50] makes all agents a part of the resource-finding process by providing each agent with an address list of other agents, which is used to find suitable workers. If an agent searches for a worker, it sends this information to all agents on this list. The agents then flood the request across the network. As soon as a suited worker is found, this information is backpropagated.

In order to reduce the communication overhead caused by broadcast, Teeming and Time To Live (TTL) are used, although both approaches increase the time until a match is found. Teeming only sends a request with a certain probability. The TTL maximises the number of hops that the request is allowed to take before it is dropped. Additionally, matching caching lists are introduced as further improvement; agents

store the last unsuccessful requests and their types, making less successful routes in the network less likely for a certain amount of time.

As we wand to abstract from neighbourhood information, prevent the system from scalability issues caused by communication channels based on broadcasting and concentrate on matchmaking in disturbed agent situations, this approach is not considered in this thesis.

An Integrated ClassAd-Latent Semantic Indexing Matchmaking Algorithm for Globus Toolkit-based Computing Grids

Montella [51] introduces a further broker-based approach, which due to its central instance is not suited for our requirements.

Resource Discovery Techniques in Distributed Desktop Grid Environments

Kim [52] introduces two strategies, which map the matchmaking problem into a routing problem.

The members of the grid are organised as a *rendezvous node tree*. Nodes are organised by their available resources. Each submitter starts with its sub-tree when searching for a worker. If there is no suitable candidate in the sub-tree, the search process continues at the higher level. If there is more than one suitable worker candidate, the worker with the lowest workload is chosen.

The second approach presented is based on a CAN overlay network based on the abilities of the available nodes. Due to the multi-dimensional routing, agents (and jobs) can be distinguished by their CPU power (power consumption) and available RAM (RAM usage). The overlay is separated into zones and each zone has a manager that receives the tasks for the zone and is responsible for the allocation of tasks to resources in this area and the maintenance of the area.

According to the authors, the CAN-based matchmaking led to better results and has been further improved in [53] and [54]. This might hold for applications which allow for a variety of points in the CAN space, but in our scenario, the agents constantly change their abilities (available computational power, bandwidth, trust). Therefore, the overlay structure would continuously have to be updated, which would cause management overhead. Therefore, this approach is not well suited for our requirements.

GridP2P: Resource Usage in Grids and Peer-to-peer Systems

[55] developed a desktop grid matchmaking approach based on a Pastry overlay network. The management functions of the system are organised in different layers that are known to all members. The overlay network is used to locate resources and is continuously updated. The updated information includes sender information (e.g. ID), supported applications and available resources (e.g. CPU, bandwidth, RAM, HDD). According to this information, agents calculate the availability of each node. A node matches a resource request of a work unit if it has more resources available than is required by the work unit. Therefore, a resource usage estimation has to be performed.

The overlay network in this publication also allows for the distributed storage of calculation results using replication. This is especially useful in highly dynamic environments where agents have a high probability to leave the system without warning.

Similar to the approach presented in the subsection above, this approach is only as good as the overlay structure which can be build using the available agent information. due to the high dynamics of the system, this overlay structure would decrease the scalability of the system due to increased management overhead by continuous restructuring. Therefore, the approach is not suited for our application scenario.

3.6 Taxonomy of Grid Systems

The overview of scheduling and matchmaking strategies has shown a variety of approaches tailored to the needs of a desktop grid. Nonetheless, there are tremendous differences between the approaches. In order to differentiate between this large set of possibilities, we used our own requirement analysis in combination with the taxonomy of grid scheduling techniques proposed in [56]. Due to Choi's definition of scheduling, he also included matchmaking in the taxonomy. Therefore, we can use his approach in combination with the requirements we determined in Section 3.3. Since many approaches could already be eliminated due to central instances or lack of agent misbehaviour consideration, we can simplify the taxonomy here by merely comparing the suitable approaches of H-Trust and the Organic Grid. Both approaches are promising candidates, but miss one requirement: Whereas H-Trust is trust-based, but not adaptive, Organic Grid shows adaptivity, but does not consider trust information. Moreover, there is no known approach in literature which com-

bines trust-consideration, adaptivity for throughput/workload optimisation and the consideration of agent capabilities in the distributed matchmaking and scheduling strategies.

Therefore, we come to the conclusion that the state of the art of grid systems, scheduling and matchmaking mechanisms does not include a system which fulfills all our requirements. Therefore, we decided to build our own application scenario grid system, the Trusted Desktop Grid (TDG).

		TDG	H-Trust [57]	Organic Grid [6]
Organisation	central			
	distributed	×	×	×
	hierarchical			
Mode	push	×	×	
	pull			×
Heuristic	Capability	×		
	Throughput			×
	Workload	×		
	Reputation/Trust	×	×	

Table 3.1: Attributes of approaches

Table 3.1 compares the main aspects of the taxonomy, according to Choi [56], of TDG, H-Trust and Organic Grid. All three systems fulfil the basic requirements of decentralised system architecture, which allows for scalability and robustness.

TDG and H-Trust rely on the push mode, which means that submitters ask workers for work unit calculations. In contrast, Organic Grid uses the pull mode—i.e. the workers advertise that they have spare computing resources. This mode makes evaluation harder because the misbehaving agents need to be 'translated' into this mode.

The heuristics in Table 3.1 are condensed to those used by at least one of the approaches.

3.7 Discussion of Grid Systems

The analysis of the state of the art has shown that the combination of agent adaptivity and trust as realised in the TDG is unique as here is no comparable approach. Nonetheless, two approaches are supposed to show similar benefits: H-Trust, which

makes matchmaking strategies based on trust and reputation information, and Organic Grid, which shows a high degree of adaptivity. Therefore, we want to compare both approaches with the TDG in the evaluation (Chapter 7).

Overall, we can say that, except for Organic Grid and H-Trust, self-organisation as a technique for efficient, robust scheduling in open desktop grids, is not considered in the literature. Most existing systems as well as scheduling and matchmaking techniques are unsuitable for systems where disturbances in terms of misbehaving agents can occur.

3.8 Trust and Reputation Models

In this section, trust and reputation mechanisms from literature are presented. This section concentrates on trust management systems, which receive trust information from the interaction of their entities (e.g. agents).

Definition 4. Trustworthiness:

Trustworthiness, the capacity to commit oneself to fulfilling the legitimate expectations of others, is both the constitutive virtue of and the key causal precondition for the existence of any society.[58]

In addition to locally measured trust, we also relate it to reputation.

Definition 5. Reputation:

Reputation is a global aggregation of locally measured trust values.

We will analyse the different features of the trust and reputation models, and formulate those aspects that are useful or interesting for usage in the TDG as criteria.

In the following section, we will introduce the requirements we need for a trust and reputation mechanism suited for our application scenario.

Based on these criteria, we will later in this chapter analyse whether there are systems that can be used as trust and reputation system of the TDG or if it is necessary for us to build our own trust and reputation system based on the aspects we have learned from the analysis of the state of the art.

If not defined differently, the scale of the trust values is in the interval [0, 1], where 0 represents the lowest trust value and 1 the highest.

3.8.1 Requirements and Criteria

The analysis of the state of the art of trust and reputation literature delivered criteria that are important for the usage of a trust and reputation model in the TDG. These criteria are listed here and their functionality is explained in such way that different implementations of this functionality can be analysed in the following.

1. **Criterion**: Differentiation of service trust and feedback trust.
2. **Criterion**: Differentiation of service trust in direct trust and recommendation trust.
3. **Criterion**: No trust increase for re-entering the system.
4. **Criterion**: Consideration of information timeliness using a decay factor.
5. **Criterion**: Higher costs for trust increase than benefit from trust decrease.

ad 1: The first criterion is the differentiation of service trust and feedback trust. The trust in the ability of an agent to fulfil a given task (e.g. compute a work unit in the TDG scenario) is separated from its ability to give honest ratings and, thus, good recommendations. The agents act in two different roles (service provider and feedback provider) and their performance in one role need not reflect their performance in the other role. This leads to a class of attacks that is based on unfair ratings and is less likely to succeed [59]. In the literature, feedback trust is often referred to as credibility and can appear in two ways: (1) from supervising the trust value of a rated interaction and the interaction result itself or (2) from using the similarity between agent and feedback trust provider.

ad 2: The second criterion is the differentiation of service trust in direct trust and recommendation trust. Direct trust is trust based on the agent's own experiences with another agent in the past. Recommendation trust consists of the trust information of other agents that have had interactions with an agent. Separating both values enables agents to adjust the weightings based on the available information. If an agent has built direct trust based on many interactions, it can rely on this information and, thus, reduce the threat of unfair rating attacks. If it had only a few interactions or if its interactions are older, it can rely on the recommendations of other agents, thereby reducing its information uncertainty. Collecting recommendations from several agents makes discrimination attacks less likely.

ad 3: An important criterion is to prevent trust increase by re-entering the system. This counters re-entry attacks on the trust management mechanism. The

easiest way to account for this criterion is to initialise all trust values with minimal trust [60]. In the TDG, the trust adaptivity of the agents needs a minimal amount of trust; therefore, agents are initialised with a medium trust value and we rely on authentication techniques whereby agents are recognised while re-entering the system.

ad 4: Considering the timeliness of information is a criterion that accounts for the dynamics of the system, which can also result in changing trust relations over time. The older a piece of information is, the less relevant it can be for the current state of the system. Differentiating information by their timeliness enables the system to forget former actions and forgive [61] agents that have improved their behaviour. This also incentivises agents to behave well and, thus, prevents the system from being harassed by playbook attacks. [62]

ad 5: The last criterion is that the costs of trust increase should be higher than the benefits of trust decrease. This again incentivises agents to behave cooperatively and, thus, the trust management system supports a good system performance.

Besides the criteria mentioned above, the analysis of the state of the art led to further aspects that are candidates for implementation in the TDG. For instance, trust abuse or misuse factors, which store all violations without being influenced by the decay factor, can be used to analyse system states, but would not be applicable in the TDG, where we aim at self-organised adaptivity. Therefore, in this analysis, we concentrate on the five most important criteria for a trust and reputation system to meet our needs for the TDG.

In the following, the state of the art of trust and reputation models is presented and analysed regarding our requirements.

3.8.2 Valuation of Trust in Open Networks

One of the earliest approaches to the formal usage of trust in open systems is [63] (1994). The general model is that entities are connected via links and able to send messages using these links. The application scenario here is an authentication system, in which an entity A can ask an authentication server AS if an entity B is trustworthy. Based on the trust value delivered by AS, A can decide whether the risk of trusting B for an interaction should be taken.

In order to determine the trust value, this model distinguishes between *direct trust* and *recommendation trust*. *Direct trust* is a value resulting from the direct interaction of two entities as shown in equation 3.1.

$$A\ trusts^{seq}\ B\ value\ v \tag{3.1}$$

Here, *seq* represents the sequence of entities, which is the way how information is forwarded.

Direct trust between entity A and entity B is measured by the value v. It is calculated using the *positive experiences* p, that A has had with B and uses the decay factor α.

$$v(p) = 1 - \alpha^p \tag{3.2}$$

Value v can be regarded as the probability that entity B will have a higher reliability for fulfiling a task than will A. If A encounters a negative experience with B, it will set the direct trust to 0 because the trust relationship has been broken.

A *recommendation trust* relationship exists if A accepts reports about a cooperation partner and rates their credibility. For instance, the credibility of reports from AS about B result in recommendation trust.

$$A\ trusts.rec^{seq}\ AS\ when.path\ S_p\ when.target\ S_t\ value\ v_r \tag{3.3}$$

This is represented in Equation 3.3, where S_t represents the *target constraint set*—i.e. the entity concerning the recommendation trust (e.g. entity B). S_p describes the *path constraint set*, which is the path of the information about entity B. This path is used if AS has no information available and needs to request it from another authentification server. The *recommendation trust* value v_r then consists of the positive and negative interaction results along with the entity to be recommended (e.g. B) as shown in Equation 3.4.

$$v_r(p,n) = \begin{cases} 1 - \alpha^p, & \text{if } p > n \\ 0, & else \end{cases} \tag{3.4}$$

Both trust components are then combined to form the actual trust value. This approach demonstrates a strict rule regarding negative experiences as well as the ability to rely on the recommendations of other agents. Separating the actual experiences (service trust) and the credibility to make recommendations (feedback trust) is a useful measure for taking new information into account without blindly 'trusting' the information's owner. This is especially useful in systems where misbehaving agents might be a threat, and giving wrong recommendations is a misbehaviour strategy. Therefore, we use the separation of service trust and feedback trust as a criterion that is useful for the TDG.

Criterion: Separation of Service Trust and Feedback Trust.

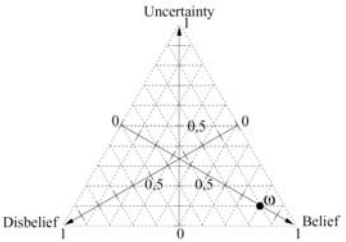


Figure 3.1: Opinion space [64]

3.8.3 A Metric for Trusted Systems

Jøsang [64] (1998) provides a mathematical definition of trust. Any event can either happen or not happen. Since knowledge (especially in open systems) is mostly limited, we are unable to make a unique statement if it occurs. Instead, there is an *opinion* about the occurrence of an event. This opinion consists of the *belief* and the *disbelief* regarding whether the event occurs as well as the *uncertainty* of this statement. These components span the opinion space as shown in Figure 3.1.

An opinion is defined as $\omega = (b, d, u)$, where ω describes the opinion, b the belief, d the disbelief and u the uncertainty. Belief, disbelief and uncertainty sum up to a value of 1.

Using subjective logic, different opinions can be concatenated and united. It is also possible to add other's experiences using recommendations. Here, direct trust and recommendation trust are separated (this is further pointed out in a later publication by the author), which is important because trust in the occurrence of an event is a subjective opinion. This definition of trust was novel, especially at the time it was published, but still holds. The separation of direct trust and recommendation trust is also important in the context of the TDG and, thus, is a further criterion to us.

Criterion: Separation of Direct Trust and Recommendation Trust.

3.8.4 The EigenTrust Algorithm for Reputation Management in P2P Networks

In [60] (2003), an $n \times n$ matrix is built for all n peers of a peer-to-peer network, based on which a global trust vector is calculated. As described in Equation 3.5,

each entry s_{ij} of the matrix describes the sum of transaction ratings t_{ij}, which i as truster assigns to j as trustee.

$$s_{ij} = \sum t_{ij} \tag{3.5}$$

Positive experiences between two peers are rated as +1 and negative experiences as -1. The trust value is the normalised sum of positive and negative ratings c_{ij} 3.6.

$$c_{ij} = \frac{max(s_{ij}, 0)}{\sum_j max(s_{ij}, 0)} \tag{3.6}$$

If a peer i wants to acquire more information about a peer k, it asks the former cooperation partner j's row for k'c column. The entries about k are weighted with i's trust towards j and then added to an aggregated trust value (3.7).

$$t_{ik} = \sum_j c_{ij} * c_{jk} \tag{3.7}$$

According to the algorithm, i is supposed to not only ask his known peers, but also their known peers and so forth. Using this flooding approach, a global trust vector for all peers can be derived from local trust values. New peers begin with an empty row in the matrix, so that it is not worthwhile for agents to leave and re-enter the system. This is important in open systems and thus represents a further criterion.

Criterion: No trust bonus for re-entering the system.

3.8.5 A Reputation-based Trust Management System for P2P Networks

In the trust management system presented in [65] (2004), a peer stores the trust values for each other peer in a bit vector. Additionally, an integer variable is needed, which indicates the size of the vector. The size starts with 8 bits, but can be increased to 16 bits, 32 bits and so forth. For each positive interaction, a 1 is shifted to the most-significant bit (left); for each negative interaction, the value shifted is 0. In order to determine a trust value between 0 and 1, the trust vector is regarded as an m-Bit binary and divided by 2^m. Using the complement of the vector, a distrust value can be computed. If there is not enough local information about a peer j, a request is sent to neighbouring peers. These peers send their trust vectors as *credibility* vectors. The entries t_i of this credibility vectors are weighted with the own credibility value c_i towards the sending peers. This results in the trust value t_j of the formerly unknown peer j according to (3.8), with k being the number of requested peers.

$$t_j = \frac{\sum_{i=1}^{k} c_i * t_i}{k} \tag{3.8}$$

This approach uses only limited memory, but has no support for adaptive agent behaviour.

3.8.6 PeerTrust: Supporting Reputation-based Trust for P2P Electronic Communities

A trust model featuring two different ways of computing the feedback trust of a peer is presented in [66] (2004). The trust value $T(u)$ of a trustee u consists of several components as depicted in (3.9).

$$T(u) = \alpha * \sum_{i=1}^{I(u)} S(u,i) * Cr(p(u,i)) * TF(u,i) + \beta * CF(u) \tag{3.9}$$

Rating $S(u,i)$ represents the rating of the i-th interaction with peer u. $I(u)$ is the number of interactions. $TF(u,i)$ is the *transaction context factor*, which can be used to differentiate between different types of interactions. Thus, larger transactions can be rewarded with a high influence on the trust value, whereas smaller tasks only have a small impact on the trust value. $CF(u)$ is the *community context factor*, which is used to incentivise agents to give feedback. The *feedback credibility* $Cr(v)$ is used to determine the credibility of a feedback given by a peer $v = p(u,i)$; thus, it represents what we call feedback trust. $p(u,i)$ here is the cooperation partner of peer u in its ith transaction.

The first approach to the credibility of feedback trust is given in (3.10). Here, the trust in the feedback is derived from the trust in the feedback-giving peer v normalised by the trust in other peers. The idea is that untrustworthy peers give bad feedback, whereas trustworthy peers good feedback. This is crucial if we regard badmouthing attacks, where agents behave well, but give inaccurate ratings.

$$Cr(p(u,i)) = \sum_{i=1}^{I(u)} \frac{T(p(u,i))}{\sum_{j=1}^{I(u)} T(p(u,j))} \tag{3.10}$$

Therefore, a second approach to feedback trust is presented in (3.11). Here, the similarity of the asking agent to the feedback-giving agent v is considered to determine the credibility of feedback trust. In order to determine similarity, the ratings of the agents are compared using the root mean square; agents with similar

ratings are trusted as feedback providers.

$$Cr(p(u,i),w) = \sum_{i=1}^{I(u)} \frac{Sim(p(u,i),w)}{\sum_{j=1}^{I(u)} Sim(p(u,j),w)} \tag{3.11}$$

Due to the heterogeneity of the agents in the TDG, it might occur that agents rate similarly not because they behave similarly, but due to external factors like the agent society currently in the system. Therefore, both approaches to feedback trust are questionable for usage in the TDG.

3.8.7 A Dynamic Trust Metric for P2P Systems

Chang [62] (2006) aims at a trust metric for peers in a peer-to-peer system, which dynamically changes their behaviour. Such peers can, for instance, perform a playbook attack by building trust in the system and then suddenly starting to distribute malicious software or refuse cooperation. In order to protect the system from such misbehaviour, a *forgetting factor* or *decay factor* is introduced. This also corresponds to the idea of *forgiveness* [61]: agents must be able to re-enter the system if they have improved their behaviour and become more cooperative.

Former behaviour vanishes over time from the collective minds of the agents. In this publication, forgetting is realised by a decay factor ρ, which rates older ratings as being less important than fresh ones.

$$D_{ij}^{t} = D_{ij}^{t'} * (1 - \rho^{t-t'}) + e_{ij} * \rho^{t-t'} \tag{3.12}$$

As shown in (3.12), a new direct trust value D_{ij}^{t} between peer i and peer j at time t consists of the decay factor ρ, the last direct trust value $D_{ij}^{t'}$ and a new rating e_{ij}. t is the time step of the current trust calculation, t' the time of the last one. ρ is a constant, but can also be defined as a function of the trust abuse if $D_{ij}^{t'}$ and e_{ij} are very different (see (3.13)).

$$\rho = \begin{cases} \rho 1, & e_{ij} - D_{ij}^{t'} \geq -\varepsilon \\ \rho 2, & else \end{cases} \tag{3.13}$$

ε is the error factor, $\rho 2$ ((3.15)) describes the trust abuse consideration, which is stored in the *misusing trust factor* AT_{ij}^{t}.

$$AT_{ij}^{t} = \begin{cases} AT_{ij}^{t'} + D_{ij}^{t'} - e_{ij}, & D_{ij}^{t'} - e_{ij} > \varepsilon \\ AT_{ij}^{t'}, & \text{else} \end{cases} \tag{3.14}$$

$$\rho 2 = \rho 2 * \frac{c}{c + AT^t_{\mathrm{ij}}(i,j)} \tag{3.15}$$

Here, c is a constant defining the decrease of the decay factor. The aim of this approach is to detect malicious peers and to reduce their trust values as soon as possible. Trust increase is possible only slowly. This makes misbehaviour less attractive and incentivises cooperative behaviour. Therefore, the decay factor must be chosen carefully. If it is too small, agents forget too fast; as a result, malicious peers are forgiven too soon. If it is too large, the trust values are deprecated and changes in behaviour are realised too slowly.

This approach also separates direct trust from recommendation trust.

$$T^t_{ij} = \lambda * D^t_{ij} + (1 - \lambda) \sum_{r \in I(j)} \frac{Cr_{ir} * D^t_{rj}}{\sum_{r \in I(j)} Cr_{ir}} \tag{3.16}$$

Recommendation trust is added as a factor in trust T^t_{ij}. It consists of the direct trust values D^t_{rj} of known peers r, which are weighted by the *recommendation credibility* Cr_{ir}, which is the direct trust value of truster i to recommending peers k.

This trust mechanism shows two core functionalities that are important criteria for the TDG. The decay factor allows for the timeliness of trust information and, thus, allows for forgetting and forgiving. Since the costs of building trust are higher than the benefits of trust loss, it incentivises agents to behave cooperatively.

Criterion: Consideration of information timeliness using a decay factor.
Criterion: Costs of trust building are higher than benefits of trust abuse.

3.8.8 Trust Network Analysis with Subjective Logic

An extension of the above-mentioned [64] is [67] (2006). The definition of trust is modelled further by introducing direct trust, recommendation trust, and indirect trust derived from recommendation trust. Direct trust defines the trust resulting from the direct interaction of two parties, while recommendation trust is direct trust forwarded from a third party. Indirect trust results from the interpretation of recommendation trust.

The opinion space $\omega = (b, d, u, a)$ is extended by the *base rate* a. This is a weighting factor representing the uncertainty of the trust computation. If there is no information about another party, it is a measure for the initial trust.

A decay factor is introduced in [67] as well in order to implement forgiveness and forgetting in the system.

3.8.9 A New Reputation Mechanism Against Dishonest Recommendations in P2P Systems

Chang [68] (2007) extended his former model [62] by adding the separation of direct trust and recommendation trust.

The direct trust D_{ij} of two peers i and j is calculated using the *values* $e.v$ of the ratings e stored in EXP_LIST.

$$D_{ij}^{t} = \frac{\sum_{e \in EXP\_LIST} e.v * \alpha^{(t-e.t)}}{\sum_{e \in EXP\_LIST} \alpha^{(t-e.t)}} \tag{3.17}$$

α is a decay factor that determines the forgetting rate using the time when the ratings were made $e.t$ and the current time t.

The *indirect trust* R_{ij}^{t}, which determines the recommendation trust at time t, is calculated as given in Equation (3.18).

$$R_{ij}^{t} = \frac{\sum_{k} Cr_{ik}^{t} * CI_{kj}^{t} * Rec_{kj}^{t}}{\sum_{k} Cr_{ik}^{t} * CI_{kj}^{t}} \tag{3.18}$$

Rec_{kj}^{t} is the direct trust between peer k and trustee j, where k is asked for its opinion. Cr_{ik}^{t} is the *credibility* of peer k and, thus, the feedback trust. CI_{kj}^{t} is the *level of confidence* that peer k has in its own experiences. This reduces the loss of feedback trust if a recommended peer behaves not as well as expected.

$$CI_{ij}^{t} = CIN_{ij}^{t} * CID_{ij}^{t} \tag{3.19}$$

CI_{ij}^{t} depends on the number of transactions CIN_{ij}^{t} made with the recommended peer, as well as the variance in the ratings of transactions CID_{ij}^{t}.

The credibility Cr_{ik}^{t} towards peer k is given in Equation 3.20. The list REC_LIST contains all recommendation ratings e of other peers.

$$Cr_{ij}^{t} = \frac{\sum_{e \in REC\_LIST} e.V_w * \alpha^{(t-e.t)} * e.CI_w}{\sum_{e \in REC\_LIST} \alpha^{(t-e.t)} * e.CI_w} \tag{3.20}$$

$e.CI_w$ is the level of confidence that peer k has in the ratings of peer j, and α is again the decay factor. $e.V_w$ is computed as the difference between the recommendation value V_{kj} and the value V_{ij} of the agent's own trust in peer j. Thus, it is the feedback trust.

$$V_w = 1 - |V_{kj} - V_{ij}| \tag{3.21}$$

The *overall trust* O_{ij}^{t} consists of the direct trust D_{ij}^{t} and the recommendation trust R_{ij}^{t}.

$$O_{ij}^{t} = \lambda * D_{ij}^{t} + (1 - \lambda) * R_{ij}^{t} \tag{3.22}$$

λ here is a factor in the interval [0, 1], where $\lambda = 0$ means that direct trust is ignored and $\lambda = 1$ means that recommendation trust is ignored.

$$\lambda = \frac{EXP(i,j).n}{m} \tag{3.23}$$

λ is dynamically adapted to the number of transactions $EXP(i,j).n$ between truster i and trustee j. With an adjustable threshold m, peers only rely on direct trust.

This trust mechanism meets all the criteria we have determined so far, except for the different weightings of trust increase and decrease. Due to this missing functionality, we decided not to use this approach in our application scenario.

3.8.10 ICRep: An Incentive-compatible Reputation Mechanism for P2P Systems

ICRep in [68] (2007) is an extension of Chang's trust model with a *trust exchange protocol* Therefore, a *level of participation* l_{ij}^t is introduced, which determines how often a peer provides recommendation trust information (see Equation 3.24.

$$l_{ij}^t = \begin{cases} \frac{I_{ij}^t}{I_{max}}, & if\ I_{ij}^t < I_{max} \\ 1, & else \end{cases} \tag{3.24}$$

I_{ij}^t is the number of answers peer i received from peer j, and I_{max} is the threshold at which all requests are answered. This extension provides an incentive to rate agents and rating information to other agents.

3.8.11 An Affair-based Interpersonal Trust Metric Calculation Method

In [69] (2008), two tree-based data structures are built: a *trust performance tree* and an *affair behaviour tree*. The two trees relate to the different trust facets defined by the authors.

The trust performance tree consists of three main branches: *benevolence*, *integrity* and *ability*, which related to the three different trust facets of the authors. The more distant that a node is to a root, the more details about its performance are available.

The affair behaviour tree stores the affairs. An *affair* is an interaction between two parties at a certain time in a certain context. New encounters are added as new branches.

To calculate the trust values, the branches of the affair behaviour tree are mapped to the leaves of the trust performance tree. Each performance here is on a scale between -1 and 5. -1 here means no trust, 0 means uncertainty, and 1 to 5 represent

different degrees of trust. The trust degree TD is calculated level by level from the leaves to the root and then the three main branches are added as a weighted sum (Equation 3.25).

$$TD = BTD * TC_1 + ITD * TC_2 + ATD * TC_3 \qquad \sum_{i=1}^{3} TC_i = 1 \tag{3.25}$$

Here, BTD is the *benevolence trust degree*, ITD the *integrity trust degree* and ATD the *ability trust degree*. TC_i is the weighting factor.

This model was initially designed for the rating of relationships between human beings. In a trust-based desktop grid like the TDG, the number of different encounters is not large enough to make a tree structure worthwhile. Moreover, the scalability of a system where each participant stores two tree structures of a large network is not sufficient.

3.8.12 Reputation System User Classification Using a Hausdorff-based Metric

The *StarWorth* algorithm presented in [70] (2008) is an approach to filter irrelevant offers from a pool of resources from other users for a user. The value of a resource is given on a worth scale from 0 to 5, where 0 is a useless/unsuited resource and 5 ($MaxWorth$) an extremely well-suited resource offer. An explicit rating of a resource of a user U_j by a user U_i is called $WE\_UU(U_i, U_j)$. If the value has been determined implicitly using the recommendation of other users, we call the rating $WI\_UU(U_i, U_j)$ (see Eq. 3.26). This corresponds to the separation of direct trust and recommendation trust.

$$WI\_UU(U_i, U_j) = \frac{\sum_{l=1}^{k} WU(U_i, U_i^l) * WE(U_i^l, U_j)}{k * MaxWorth} \tag{3.26}$$

U_i^l are the users asked by truster U_i about trustee U_j. k is the number of users that have been asked, $WE(U_i^l, U_j)$ is the users' explicit rating of U_j.$WU(U_i, U_i^l)$, and the rating of U_i about a user U_i^l is used as a weight.

Similarly, the reputation of a user in the system can be computed by extending the implicit worth value to the opinion of all users.

3.8.13 TwoHop: Metric-based Trust Evaluation for Peer-to-peer Collaboration Environments

TwoHop [71] (2008) received its name due to reduction of the time-to-live (TTL) of recommendation trust requests to two hops.

Each peer maintains a *collection* C, which contains for each *service* S of a *service provider* i a *portfolio* P.

$$C = (P_j, ..., P_k), \qquad j, k \in S, \qquad j \neq k \tag{3.27}$$

A portfolio (Eq. 3.28) consists of identity i of the providing peer, the *value* v, *assessment* value a and the *review* value r.

$$P_s = (i_0, r_0, a_0, j_0), ..., (i_n, r_n, a_n, j_n), \qquad n \in \mathbb{N}, \qquad s \in S \tag{3.28}$$

As the value is the result of the evaluation of a situation, the assessment judges the evaluation, and the review surveys this rating. To compute the trust value T of a service s, a peer uses its own available data in its root portfolio P_{root}, and sends a recommendation trust request to its direct neighbour j. This is used to determine the *OneHop* portfolio. Within this portfolio, all direct evaluation entries of the service as well as the assessments of the evaluating peers are selected. The found evaluations of a peer j are weighted with the assessment for j from P_{root} and added to sum_{trust} as per equation 3.29.

$$T = \frac{sum_{trust}}{sum_{weights}} = \frac{\sum_{j=1}^{V_{i,s}} w_j * v_j}{\sum_{j=1}^{V_{i,s}} w_j} \tag{3.29}$$

$$w_j = \begin{cases} a_j & if\ v_j \in OneHop \\ r_j * a_j & if\ v_j \in TowHop \end{cases} \tag{3.30}$$

Here, $V_{i,s}$ is the set of all collected evaluations of a service s of provider i. The assessments about a service found in the OneHop portfolio are used as pointers for new trust requests, whereas only peers are asked for information that is not already known in P_{root}.

The results of the second request wave are stored in the *TwoHop* portfolio, which is again analysed with respect to the evaluations of service s. Found entries are combined with value v_j, the review value of P_{root} and the assessment value from OneHop to sum_{trust}. If several paths lead to the same evaluation value, the shortest path is chosen.

The approach is interesting and scalable, but not suited for our application scenario, which abstracts from neighbourhood information.

3.8.14 SFTrust: A Double Trust Metric-based Trust Model in Unstructured P2P System

Zhang's [72] (2009) trust mechanism *SFTrust* defines service trust, feedback trust, direct trust and recommendation trust as we use the terms throughout the thesis.

The direct trust DT_{ij}^n (Eq. 3.31) between peer i and peer j is calculated using the service values S_{ij} of peers j. λ is a decay factor in the interval [0, 1].

$$DT_{ij}^n = \frac{\sum_{k=0}^{n} \lambda^k S_{ij}^k}{(1-\lambda^n)/(1-\lambda)} \tag{3.31}$$

n is the number of transactions. To update the direct trust value, the subsequent direct trust value (see Eq. 3.32) can be calculated with the new service and the former direct trust value, which decreases the storage needs of the approach.

$$DT_{ij}^{n+1} = \frac{DT_{ij}^n * (1-\lambda^n) + \lambda^{n+1}(1-\lambda) * S_{ij}^{n+1}}{(1-\lambda^{n+1})} \tag{3.32}$$

The recommendation trust RT_{ij} (Eq. 3.33) is computed from the recommendations of peers from a neighbour list NL. Therefore, each peer i sends a neighbour request to several peers when it enters the system. The request is accepted if i's trust value is high enough. Therefore, all new peers have a medium initial trust value.

$$RT_{ij} = \frac{\sum_{k=1}^{m} FT_{ik} * f_{kj}}{\sum_{k=1}^{m} FT_{ik}} \tag{3.33}$$

f_{kj} is the service trust value that truster i receives from neighbouring peers k about trustee j. FT_{ik} is the feedback trust value, which depends on the cosinus similarity (Eq. 3.34) of truster i and neighbour k.

$$Sim(i,k) = \frac{\sum_{l \in CS(i,k)} f_{i,l} * f_{k,l}}{\sqrt{\sum_{l \in CS(i,k)} f_{i,l}^2} * \sqrt{\sum_{l \in CS(i,k)} f_{k,l}^2}} \tag{3.34}$$

The *common set* CS includes all peers l that have rated trustee j and have been rated by truster i and neighbour k.

$$FT_{ik}^{n+1} = \begin{cases} FT_{ik}^n + \eta * (1 - FT_{ik}^n), & Sim(i,k) > Sim_\theta \\ FT_{ik}^n, & Sim(i,k) = Sim_\theta \\ FT_{ik}^n - \theta * (1 - FT_{ik}^n), & Sim(i,k) < Sim_\theta \end{cases} \tag{3.35}$$

Factors η and θ realise the different increase and decrease of the feedback trust (Eq. 3.35) with a corresponding increase or decrease in trust.

SFTrust is an interesting approach because it combines the similarity-based feedback trust calculation of *PeerTrust* [66] with the general mechanics of the *ICRep* algorithm.

As the trust values of new peers are set to a high value, the model is vulnerable to re-entry attacks. Additionally, the number of transactions is not included in the trust relationship evaluation; thus, the credibility of the trust values is not always given.

3.8.15 RATM: A Reputation-based Attack-resistant Distributed Trust Management Model in P2P Networks

The *RATM* trust management model presented in [73] (2010) also distinguishes direct from indirect trust.

The direct trust value $D_n(i,j)$ (Eq. 3.36) is computed from the contentedness with a transaction $f(i,j)$ and is averaged by the number of transactions m.

$$D_n(i,j) = \begin{cases} \frac{\sum_{k=1}^{m} f(i,j)}{m}, & m \neq 0 \\ 0, & m = 0 \end{cases} \tag{3.36}$$

The *indirect trust*—i.e. recommendation trust—consists of the direct trust values $D_n(m,j)$ delivered by peers m and the *credibility* Cr_{im} (i.e. feedback trust).

$$R_n(i,j) = \frac{\sum_m D_n(m,j) * Cr_{im}}{\sum_m Cr_{im}} \tag{3.37}$$

The feedback trust depends on the received recommendations $R_n(i,j)$ (Eq. 3.37) and the agent's own direct trust $D_n(m,j)$ in the feedback-giving peer.

$$\varepsilon = |R_n(i,j) - D_n(m,j)|/s_{ij} \tag{3.38}$$

The dynamics of the feedback trust change ε (Eq. 3.38) and are computed using the standard deviation s_{ij} of the recommendation trust calculation.

$$Cr_{i,m}^{k} + 1 = \begin{cases} Cr_{ij}^{k} + \delta(1 - Cr_{ij}^{k})(1-\varepsilon), & 0 \leq \varepsilon \leq 1, k > 0 \\ Cr_{ij}^{k} - \gamma Cr_{ij}^{k}(1 - (1\varepsilon)), & \varepsilon > 1, k > 0 \\ 1/2, & k = 0 \end{cases} \tag{3.39}$$

In order to have different weights for trust increase and decrease, δ and γ are used.

Direct trust and recommendation trust are aggregated (Eq. 3.40) to a peer trust value, which corresponds to our definition of service trust.

$$PT_n(i,j) = \alpha * D_n(i,j) + (1-\alpha) * R_n(i,j) \tag{3.40}$$

α in [0,...,1] is another weighting factor.

RATM introduces further values to investigate trust abuse. The *trust deviation value* P_{ij} (Eq. 3.41) computes the previous deviation of a peer's trust ratings.

$$PT_{i,j} = \sqrt{\frac{\sum_k^{maxTZ}(g(k)(D_k(i,j) - R_k(i,j))^2)}{\sum_k^{maxTZ} g(k)}} \tag{3.41}$$

Here, k marks the beginning of the time interval, and $maxTZ$ marks the end. A *time fading function* $g(k)$ (Eq. 3.42) is introduced in order to make old ratings less important than new ones.

$$g(k) = \rho_{fade}^n - k \tag{3.42}$$

Here, n is the current time interval and ρ the *fading factor*.

With the information about the trust deviation P_{ij}, the*trust abuse value* Q_{ij} (Eq. 3.43) is computed.

$$Q_{ij} = \frac{\sum_{k=1}^{maxTZ}(g(k) * max(0, R_k(i,j) - P_{ij} - D_k(i,j)))}{\sum_{k=1}^{maxTZ} g(k)} \tag{3.43}$$

As the trust abuse value includes all encounters of abusive behaviour, negative behaviour is never completely forgotten. In order to prevent (or at least punish) oscillation, in Equation 3.44, the peer trust value is updated using these two factors.

$$PT_n(ij) = \beta PT_n(i,j) - (1-\beta)(\mu P_{ij} + \tau Q_{ij}) \tag{3.44}$$

Therefore, μ and τ are used as weights for trust deviation and trust abuse. Factor β can be used to set the impact of a punishment.

Additionally, the new peer trust value can be used to differentiate between short-term and long-term trust trends.

$$ST_{n+1}(i,j) = \begin{cases} (1-u)ST_n(i,j) + uPT_{n+1}(i,j), & PT_{n+1}(i,j) - ST_n(i,j) \geq -\varepsilon \\ (1-v)ST_n(i,j) + vPT_{n+1}(i,j), & else \end{cases} \tag{3.45}$$

In order to determine the *short trust value* $ST_{n+1}(i,j)$ (Eq. 3.45) at time $n+1$, the new peer trust and the last short trust are taken into account. The factors u and v are again used to differentiate between trust increase and decrease.

$$LT_{n+1}(i,j) = \frac{LT_n(i,j) * n + PT_{n+1}(i,j)}{n+1} \tag{3.46}$$

The *long trust value* $L_{n+1}(i,j)$ (Eq. 3.46) computes the mean trust value of all the time intervals regarded.

The *final trust value* shown in Equation 3.47 is the minimum of the short trust and the long trust.

$$T_n(i,j) = min(ST_n(i,j), LT_n(i,j)) \tag{3.47}$$

The analysis of this approach shows that the decay factor (time fading function) is introduced very late (and applied to the trust deviation). Therefore, the trust deviation is crucial to detect oscillation. In the TDG, a certain degree of oscillation is planned by the system designer as it is part of the adaptivity of the agents.

Moreover, since negative behaviour is never forgotten, it represents a problem in the TDG because we want to incentivise malicious agents to change their behaviour and, therefore, need to implement forgiveness [74].

3.8.16 ARRep: An Adaptive and Robust Reputation Mechanism for P2P Networks

The last trust model in this state-of-the-art analysis is *ARRep* [59].

It computes the direct trust T_{ij}^D (Eq. 3.48) between truster i and trustee j using the ratings ex_{ijk}, which are available after transaction n_{ij} about peer j.

$$T_{ij}^D = \frac{\sum_{k=1}^{n_{ij}} (\lambda^{n_{ij}-k} * ex_{ij}^k)}{\sum_{k=1}^{n_{ij}} \lambda^{n_{ij}-k}} \tag{3.48}$$

λ is the decay factor in (0,1].

$$T_{ij}^R = \frac{\sum_{k \neq i} (C_{ik} * \hat{T}_{kj}^D)}{\sum_{k \neq i} C_{ik}} \tag{3.49}$$

The recommendation trust T_{ij}^R (Eq. 3.49) is computed from the *recommendation credibility* C_{ik} and the *transaction-based direct trust* $\hat{T}_{kj}^D$ (Eq. 3.50), which is the direct trust weighted by the number of transactions with trustee j.

$$\hat{T}_{kj}^D = T_{kj}^D * \eta^{1/n_{ij}} \tag{3.50}$$

η is a weighting factor in (0,1] to control the influence of the direct trust. Therefore, reports from peers, with which many interactions have been made in the past, are taken into account to an increasing extent.

The recommendation credibility C_{ik} is computed based on similarity as shown in Equation 3.51.

$$\widehat{Sim}_{ik} = 1 - \sqrt{\sum_{l \in CS} (\hat{T}_{il}^D - \hat{T}_{kl}^D)^2 / w} \tag{3.51}$$

The similarity $\widehat{Sim}_{ik}$ computes from the deviation of ratings of truster i with the ratings of peer k about peer l. The *common set* CS includes the peers that have rated trustee j and have been rated by truster i. w is the number of peers in the common set.

$$C_{ik} = \widehat{Sim}_{ik} * (1 - \sqrt{u - w/w}) \tag{3.52}$$

Based on w and the number of peers u, which have interacted with peer i, the feedback trust is computed in (3.52).

Direct trust and recommendation trust are combined in (3.53) to the *overall trust value* T_{ij}, which corresponds to our service trust.

$$T_{ij} = \alpha * T_{ij}^{D} + (1 - \alpha) * T_{ij}^{R} \tag{3.53}$$

The weighting factor α depends on the number of transactions n_{ij}.

$$\alpha = \begin{cases} \frac{n_{ij}}{M}, & n_{ij} < M \\ 1, & else \end{cases} \tag{3.54}$$

If n_{ij} is larger than threshold M, only direct trust is used to compute service trust.

This approach fulfils most of the criteria that we analysed to be worthwhile for a trust management system in the TDG; only the different weighting of trust increase and decrease is missing. It is questionable whether relying solely on direct trust with a certain number of own interactions is worthwhile without taking into account the age of the interactions. If an agent's own interactions are old, he might be better off relying on recommendation trust as well.

3.9 Analysis and Discussion of Trust and Reputation Models

In this section, the different approaches from literature are rated based on the criteria we have determined as important mechanisms for the TDG.

Table 3.2 shows which trust models fulfil the criteria and are thus compared more thoroughly. In the following, we will argue which trust model is best suited for the TDG and why we needed to implement our own approach rather than use a model from the reviewed literature.

After having applied the criteria to the trust models from literature, we now take a deeper look into the approaches that fulfil most of the criteria and their dynamics in a system like the TDG.

Trust Mechanism	Criterion					No. of fulfiled Criteria
	ST and FT	DT and RT	No Re-Entry	Decay Factor	Weighting	
Beth [63]	√	–	√	–	–	2
Jøsang [64]	–	√	√	–	–	2
Eigentrust [60]	–	–	√	–	–	1
Selcuk [65]	–	√	√	–	–	2
PeerTrust [66]	√	–	√	–	–	2
Chang [62]	–	√	√	√	√	4
Jøsang 2[67]	–	√	√	√	–	3
Chang 2[68]	√	√	√	√	–	4
ICRep [75]	√	√	√	√	–	4
Affair-based [69]	–	–	√	–	–	1
Star Worth [70]	–	√	√	–	–	2
TwoHop [71]	–	√	√	–	–	2
SFTrust [72]	√	√	–	√	√	4
RATM [73]	√	√	√	√	√	5
ARRep [59]	√	√	√	√	–	4

Table 3.2: Criteria fulfilment

The trust model from [62] is not suited for the TDG because service trust and feedback trust are not separated, which makes the system vulnerable to unfair rating attacks.

As [75] includes the approaches of [68], the younger model is obsolete.

Therefore, four trust mechanisms, *ICRep* [75], *SFTrust* [72], *RATM* [73] and *ARRep* [59] remain as interesting candidates for the TDG as they fulfil most of the criteria and are applicable in our simulation environment.

At first glance, *RATM* is the only trust model fulfiling all of the five criteria. In this approach, feedback trust is calculated, as in *ICRep*, using the difference between the reputation of trustee j and an agent's own trust towards trustee j. Therefore, in this approach, service trust and feedback trust are separated, but are not truly independent from each other.

SFTrust and *ARRep* evaluate feedback trust using heuristics to estimate the similarity of two agents. This makes discrimination and collusion attacks less likely because cooperative agents do not rely on the feedback trust of malicious agents [59].

ICRep and *ARRep* both miss the criterion of different weightings of trust increase and decrease. Adding this aspect to the trust models would be possible, but the modification might affect the dynamics of these trust models. Such effects cannot be foreseen completely by analysing the models beforehand.

SFTrust does not meet the criterion of prevention of trust increase for re-entering the system. The authors argue that authentication is a task of the system itself and need not be performed by the trust mechanism [72]. This is a valid assumption as there already exists related work on authorisation and authentication for grid systems, which could be used by the TDG. Additionally, *SFTrust* does not take into account the number of interactions on which a trust value is based [72].

Based on the analysis of our taxonomy in 3.2, we can see that there is no trust model in literature which matches all the criteria necessary for the TDG.

Therefore, we decided to develop a trust and reputation management system that is customised for the requirements of the TDG. Apart from fulfiling all necessary requirements for our application scenario, this model also incorporates dynamic components as presented in the Section 5.

3.10 Summary

In this chapter, we have presented the state of the art of self-organised or trust-based grid systems, scheduling and matchmaking techniques for grids and trust and

reputation models. We have seen that many similarities exist, but each system aims at slightly different purposes and, thus, fulfils different criteria. Therefore, we decided to develop a system, which fulfils all our requirements regarding open desktop grids and trust and reputation mechanisms. We use a self-organised agent-based approach which embeds the trust and reputation mechanism as part of the agent interaction as presented in Section 5.2. Nonetheless, we adapt several techniques from the state of the art as will be presented in the following chapter.

Chapter 4

Supportive Technologies

This chapter presents the technologies from the state of the art, which have been extended or adapted in order to be used in this thesis.

This thesis is classified into the area of Organic Computing techniques. Therefore, we will present the state of the art of the most important aspects of OC for this thesis, the Observer/Controller pattern, and learning techniques.

As we use techniques from MAS and normative systems, an overview of these will be given as well.

4.1 OC Techniques

Self-organisation is a bottom-up mechanism based on the emergent global effects of local interactions. Therefore, local decisions are an important aspect of such systems. This makes way not only for local optimisations leading to global improvements, but

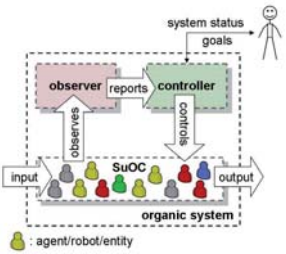


Figure 4.1: Observer/Controller pattern according to [78]

also for new ways of modelling such self-organising systems. Moreover, we must ensure that these systems are secure even though they reorganise at run-time, and subsystems can join and leave at any time. One example of these security techniques are assertions [76] that constrain the system into an area predefined by the designer. Within this area, the system is autonomous. One example of an architecture enabling this controlled autonomy is the Observer/Controller pattern [77].

Figure 4.1 shows the general architecture of the Observer/Controller pattern. The underlying system (system under observation and control (SuOC)) is constantly observed and controlled by a higher-level component, which consists of an Observer part and a Controller part. The Observer supervises the system and analyses certain parameters. These parameters, as defined by the observation model, have to be initialised at system startup by the Controller (derived from user goals) or the system designer. All parameters of the observation model are observed and analysed. If a critical situation occurs, the Controller is informed. The Controller plans the future actions, changes the parameterisation of the SuOC, and forces the system back into the desired target space [79].

In this thesis, we extend the Observer/Controller pattern in several ways.

1. **Adaptive Observation Model**

   We make the observation model adaptive in such a way that the Controller can choose at run-time the parameters that need to be observed in the current situation. This enables the agent to make better behaviour decisions because it can itself define which information is currently necessary. Moreover, we reduce the communication overhead by adapting the set of observed parameters to the needs of the Controller and to the situation; i. e. we observe only what is

necessary. Since Organic Computing systems are especially suited to handle changing situations, we extend this suitability by adapting not only system actions, but also system observations to the changed situation.

2. **Situation Prediction**

   Moreover, we extend the Observer component with the ability to predict situational changes before these even occur. This enables the Controller to adapt proactively to situations changing in the near future rather than react after the situation has already changed.

3. **Controller Learning**

   Furthermore, we enhance the Controller's decision-making process. The Controller is able to learn at run-time which action leads to the best result in which situation.

4. **Norm-aware Controller**

   We extend the Controller's behaviour selection by the ability to include institutional norms into the decision-making process. A higher-level institution observes and analyses the situation, and legislates norms regarding how the agents need to behave in order to resolve a global system conflict. The agent's Controller is enabled to interpret these norms and to tailor its decision to a norm-conforming one.

4.1.1 Analysis of Suitable Learning Techniques

This section provides an overview of the learning techniques that we have analysed regarding their suitability for usage in our trust-adaptive agents. In the beginning of this section, we will introduce the possible candidates for implementation in our system. Later, we will analyse these candidates using a taxonomy that we have developed and will discuss the algorithms that we chose for implementation.

The techniques that we have chosen for analysis and comparison are *CACLA* [5], *Fuzzy XCS* [80], *SMC Learning*, [81], *Continuous Q-Learning*, [82], and *Hedger* [83]. These algorithms all extend classical reinforcement learning techniques like learning classifier systems (LCS; cf. simplified *ZCS*-algorithm [84]) and *Q-Learning* [85].

Learning Algorithm Prerequisites

In order to be suitable for our agents, learning approaches must be able to make agents improve their decision process without any user interaction. Therefore, three

prerequisites have to be met.

First of all, the agent needs to be aware of its environment. Therefore, we have introduced an Observer [11] that collects information regarding the agent itself and its environment.

The second prerequisite for the application of learning mechanisms is that the agent needs to be able to influence its environment. This is realised by a Controller [11] that maps situations into actions.

The last prerequisite is a reward function. The agent needs to be able to rate and judge the results of each reaction in a given situation. The reward is used to optimise the situation/action mapping of each agent.

These prerequisites are crucial for learning algorithms to be applied to our agents in general. However, there are also requirements a learning algorithm has to meet in order to be suited as a good learning mechanism for our agents. These requirements are explained in the following section.

Learning Algorithm Requirements

In this section, the requirements a learning algorithm needs to comply with in order to be suited for our agents are presented.

1. In order to prevent the agents from sampling errors, we want the learning mechanism to be able to map **continuous situations** into a continuous action space. Therefore, no errors will result from the fact that the sampling of a situation has not appropriately been chosen.

2. **Knowledge representation** is crucial for a learning mechanism: The more efficient a representation is chosen, the faster the runtime performance of the algorithm is. Moreover, it is one of the key factors to ensure scalability in large systems. One approach to improve the runtime performance is to decouple the storage of the situation/action representation (e.g. Q/V) and reward. The ability to **generalise knowledge** and thus transfer knowledge to similar situations is also helpful regarding efficient knowledge management.

3. In addition to that, the ability to **include a priori knowledge** into the learning process is interesting for our agents, because the mechanisms that have been developed by system designers can be used as an efficient starting point for the learning algorithm.

4. In our application scenario, feedback (e.g. in terms of reward) is usually given after a job has been completed. Therefore, the reward is not measurable after the decision, but rather after the job, for which the decision was made, is finished. Therefore, the learning algorithm has to be able to cope with **delayed rewards**.

5. Apart from delayed rewards, the learning algorithm also needs to be suited for **multi-step** learning. The reward is given after the next job is finished, but the reward cannot always be broken down to exactly one decision. Moreover, a set of decisions is responsible for a reward outcome. Therefore, the learning algorithm has to make sure that the reward is broken down to all decisions which led to the reward in a suited manner.

Fuzzy XCS

The general problem of LCS is the restriction to discrete situation and action spaces. Fuzzy XCS [80] eliminates this constraint by using fuzzy logic [86] to map continuous input values to linguistic terms with certain affiliation degrees that are then concatenated using if/then statements and logical operators within $(0, 1)$ intervals to define output variables, which are then aggregated. Broadly speaking, this matches our definition of reputation values being low, medium or high, but adds optimisation by weight adaptation.

SMC Learning

Sequential Monte Carlo (SMC) learning [81] belongs to the category of actor-critic learning techniques. These consist of two separate subsystems: The *actor* saves the behaviour rules (action policy) and contains situation/action mappings. The *critic* uses the reward to predict expected rewards for actor actions. This rating is used by the actor to optimise the stored situation/action mapping using a specialised version of the q function (the q value represents the expected reward). SMC learning requires discretized situations; therefore, it would need further adaptation in order to be used for our agents.

Hedger

Hedger [83] is a Q-learning based mechanism that uses regression methods to generalise the representation of its knowledge. This is especially useful in situations where rewards appear only rarely. The knowledge is represented and stored. As it

is impossible to store all situations and actions that might occur, Hedger stores only tuples that have already occurred in the system. The storage is accessed by seeking a q value (expected reward) for a given situation/action pair. If a situation/action pair is not stored yet, Hedger uses a regression function to approximate the expected reward. This regression uses all situation/action pairs within a given euclidean distance k_{thresh}. The success of this approximation depends on the quantity of supporting points used. In order to determine whether a useful approximation can be computed, Hedger uses a convex covering of the sampling points. Additionally, it ensures that the quantity of sampling points is larger than a predefined parameter k_{min}. In general, to define a hyperplane in an n-dimensional space, we need at least n sampling points. If insufficient points are given, a default q value is returned. After the execution of the best-suited action, its q values are updated according to the reward achieved by the action. Additionally, all sampling points that have been used to approximate an action's q values are updated in such a way that the weights are adapted.

This is possible even if the reward is generated only seldom—for instance, in our system, most rewards are calculated after the entire job has been processed, which is why the reward is delayed.

Additionally, Hedger is able to use designer-given information (such as our TT^{acc} table) as starting sampling points. Therefore, the learning algorithm needs not start from scratch. The knowledge possessed by the agents is already part of the algorithm's solution space.

CACLA

The continuous actor-critic learning automata (CACLA) [5] is similar to SMC, and stores the mapping between situations and actions as an Actor Critic (AC) function. The critic part of the algorithm stores the expected reward of situation/action pairs: the so-called V function.

The actor and critic parts in CACLA are both realised as neural networks [1] to store the AC and V function. Thus, both situation and action spaces can be continuous. The adaptation of the weights within the neural networks is carried out using the *gradient ascend* method.

After an action has been selected and executed, the reward is compared to the expected reward (see Equation 4.1). This *temporal difference error* is called δ_t, r_t is the reward, and $V_t(s_t)$ the current approximation. In order to account for the future

[1] An overview of neural networks can be found in [87] and [88].

development of the reward, the current approximation of the following situation, $V_t(s_{t+1})$, is also taken into account. The influence of the following situation is defined by the *discount factor* γ.

$$\delta_t = r_t - V_t(s_t) + \gamma V_t(s_{t+1}) \tag{4.1}$$

Thus, δ_t can be used to improve the approximation of the critic. Similarly, CACLA improves the actor's decision using a Gauss exploration. A parameter defines the probability of changing an action. Therefore, a Gaussian distribution with the starting action as the mean and a predefined variance is used. Within this distribution, a random action is chosen. If the chosen action is executed and rated with a positive error (reward), the actor is adapted accordingly.

Taxonomy of Learning Mechanisms

Learning Techniques	Criteria								No. of Criteria Met
	Cont. Situations	Cont. Actions	Multi-Step	Generalisation	Separation Q/V and π	Representation	Knowledge Integration	Delayed Reward	
ZCS	–	–	√	√	–	√	√	√	5
Q-Learning	–	–	√	–	–	–	√	–	2
Fuzzy XCS	√	√	(√)	√	–	√	√	√	6 (7)
SMC Learning	–	√	√	–	√	√	–	–	3
Cont. Q-Learning	√	√	√	√	–	√	√	–	6
Hedger	√	√	√	√	–	√	√	√	7
CACLA	√	√	√	√	√	–	–	–	5

Table 4.1: Taxonomy of learning techniques

In this section, we analyse the learning techniques using the taxonomy provided in Table 4.1.

Due to the restriction towards discrete situations and discrete actions, ZCS and Q-learning are not suited as learning mechanisms in our scenario. SMC learning is

suitable only for discrete situation spaces; therefore, it too is not applicable in our system. Due to the usage of fuzzy logic, Fuzzy XCS is able to deal with continuous situation/action spaces. Knowledge is represented in a compact way and *a priori* knowledge can be implemented as rules. Additionally, it can be used in a multi-step manner, which means that the reward can be broken down to the set of actions leading to this reward. In other words, it is possible, but that delayed rewards can be dealt with has not yet been shown.

Continuous Q-Learning presents many useful properties (see table 4.1), but is unable to deal with delayed rewards—a factor that is crucial in our system.

Hedger has been developed to map continuous situations to continuous actions. The adaptation of q values using temporal difference enables Hedger to deal with multi-step problems. Generalisation is enabled by using regression with a few sampling points. Actions and q values are not separable, but at least knowledge is well represented because the complex q function is approximated using only a few sampling data points. Therefore, Hedger is a suitable candidate for learning in our application scenario agents. Thus, we implemented Hedger but came to realise that, due to the complex regression algorithm necessary for decision making in our application scenario, Hedger's run-time behaviour was insufficient. Therefore, although Hedger is theoretically a suitable candidate for learning in our agents, we had to rule it out due to its poor run-time behaviour for large, complex systems.

CACLA has also been developed to work in continuous situations and to compute actions in a continuous spectrum. Also, based on temporal difference, it is able to solve multi-step learning problems. Due to the neural networks used in both, actor (AC function) and critic (q function), knowledge is represented in a compact and generalised manner. But this has the disadvantage of CACLA's being unable to include *a priori* knowledge (like our TT^{acc} table) into the learning mechanism. Additionally, CACLA in its basic implementation is unable to deal with delayed rewards. But, since it is the most lightweight learning algorithm regarded here, we decided to implement the missing functionality of CACLA.

In Chapter 6.8.3, the adaptations necessary to use CACLA in our system will be explained in detail.

4.2 MAS

In concordance with Wooldrige, we define multi-agent systems (MAS) as follows:

Definition 6. Multi-agent Systems:
Multi-agent systems are 'systems composed of multiple interacting computing systems, known as agents'[89].

Accordingly, agents in this thesis are defined as being capable of showing autonomous behaviour and interacting with each other in a social fashion within an environment. Agents have specific goals that are usually defined by the user they represent in the system.

Definition 7. Agents:
'Agents are computer systems with two important capabilities. First, they are at least to some extent capable of autonomous action – of deciding for themselves what they need to do in order to satisfy their design objectives. Second, they are capable of interacting with other agents'[89].

Agents continuously perceive their environment, use reasoning to make their decisions, and then act within the environment. The fitness landscape [79] within the environment can be influenced by the different agent's actions.

This way of thinking and system design allows for a new way of dealing with the complexity of open distributed systems. Therefore, in this thesis, we use the multi-agent system approach as the basis of the system in which our trust-strategic agents act and interact.

In the following sub-sections, we will provide an overview of types of agent architectures and agent design principles before introducing the class of normative systems and the principles that we adopted from this area for the mechanisms in this thesis.

4.2.1 Types of Agent Architecture

There exist many different paradigms regarding how agents should be built in order to fulfil specific needs. Many approaches focus on the communication of agents, which is the basis of all cooperation and interaction. For instance, FIPA [90] and the FIPA-conform JADE [91] standardised agent communication as a means to enable agents from different developers to 'understand' each other.

In our MAS implementation, we abstracted the agent communication by encapsulating the communication component. Therefore, we can either use the physical communication layers of an underlying middleware [92] or the communication system based on encapsulated method calls within Repast [93] agents.

Apart from the communication base, the agent architecture, in general, is used to decompose complex decision-making mechanisms.

A prominent example of agent architectures is *BDI* [94], where, similar to a human being's reasoning mechanism, the agents' beliefs, desires and intentions are separated. The belief stores the agent's observation of its environment, including other agents' behaviours. The agent's motivation and goals are formalised in the desire component. As a result of these desires and the agent's perception of the environment, the intentions are formalised: The agent selects a specific desire that must be achieved in the current situation.

The BDI pattern is realised in several types of agent architecture. The *Open Agent Architecture (OAA)* [95] uses a broker-based approach to allocate software services on BDI-based agents. Apart from the restriction towards software services, the support of the community seems to have been lost, and the most recent update has been released in 2007.

The *Adaptive Agent Architecture* by Kumar [96] also uses a broker-based BDI approach, but concentrates on teamwork and, thus, on the self-healing of the system. The self-healing and fault-tolerance properties of this approach are interesting and relevant, but do not go as far as the self-organisation, learning, optimisation and proactive approaches reached by the Adaptive Agent Architecture developed and presented in this thesis. Specifically, the combination of trust-awareness, the Adaptive Observation Model, situation prediction and Controller enhancement regarding learning optimal behaviour and norm-awareness reach to a combination of self-organisation, self-healing and self-optimisation far beyond Kumar's approach.

The DARPA-funded Cognitive Agent Architecture Open Source Project (COUGAAR) [97] provides a framework for developing distributed systems with a large number of member agents. This approach concentrates on a common goal of the agents. This is in contrast to our application scenario, where the individual goals might interfere with the individual goals of the other agents within the system.

Similar to the three-level architecture presented in [98], we order the different agent decisions in levels to enable different behavioural classes in one combined architecture. For instance, the temporal component when the agent's decision is made can be decomposed into two levels.

4.2.2 Reactivity and Proactivity

Reactive agents are able to perceive their environment and react to changes by changing their own behaviour in order to achieve a better performance. Deliberative agents are able to select between different types of behaviour, which means that they can act differently in different situations. The agents presented in this thesis are called proactive.

Definition 8. Proactivity:

Proactive entities are able to react or act deliberatively, predict future system developments or agent actions, and adapt their behaviour even if the situation has not occurred yet.

This proactivity enables agents to reduce the performance decrease that usually occurs before the reactive adaptation has taken effect.

4.2.3 Awareness

Awareness can be defined as the ability of agents to perceive their environment. There exist different gradings of awareness. Perception is a prerequisite for agent actions because agents, in general, need to know what happens around them as well as how their actions influence other agents and the environment.

Definition 9. Awareness:

Awareness is the ability of an entity to perceive its environment.

Usually, the agent awareness is static and provided in the observation model, where the set of perceived parameters is defined once at startup by the system designer. In this thesis, we show how this observation model can be adapted at run-time by the agent itself. Agents only need to perceive the information that is important in its current situation. This reduces the observation overhead, which always causes a communication overhead in our application scenario because other agents need to be asked for information. This is especially interesting in systems with limited bandwidth or a high amount of payload compared to the available bandwidth.

4.2.4 Institutions

As discussed in Section 1, our application scenario TDG can be viewed as a common-pool resource management problem. Ostrom [7] argued that open access to a

common-pool resource need not inevitably lead to a tragedy of the commons (the total depletion or exhaustion of the resource) as predicted by game theory. Under certain conditions, there is an alternative to privatisation or centralised control of the resource—namely, management of the commons by a self-governing institutions.

Definition 10. Institution:

An institution is 'a set of working rules used to determine who is eligible to make decisions in what area, what actions are allowed or constrained, what aggregation rules are used, etc.'. [99]

In this thesis, the institution is represented by a manager (either a norm manager or part of a Trusted Community manager) that observes and analyses the system states and legislates norms.

Ostrom identified eight principles for the self-management of CPRs, which were necessary and sufficient conditions for institutions to endure (and, by implication, for the common pool resource to be maintained):

1. Clearly defined *boundaries*: Those who have rights or entitlement to appropriate resources from the CPR are clearly defined, as are its boundaries;

2. *Congruence*: between appropriation and provision rules and the state of the prevailing local environment;

3. *Collective choice* arrangements: In particular, those affected by the operational rules participate in the selection and modification of those rules;

4. *Monitoring*: of both, state conditions and appropriator behaviour, is by appointed agenciesthat are either accountable to the resource appropriators or are appropriators themselves;

5. A flexible scale of graduated *sanctions*: for resource appropriators who violate communal rules;

6. Access to fast, cheap *conflict resolution* mechanisms;

7. Existence of and *control over their own institutions*: is not challenged by external authorities; and

8. *Systems of systems*: layered or encapsulated CPRs, with local CPRs at the base level.

Ostrom proposed the idea of self-governing, common-pool resources through collective action in the context of an institution.

In the following sections, we analyse the TDG from the perspective of institutions for common-pool resource management.

Analysis of Institutional Concepts within the TDG

In this section, we visit each of Ostrom's principles, as introduced in the previous section, and analyse the extent to which they are already encapsulated within our iTC and eTC concepts.

Principle 1: Clearly Defined Boundaries

The first principle to ensure enduring institutions is the definition of clearly defined boundaries. The iTC is adaptive in such a way that the boundaries are implicitly defined by the set of agents belonging to the community. This set changes according to the community's need in the current situation. Therefore, an iTC does not have clearly defined boundaries known to all member agents. Thus, we see that the current system does not entirely conform with the first principle for enduring institutions. An eTC, on the other hand, has clearly defined boundaries as it has an explicit membership function and the TCM observes whether an agent's behaviour conforms to the community's norms.

Principle 2: Congruence The second principle of enduring institutions is congruence between appropriation and provision rules and the state of the prevailing local environment.

In an iTC, it is arguably precisely the lack of correlation between the interaction norms and the system states in which there is no mutual cooperation because of trust breakdown. Hence, an iTC only partially implements the second principle for enduring institutions.

An eTC, on the other hand, can be made fully compliant with Principle 2, but this requires an additional communication protocol so that global states can be identified from reported local states. Furthermore, there has to be some assurance that the configuration of 'community rules' is precisely correlated with the intended outcomes.

Principle 3: Collective choice arrangements According to Principle 3, the agents affected by the operational rules participate in the selection and modification of those rules. Due to the adaptivity in our system, the agents are able to chose the extent to which they obey the existing norms, but they do not control the norms themselves. For instance, it is a norm that agents should work for other agents—i.e.

offer their idle computing resources. If the majority of the agents chooses to be less cooperative, this results in a new situation of less trust among the agents. We call this an effect of a *self-referential fitness landscape*: by making a choice, each of the agents actively changes the situation in which it acts. Thus, the agents have the local choice of whether or not to obey a rule, but they do not have collective legislative power.

In an eTC, the agent occupying the role of TCM has the institutionalised power to pass, enforce and adapt norms. As these norms could also be decided by collective choice (e.g. by voting or consultation), it would be possible to implement the third principle of enduring institutions in eTCs.

Principle 4: Monitoring In our distributed self-organised system, monitoring is carried out by the agents themselves. Each agent has

1. knowledge about the agent itself,
2. private knowledge about other agents gathered during former interactions,
3. community knowledge about other agents (especially from the reputation database).

Thus, there exists no entity or entities with a responsibility for monitoring *global* states or behaviour. This is a problem if situations occur that cannot be observed locally—for instance, a breakdown of trust or other more complex macro-level properties that emerge from micro-level behaviour and interactions. For these situations, the introduction of the TCM, which observes or infers not only memberships but also system states among the members, is necessary. With an eTC run by a TCM, Principle 4 can be fulfilled.

Principle 5: Graduated Sanctions If the community rules are violated, the violator needs to be punished according to the severity of the damage caused. This principle is mandatory for the isolation of malicious agents from the iTC, which is built based on a trust management and sanctioning mechanism. Our system already has a trust and reputation mechanism, which also takes into account the amount of violations that the agent has committed. Therefore, Principle 5 of enduring institutions is already established in the iTCs of the TDG, and must be retained for eTCs as well.

Principle 6: Conflict Resolution According to Principle 6, agents require access to fast and cheap conflict resolution mechanisms in order to establish enduring institutions. In case an agent objects to a sanction (e.g. bad rating for a wrong or delayed computational result, or unintentional error), this conflict needs to be resolved. In the existing TDG, the ability to object to a rating is not implemented.

Instead, we rely on demanding a minimal trust value from agents to be allowed to provide feedback in the reputation system. But as soon as an objection to negative feedback is allowed, some form of conflict resolution is needed—in particular, to avoid escalating negative feedback or the stoning effect observed in some reputation systems [100]. Managing and adjudicating the process would be another task for the TCM of an eTC.

Principle 7: No External Control As there are no external instances in both implicit and explicit TCs, Principle 7 is met by the current version of our system. For a future version, we envision multiple TCs with a possible overlap of concerns. In such a system, it might be necessary to allow a mutual influence of TCMs of different TCs.

Generally, there will always exist the possibility of human intervention to override the decisions of the TCM. Hence, Principle 7 is useful only to a certain degree within a socially organised system.

Principle 8: Systems of Systems If we regard TCs (implicit or explicit) as systems, systems of systems are TCs composed of sub-TCs. In the current implementation, there exists only one bottom-up built TC, but we are also interested in the relations between different TCs. In particular, overlapping TCs might have different, even contradicting goals, and hence utility functions. Furthermore, as TCs are currently defined only by the trust values and thresholds of the members, we could also imagine nested TCs where larger TCs could define a minimal trust threshold (to skip basic security measures) and nested sub-TCs accept only trustworthy agents (for instance, to guarantee a certain QoS standard). The TC approach is well-suited for systems of systems aspects, but so far, Principle 8 has not been met because the application showed no necessity to implement it.

Analysis of Trusted Communities and Institutions

The aim of this section was to analyse, which of the design principles are already implemented in the trust-adaptive algorithms of our TDG agents and which principles need to be augmented by a TCM in order to transform our TCs into enduring institutions with hierarchical management properties. Principles 5 and 7 are already represented in the TDG and the iTC approach to agent management. One general task of the TCM is to observe the members of its explicit Trusted Community (Principle 1). It is also responsible for the recruitment of new members for the TC. If an agent misbehaves, it needs to store this interaction result in a reputation database. Additionally, it needs to resolve rating conflicts (Principle 6). If this occurs repeat-

edly, it needs to exclude this malicious agent from the TC and inform the members. This refers to Principle 2, which ensures that only members are given resources. One of the monitoring functions of the TCM is to react if a global situation occurs that cannot be resolved locally—e.g. trust breakdown within the community. In these situations, the TCM is required to install norms for the agents' behaviour (Principle 3), and monitor whether these norms are obeyed by the agents (Principle 4). As soon as we regard more than one TC/institution, the TCM will also be responsible for the inter-TC relations, even with respect to systems of systems, and will thus fulfil Principle 8.

In addition, a way of explicitly realising Principle 6 would be through a forgiveness mechanism [74] based on explicit evaluation of the violation incident rather than 'forgetting'.

In the current thesis, we use a forgetting mechanism that, when combined with the dynamics of trust and workload, leads to a reintegration of formerly untrustworthy agents if they have changed their behaviour.

4.2.5 Normative MAS

In this thesis, useful aspects or normative MAS are adapted in order to improve system performance. In particular, we introduce a hierarchical component (representing an institution) that is able to observe those system states which might not be observable through only a local view (e.g. trust breakdown). From the analysis of these system states, the institution derives norms. Norms define the expected behaviour of an agent in a society of agents.

Two basic definitions can be used to define normative MAS, *the normchange definition* [101], which describes a top-down approach, and *the mechanism design* [102] describing a bottom-up approach.

Definition 11. The Normchange Definition:

'A normative multi-agent system is a multi-agent system together with normative systems in which agents on the one hand can decide whether to follow the explicitly represented norms, and on the other the normative systems specify how and in which extent the agents can modify the norms.'[101]

This definition implies that the MAS and the normative system are encapsulated and separated from each other.

Norms need to be explicitly represented and modified, and agents need the ability

to decide whether or not to obey to a norm.

Definition 12. The Mechanism Design Definition:

'A normative multi-agent system is a multi-agent system organized by means of mechanisms to represent, communicate, distribute, detect, create, modify, and enforce norms, and mechanisms to deliberate about norms and detect norm violation and fulfilment.'[102]

In this bottom-up definition, the agents themselves are responsible for legislating and controlling the norms. In this thesis, the top-down definition of *the normchange definition* is used to enhance the MAS with norms. Therefore, we need to enable agents to 'listen' to such standardised norms and to provide the system with an institutional component that is able to analyse the system state and to legislate norms to improve the system's performance.

Formalising Norms as Constraints

Norms can be represented in many different ways. Making the norm definition adhere to a well-known standard makes the mechanics more intuitive and understandable; moreover, standards make the extension of the existing sources easier. Therefore, we here analyse which norm representations are suited for norm-aware TDG agents (Section 6.9.2) and provide examples of how 'typical norms' can be represented in different language structures.

Constraints are used by the norm manager, which observes the system state and enacts norms for the agent in order to improve the global system state with the set of local interaction changes. In general, constraints can be defined as a set of conditions that have to be met in order for a norm to be enacted. The norm then represents the action instructions given from a norm manager to the agents in the system.

Requirements for Norm Representation Languages

First of all, we need a standardised representation of norms in order to define norms in such way that they can be understood and interpretable by our agents. Therefore, we need to define a condition and an action part as well as provide a way to observe whether conditions are met. The language should also be easy to understand for system and agent designers in order to make extensions as simple as possible. Also, we need a language which has a good integration into our tools. Therefore, the possibility to use the language in the software planning phase, e.g. integrated in

Unified Modeling Language UML would be interesting. Similarly, the usage of the language as additional Java annotations would be an interesting way to define norms understandable for both agents and humans.

An analysis of the state of the art led to two promising candidates for norm representation languages.

One of the most prominent languages for constraints is the Object Constraint Language (OCL) [103]. It is an extension of the Unified Modeling Language (UML) and, thus, is quite common. One of the benefits is that during the planning phase of a software artefact, OCL constraints can be included in UML diagrams, extending the graphical representation of UML through a more formal way of defining conditions. At run-time, OCL can be used to monitor the constraint requirements.

As an alternative, the Java Modeling Language (JML) is introduced. JML is also a language for constraints, but uses the Java annotation to supervise directly at run-time whether or not constraints are met.

In the following sections, we will take a deeper look into the functionality of OCL and JML and decide, based on our requirements, which language is most suited for our norm-aware agents.

OCL

The Object Management Group, Inc. (OMG) has been founded in 1989 and is an open membership, not-for-profit computer industry standards consortium, which, for instance, released the UML and OCL standards. OCL extends UML as a formal language to express constraints in a system.

We here give an overview of the OCL definitions and expressions used in this thesis.

- **Invariants**

  An invariant must hold at all times for all instances of a type.

- **Initial Condition** The initial condition defines the start value of a variable.

- **Pre- and Post-condition for Operations and Methods**

  A precondition defines the state that a variable has to be in before an operation or a method can be executed.

  A post-condition defines the state of a variable after an operation or method has been executed.

- **Body Definition**

  The body definition is a function. Its result must equal the result of an operation or method. Preconditions, post-conditions and body conditions can be combined.

.

Listing 4.1: OCL syntax

```
1  context NormManager::createOverloadWLNorm()
2  pre: self.overloadWLActive==false and self.avgWL>400
3
4  context NormManager::deactivateOverloadWLNorm()
5  pre: self.overloadWLActive==true and self.avgWL<=400
6
7  context NormManager::createTrustbreakdownNorm()
8  pre: self.trustberakedownNormActive==false
9  and self.avgRep>0.2
10
11 context NormManager::deactivateTrustbreakdownNorm()
12 pre: self.trustberakedownNormActive==true
13 and self.avgRep<=0.2
```

The OCL syntax allows the expression of highly complex constraint combinations. Therefore, we defined functions that facilitate the usage of OCL within the Adaptive Agent Architecture. A parser within the normative framework of this thesis ensures that the definitions can be transferred into agent behaviour decisions at run-time. For instance, Listing 4.1 shows a combination of preconditions that has been used to define agent behaviour; specifically, the norm manager defines when a norm is activated or deactivated.

Context always begins the expression. The context is followed by the expression itself, here defined by the class and method it is used in. A nesting of norms is possible, but currently not needed in the TDG. Nonetheless, extensions can easily be made due to the normative framework structure. The precondition *pre:* defines which variable combination needs to hold in order to make the norm active. Here, the *self* again points at the variables being in the same class as the method. For instance, the first part of the example says that the boolean *overloadWLActive* has to be true and the average workload $avgWL > 400$ in order to call the function.

JML

The Java Modeling Language (JML)[104] is based on *design by contract* (DBC). DBC is a pattern, which defines that a class has a contract with its client. The client needs to fulfil prerequisites in order to execute a method. JML presents an easy way to include conditions by adding Java annotations into the source code. Like OCL, JML can be used directly with the Eclipse IDE, and the conditions defined at design time can be supervised by the JML library at run-time.

The aforementioned four functions from the TDG example can be expressed in JML syntax as follows:

Listing 4.2: JML syntax

```
1
2 //@ requires overloadWLActive=false and avgWL>400
3
4 //@ requires overloadWLActive=true and avgWL<=400
5
6 //@ requires trustberakedownNormActive==false and avgRep>0.2
7
8 //@ requires trustberakedownNormActive==true and avgRep<=0.2
```

Since the constraints have to be defined directly before the method definition, the scope need not be defined here. Despite the different names, JML and OCL conditions can be mapped:

- *Invariant* can be mapped with the OCL invariant.
- *Requires* can be mapped as OCL preconditions.
- *Ensures* can be mapped as OCL post-conditions.

OCL Initial and OCL Body Definition are not known in JML. Instead, there is a set of JML commands, which refers to Java functions and lacks any exact correspondence with OCL. Some examples are listed here.

- **Signals** There is a post-condition if an exception is thrown.
- **Assignable**

  Assignable defines which areas can be changed by the method.

- **Spec Public**

  Spec public enables the designer to make a private order-protected variable public for a specific cause.

Analysis of Norm Representation Languages

The examples in the previous section show that there are similarities between OCL and JML, but both languages have special abilities suited for specific needs. For the needs of the norm-aware agents in this thesis, both OCL and JML can be used. We use a tailored version focusing on the core aspects of OCL, but making the definitions easier to handle due to our own language parser. The integration of OCL into UML helps the designer to include norms already during the planning phase of new agents. The standardised representation of norms in OCL makes norms easily interpreted by agents and also easy to add further norms to already exisiting agents and norm managers.

Several OCL implementations are available, but due to the wide distribution, free license and active support, we decided to use DresdenOCL as the basis of our norm definitions. DresdenOCL is based on Java and can be used directly in the Eclipse IDE. The integration in Eclipse IDE-based projects like our RePast [93] simulation TDG makes it easy to use for designers. It enables designers to define constraints formally and to execute a function if and only if the constraints are met. Therefore, the constraints are saved in an '.ocl' file, which is referred to by the source code.

4.3 Summary

In this chapter, the technologies, which have been adapted for usage in this thesis, have been presented and discussed. The extension of the Observer/Controller patter, usage of learning algorithms as well as agent architectures, norms and institutional components from the MAS area have been analysed regarding their suitability for the thesis.

In the following chapter, the system view of this thesis will be presented.

Chapter 5

System View

In this chapter, the functionality and dynamics of the trust and reputation system are presented. The second part introduces the concepts of implicit and explicit TCs. Work on the trust and reputation system, the application scenario (Section 1.2) as well as TCs has been conducted in cooperation with Lukas Klejnowski and Jan Kantert in the context of the DFG research unit OC-Trust (FOR 1085).

5.1 Trust Feedback Loops

In this section, our trust and reputation model as well as its dynamics are presented. This model is the foundation of all trust-based agent interaction and system mechanics in this thesis (see Figure 5.1).

As discussed in Section 3, we defined our own trust and reputation system in order to tailor it to the needs of our agent dynamics and application scenario prerequisites. Therefore, the model is described in detail here.

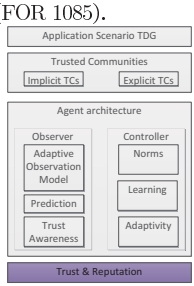


Figure 5.1: Trust and reputation

The current implementation of the reputation system uses
a centralised database, but it is also possible to exchange
the underlying reputation component and use a distributed storage—for instance, the middleware currently developed in the OC-Trust project.

To make interaction and cooperation decisions, agents use the aggregated trust level $T_{i,j}$, which is an aggregation of the agent's own experiences and the global reputation of another agent. The global reputation is used for feedback loops, which increases the agents' efficiency as well as the system's robustness. We will first introduce the basic mechanics of the trust feedback loops before providing detailed information about the calculation of $T_{i,j}$.

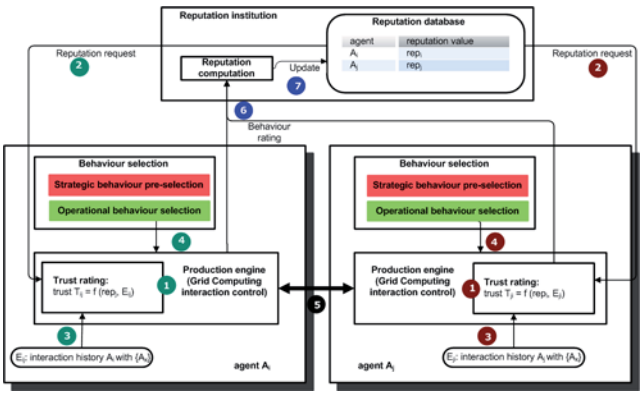


Figure 5.2: Trust-based interaction of agents A_i and A_j using reputation

Figure 5.2 shows the steps of a trust-based interaction. The agents have a production engine that performs the actual task (e.g. the grid client software of an agent in the TDG). In the TDG, the production engine distributes work units (submitter) and accepts work units from other agents (worker). To decide which agent to ask for work unit calculation and whose work units to accept, the production engine of agent A_i calculates the aggregate trust value $T_{i,j}$ of the cooperation candidate A_j (step 1).

$T_{i,j}$ is calculated using the reputation value rep_j of agent A_j (step 2) and the own experiences of A_j, $E_{i,j}$ (step 3). The type of interaction is additionally influenced by the behaviour selection component (step 4). In this component, strategic preselection

is performed—for instance, altruistic behaviour in order to build up reputation in the long-term.

The agent A_j undergoes an analogous loop, until in step 5, the actual interaction between A_i and A_j takes place. After the interaction, the behaviour of the interaction partners is rated (step 6). The aggregated ratings of several agents leads to an update of the reputation value (step 7).

The aggregated trust value $T_{i,j}$ of an agent A_i towards an agent A_j is computed as a weighted sum of the reputation rep_j of agent A_j and the own experience $E_{i,j}$ of agent A_i with agent A_j. The aggregated trust value is within an interval between -1 (not trustworthy) and +1 (trustworthy).

Each time that agent A_j is asked for cooperation (e.g. to compute a work unit), the asking agent A_i rates the interaction with a rating $r_{i,j}$. The value of this rating depends on the behaviour of the agent and the application scenario. For instance, in the TDG, it is more crucial for an agent if a wrong work unit result is returned (which has to be distributed and computed once more) than a rejection of a work unit, because in case of a reject, the work unit can be distributed in the next time step. With negative ratings $r_{i,j} \in \mathbb{R}^-$, positive ratings $r_{i,j} \in \mathbb{R}^+$, in the TDG, the ratings are realised as follows:

- If A_j has accepted and correctly returned a work unit, the work unit owner A_i informs the reputation database about a positive rating and adds a positive rating to its own experiences.

- If A_j has accepted a work unit but retrieved an incorrect result, the owner A_i will give a strong negative rating to both, the reputation database and its own experience storage.

- In case a work unit has been rejected by A_j, the agent will receive a negative rating in both, the reputation database and the work unit owner's own experience database.

- If A_j has accepted a work unit, but cancelled it during computation, the owner A_i stores and reports a negative rating, which depends on the computation progress of the cancelled work unit ($0 \leq progress \leq 1$). If the work unit has been cancelled in the beginning of the calculation, A_i loses a small calculation time interval and, thus, gives a slightly negative rating ($Rating_c^{min} \in \mathbb{R}^-$). If the work unit has been cancelled following a long calculation time, this is more crucial to the owner, and it will give a strong negative rating ($Rating_c^{max} \in$

$\mathbb{R}^-$). $Rating_c^{min} \geq Rating_c^{max}$. The rating ($r_{i,j}$) for a cancelling (c) of a work unit is a linear function defined as follows:

$$r_{i,j} = Rating_c^{min} + (Rating_c^{max} - Rating_c^{min}) * progress \tag{5.1}$$

In concordance with the literature [60] as well as the human understanding of trust, a negative experience is weighted as more severe than is a positive experience. Hence, we analysed different weighting models [105] and came to the conclusion that, for our model, a weighted sum depending on the ratings is best suited. The more of its own experiences an agent has, the less it needs to rely on the community knowledge of reputation values. Therefore, we introduce a parameter $RepWeight$, which accounts for the influence of negative ratings:

$$RepWeight_{i,j}(r_{i,j}) = \begin{cases} r_{i,j}, & r_{i,j} \geq 0 \\ -r_{i,j} + 1, & r_{i,j} < 0 \end{cases} \tag{5.2}$$

This weighting is used to compute a reputation value rep_j, which includes all ratings. In order to compute the aggregated trust value $T_{i,j}$, rep_j and $E_{i,j}$ are weighted with the reputation weight β ($0 \leq \beta \leq 1$). β depends on the number of own experiences $\#E_{i,j}$ as follows:

- If the number of own experiences is larger than a threshold $RT_{i,j}^{ratings,min}$ ($\#E_{i,j} \geq RT_{i,j}^{ratings,min}$), the reputation is only included with a minimal weight ($\beta = \beta^{min}$) because A_i can mostly rely on its own experiences with A_j.
- If $\#E_{i,j} < RT_{i,j}^{ratings,min}$, β is computed proportional to the number of own experiences:

$$\beta = (-\#E_{i,j} * \frac{1 - \beta^{min}}{RT_{i,j}^{ratings,min}}) + 1; \tag{5.3}$$

Using the weighting factor β, the aggregated trust value that A_i has in A_j is computed as:

$$T_{i,j} = (1 - \beta) * E_{i,j} + \beta * rep_j. \tag{5.4}$$

Figure 5.3 shows the control loops that result from our reputation model. Control loop I (blue) is located on the productive level. The production engine of agent A_i requests the reputation value of a potential cooperation partner. Based on the aggregated trust value calculated from the reputation value and the agent's own experiences with the potential partner, it decides whether or not to interact with the agent. After each interaction, the behaviour of the cooperation partner is rated.

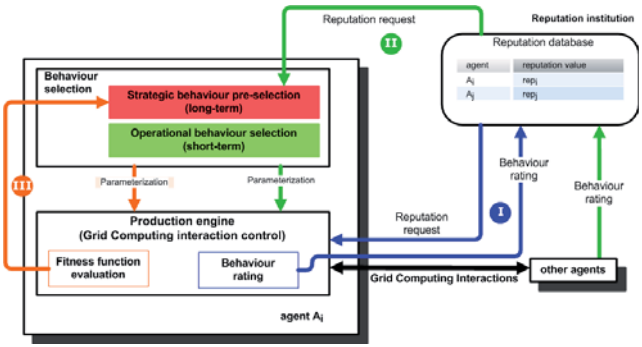


Figure 5.3: Trust-based nested control loops

Control loop II (green) is used for the self-adaptation of agent A_i. It requests its own reputation value rep_i and modifies its behaviour in order to change this value. The actual modification of rep_i is performed by the other agents, which rate the interactions they have with A_i. This control mechanism is based on the assumption that changes in reputation values result in performance improvement. Although there are many cases where this assumption holds, there are system situations where agents can change their reputation but will not reach better performance—for instance, in an overload situation.

Therefore, a third control loop for self-adaptation is loop III (orange). Agent A_i evaluates its own performance and changes its behaviour accordingly in order to optimise its performance. This control loop is realised by learning agents (see Section 6.8.3). Instead of using the reputation value as a steering parameter for adaptation, agents using control loop III directly concentrate on the performance metric and find behaviour parameters that optimise the performance. Such behaviour can lead to changes in reputation, but does not have to do so. Moreover, control loop III overcomes the problem of situations where reputation optimisation does not lead to the best performance output.

5.2 Trusted Communities

In this section, the concepts of implicit and explicit trusted communities are introduced (Fig. 5.4). We generally differentiate between implicit and explicit Trusted

Communities. Implicit Trusted Communities as result of local agent interaction (Chapter 6), whereas explicit TCs are an agent organisation based on unique membership functions.

5.2.1 Trusted Communities Definition

A Trusted Community (TC) is an association of agents that mutually trust each other due to previous experiences from direct interactions and knowledge about reputation. TCs are understood as a general approach that can be used in MAS with underlying trust mechanics. In our scenario TDG, agents act on behalf of their respective grid users and some of the users' properties are mirrored in the agent (e.g. an egoistic agent represents a user who might suddenly restrict the agent's access to computer resources).

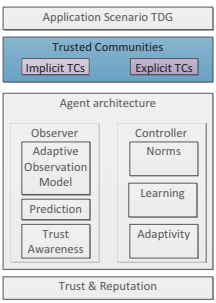


Figure 5.4: Trusted Communities

TCs are characterised by strong trust relations among the members, which enables them to share information and perform tasks more efficiently. If, for example, an agent A has not yet had any interaction with an agent B, he can rely on the estimation of his community about the reputation of this particular agent. This reduces the information uncertainty that generally exists in dynamic systems with respect to interactions between autonomous agents; therefore, it enables and improves cooperation among these agents. Besides, in a TC, agents can rely on the trustworthiness of the other members. Thus, they omit control mechanisms like work unit replication as long as their work units are processed by other members of the TC. Obviously, this can lead to better system performance.

Generally, in a TC, an agent always needs to find a trade-off between its own personal goals (like processing all work units as fast as possible) and the system goal (like maximum overall system performance that can be reached if agents not only submit work units but also process work units for others). TCs can increase the overall system performance as well as a single user's benefit from participating in the network.

In our system, TCs emerge over time. Each TC represents an agent organisation that is dynamic regarding the membership of its agents. If an agent changes its behaviour in such a way that it influences the TC in a negative way, it will be excluded from this organisation. If an agent that formerly acted uncooperatively changes its behaviour and becomes more altruistic, it will be able to re-join the TC. An implicit Trusted Community (iTC) emerges from the local views of the agents whereas an explicit Trusted Community (eTC) has predefined rules for the entry and departure of agents. These rules lead to a closed set of member agents. Therefore, using eTCs enables us to explore how agents from different disjointed TCs interact with each other. In the following, both shapes of TCs are described in detail.

5.2.2 Implicit Trusted Communities

iTCs have no fixed rules governing how an agent enters or leaves the society, and agents do not have explicit knowledge about these communities or their membership. Instead, the membership can be defined according to membership functions. Since, in our system, agents generally decide to send their work units to trustworthy agents and each agent has its own view on these rankings, the structure of a TC can be defined as all agents that are perceived as trustworthy by the majority of other agents. Therefore, the organisation of an iTC results from the exclusion of uncooperative agents by the means of avoiding cooperation with these agents.

Figure 5.5 shows an example of an iTC that we will describe in the following. Agent X has a set $S_{trusted,X}$ of agents it trusts, which is a subset of all the agents it knows.

$$S_{trusted,X} \subseteq S_{known,X} \tag{5.5}$$

This means that the aggregated trust level of each trusted agent A is greater than a threshold predefined by agent X. Agent X can adapt its constraints regarding the

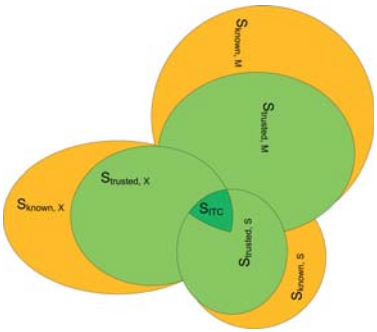


Figure 5.5: Implicit Trusted Community (iTC)

aggregated trust level to the current system situation over time.

$$\forall A \in S_{trusted,X}(aggTL_X(A) \geq aggTL_{min,X}) \tag{5.6}$$

Analogously, agents M and S define their trusted agents. The implicit Trusted Community is the set ITC, which is highlighted in Figure 5.5 and which forms the intersection of the sets of trusted agents of X, M and S:

$$S_{ITC} = S_{trusted,X} \cap S_{trusted,M} \cap S_{trusted,S} \tag{5.7}$$

If an agent needs to distribute more work units than its preferred (top-ranked) agents (e.g. the community members) are able to accept, it will either wait and distribute the remaining work units at a later time or ask less trustworthy agents. Waiting and distributing the remaining work units will decrease its personal benefit from participating in the system, whereas asking less trusted agents may result in decreased overall system performance. If the agent decides to ask less trustworthy agents, it will also increase the control mechanisms and, e.g. send the work units to more than one agent, allowing majority voting to decide the correct result. This replication technique helps it to receive correct results, but influences the overall system throughput in a negative way by increasing the overall workload.

As can be seen in Figure 6.8, we expect our system to isolate egoistic and free-riding agents in such a way that they are the last agents in a ranking based on the aggregated trust level. This means that these agents will only be asked if all better-ranked agents are unable or unwilling to process a work unit. Although the poorly

rated agents show sub-optimal cooperation, they can still be asked to process work packages. For instance, instead of waiting until a better-rated agent is available, a submitter can decide to replicate a work unit and give it to several agents and determine the correct result by majority voting. The strategy of asking average-rated agents rather than waiting for top-rated agents will, on the one hand, lead to better system performance because if the system load is high, all available workers are used. On the other hand, it will lead to a better personal benefit because the waiting time is minimised. We will later introduce the ability of agents to learn and change their behaviour over time. This means that agents that shift to more altruistic behaviour rise in the ranking over time, whereas non-adaptive agents remain in their excluded ranking position. Checking whether an agent has become more altruistic occurs automatically if the system load forces an agent to not only ask the top-rated agents but also other agents with the cost imposed by additional control mechanisms.

5.2.3 Explicit Trusted Communities

Klejnowski [106] presents a very detailed description and evaluation of eTCs. Explicit trusted communities (eTCs) have clear rules defining the requirements that an agent has to fulfil in order to join a TC. This enables the system designer to define inter-TC interactions. These will enable collaboration between different TCs within a system, thereby enhancing the possible actions and interaction partners of an agent (and, thus, indirectly influencing the system performance). For instance, if a submitting agent requests a task that cannot be fulfilled by any agent of its TC, it is able to ask agents of another TC that cooperate within their own community.

In an eTC, a trusted manager, which can be dynamically chosen, monitors the actions of other agents in the group and receives ratings for each interaction. It can decide whether a new agent will be allowed to join the TC or not. Eventually, it tests members of the community with sporadic, pre-processed tasks in order to estimate if their behaviour is correct (return right results) and conforms with the community (cooperate with members). The trusted manager is also responsible for the exclusion of agents whose behaviour no longer complies with the expectations of the TC.

The trusted manager can be implemented using the observer/controller pattern [11], which is common in the area of Organic Computing. A generalised picture of an eTC architecture can be seen in Figure 5.6. A dedicated TCM manages and observes the members of its TC. Cooperation with unassociated agents is possible, but TC members are preferred partners.

The observer supervises certain parameters of the TC and reports the results

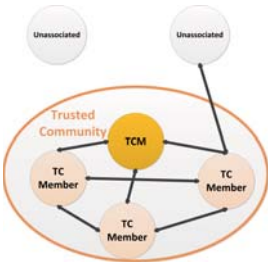


Figure 5.6: Explicit Trusted Community

to the controller. The controller realises if any critical condition is reached. The controller will then influence the community in a way that leads to a better overall system status. As there are several observer/controller entities for a system consisting of more than one TC, the system is still decentralised in its core, though a few hierarchical components exist.

Trusted Community Manager

The Trusted Community manager (TCM) can, for instance, be chosen using distributed leader election algorithms from the distributed systems domain. For instance, the agent with the highest ID can be chosen if each agent sends its own id and forwards the highest id it receives along the network.

An agent can also autonomously decide to found an eTC. This decision is usually made if the agent computes that it would reach a better performance within an eTC. If the agents decides to establish a new TC, it will invite other agents to join the TC. Usually, it will decide to invite only trustworthy agents in order to maintain a robust TC. Agents that are invited compute whether it would be worthwhile for them to accept the invitation.

The TCM is responsible for the maintenance of its TC. It needs to supervise its members and their actions. If necessary, it is responsible for expelling a misbehaving agent from the TC. This can be due to reports from other members or test packages that have been rejected or returned incorrectly from the misbehaving agent.

The TCM is able to assign roles and tasks to other agents in order to optimise the TC management as well as its own performance within the system and TC.

The TCM can also be responsible for legislating norms that resolve undesired

system states. If the TCM realises that the agents would be better off if the TC is dissolved, it needs to evoke this action, too.

eTC Life Cycle

Here, we describe the life cycle of an eTC.

In the first phase an eTC is always in the *unassociated phase*:

The agents of the system are completely without central control and act autonomously. The only common structure in this phase is the trust and reputation system where the agents rate the interactions that they had with each other. Additionally, constructs without centralised control, like the iTC, can emerge over time. Agents build up trust relations over time. After a certain amount of interactions and trust decisions, agents reach a point where they predict that their own performance can be improved if an eTC is formed. As soon as a critical number of agents has decided to initiate the formation process of the eTC, the next phase is reached.

In the *formation phase*, the agents try to find suitable members for the TC and elect a TCM. Thus, the eTC is formed.

In the *TC maintenance phase*, agents are able to optimise their performance by mostly relying on TC members when looking for cooperation partners. Nonetheless, they are also able to interact with agents outside the TC as well as with agents from other TCs. But it is a norm that agents which are asked for cooperation from members of their own TC, should be cooperative.

Moreover, within the TC, due to the bilateral trust relations, agents can leave out safety measures like work unit replication. This might enable them to reach a higher performance than without the TC membership.

The TCM supervises the agents and their interaction before deciding which agents to invite to join the TC and which agents should be excluded from the TC.

If a TC becomes too big, the maintenance cost do not scale anymore, and the agents are better off if they split the TC into two smaller, independent TCs.

Each agent supervises its own performance and estimates its performance potential if it were not a member of the TC. In case the membership leads to a disadvantage for the agent, it can decide to leave the TC. If the TC becomes too small over time, the TCM will dissolve the TC.

In case the TCM leaves the system (for example, loses internet connection), the TC will execute the election process as already performed in the formation phase.

5.3 Summary

In this chapter, we have introduced how trust-based cooperation takes place. We analysed the agent dynamics and determined three nested control loops that are responsible for the trust-adaptive behaviour of agents.

In the second part of this chapter, we presented two definitions of Trusted Communities, implicit and explicit, and discussed their suitability for open grid systems.

Chapter 6

Adaptive Agent Architecture

6.1 Motivation

This section presents the adaptive agent architecture and different implementation details of each included agent class (Fig. 6.1).

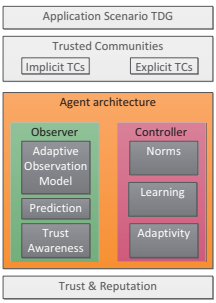


Figure 6.1: Agent component of the thesis

In order to enable agents to adapt continuously to both, changing environmental situations and varying trust relations, we need to extend agents as follows: Their awareness needs to be augmented with the ability to observe trust and reputation values. Moreover, our agent observer is able to predict how situations might evolve and how the trustworthiness of an agent could change. This enables agents to be not only reactive, but also proactive by adapting even before the situation change has occurred.

Agents are extended through their ability to learn. This is extremely helpful in open dynamic systems because the system designer cannot predict every situation that might occur and so cannot always preset optimal behaviour parameters. Therefore, enabling agents to continuously optimise their behaviour at run-time is promising.

Additionally, there are critical situations that the agents cannot observe using only their local knowledge. Therefore, we extend the agents' ability in such a way that they are able to percept norms from an institution which observes the system state. The institution derives norms from its observation, which bring a system from critical to desired states.

Extended agent abilities come at a cost: an agent can only react to parameters it has observed. This might be useful in critical situations, but in other situations, there is a large observation overhead that is simply not necessary. Therefore, we enable the agent's observer to adapt the set of parameters and their update intervals to the needs of the current situation.

6.2 Systematisation

In the following sections, we will first introduce an architecture that enables us to add all the above-mentioned features to the agent in a level-based way. Not all levels need to be implemented for all types of agents. Instead, we decompose the agent's decision-making process into different levels; the more the number of levels implemented in an agent, the more complex its decisions can become. This chapter

is ordered by growing agent awareness and, consequently, increasing agent abilities. Starting with agents that have no notion of trust and are, thus, static in their social behaviour, we introduce agents that have a static notion of trust. The next step in the agent hierarchy is to make the agents trust-adaptive in order to ensure optimisation. Further improvements to adaptivity can be reached by enabling agents to learn, as has been demonstrated by the two example implementations. The next level is to make agents act strategically: agents are enhanced by prediction methods and the ability to consider institutional norms in their decision-making process. Additionally, they are able to improve the usage of their communication bandwidth by adapting not only their controller decisions, but also their observer actions to the current situation.

6.3 Adaptive Architecture Model

Agents can be distinguished by the amount of information available to them and the solution quality that they can reach. In Figure 6.2, we introduce an agent architecture featuring three agent levels, which corresponds with the agent types that will be dealt with in the following sections. Not all agent types need to implement all levels of the architecture. Agents can be built using one, two or three levels, depending on the level of awareness and decision complexity. Moreover, level-based architecture is used to decouple the different decision mechanisms located in each of the levels. Using the principle of subsidiarity in our agent architecture, we are able to reduce the complexity of agent design. Moreover, the level-based agent design increases the benefits of adaptivity by making not only the agents, but also the agent architecture adaptive. For instance, agents can decide to activate a higher level in order to make better decisions at run-time. Analogously, agents can switch off the usage of parts of their architecture in order to reduce the observation overhead and the decision complexity in non-critical situations.

The productive level is the basic level of each agent designed according to the architecture introduced in Figure 6.2. It is able to interact with other agents based on trust levels. In the productive level, there exists a simple mapping between the internal situation that has been observed by the observer component and the productive interaction parameters selected by the controller component. The productive interaction of the productive level's controller in the TDG is achieved through two roles: submitter (decision of which agent to give work units to) and worker (decision of whose work units to accept for processing). The parameters used in this single

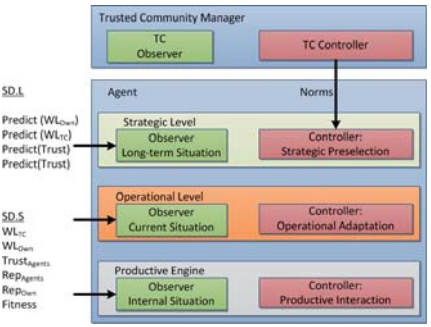


Figure 6.2: Architecture combining agent types

level are predefined by the user; they are very coarse and static as there is no way to adapt the behaviour to changing situations or to optimise parameters at run-time.

Therefore, we introduce an operational level that can be used on top of the productive level in order to choose parameter boundaries based on the analysis of the current situation. The behaviour chosen by the operational level determines the configuration of the productive level. The observer of the operational level is able to categorise the situation based on the observable environmental parameters (workload in the Trusted Community, trust levels of available agents, own trust level). Based on the determined situation, the controller of the operational level chooses a suitable behaviour. Using a mapping of situation, behaviour and outcome reached in a certain situation with a selected behaviour, the operational level of the agent is able to learn which behaviour is most suitable for a certain situation. This learning and optimisation is done at run-time. The behaviour selection is a short-term decision based on the current situation. Thus, it may be worthwhile to adopt a longer-term perspective on the system's environment. This is realised in the optional strategic level. The observer of the strategic level is able to analyse previous situations, and thereby determine trends and make predictions for the future. Therefore, the controller of the strategic level is able to act proactively before the situation actually takes place. This proactive strategic decision is given to the operational level in terms of pre-selected behaviour. Thus, the strategic level diminishes the operational level's set of possible behaviour by only allowing behaviours that conform to the long-term strategy.

As an institution outside the agent, the norm manager (for instance, the Trusted

Community manager of an explicit TC) needs to be able to create norms in order to constrain the behaviour of the agents. These norms can be used as policies filtering the decisions of the strategic level. The agent is still able to decide the extent to which it follows the rules given by the institution. As the inclusion of the norm is realised at the top level, the decision of whether or not to stick to the norm is a strategic one. The productive and operational levels are not concerned with this decision at all as they only receive the already predefined behaviour (including norm-filtering) from the strategic level.

6.4 Agent Type Hierarchy

Agents can be classified in two dimensions:

- The information space—i.e. the set of data available to the agent
- The configuration space—i.e. the set of parameters determined by the agents as well as their granularity and update frequency. Thus, we define the *control flow* of an agent to the productive entity as the flow of information per time unit (e.g. in bit/sec).

A growing information space leads to a knowledge increase, which is also in concordance with growing complexity in the agent's decisions. Nonetheless, a broad information space enables the agent to cope with a larger configuration space. If the configuration space itself is similar for all agent classes, they can still differ in the quality of the solution. For instance, having more information on a situation can result in a more suitable solution for this situation even though it might occupy the same configuration space. Additionally, a broader information space can be used to adapt to changing situations more quickly because the recognition of this situation change happens faster. For instance, having long-term information of situations enables agents to predict future system states and, thus, to adapt proactively before the actual situation change takes place.

We define the aggregated trust value $T_{i,j}$ of an agent A_i in an agent A_j as a weighted sum of the reputation rep_j of agent A_j and its own experience $E_{i,j}$, which contains an aggregation of agent A_i's experiences with agent A_j (see Chapter 5 for details).

In the TDG scenario, the agent's configuration space is broken down into two parameters (TT_i^{sub} and TT_i^{acc}) in the two roles of an agent A_i. The submitter role deals with the question of which other agent should be given work units (WU). The

submitter trust threshold TT_i^{sub} determines the threshold up to which agents are regarded as trustworthy. Agent A_i only submits a WU to agent A_j if $T_{i,j} \geq TT_i^{sub}$. In the worker role, A_i decides whether or not to accept a WU that has been offered to it by agent A_j. A_i will only accept a WU offer by A_j if $T_{i,j} \geq TT_i^{acc}$. In this example, all agent types determine the same configuration space parameters TT_i^{sub} and TT_i^{acc}. Depending on their information space and decision-making complexity, the difference between the configuration spaces of the agents is the update frequency of these parameters as well as the granularity of the chosen parameters—e.g. the sampling resolution of the two parameter values.

Figure 6.3 presents a structured overview of the different agent types introduced in this thesis. In general, the more the information available to an agent, the higher the solution quality of the agent behaviour. The information space of trust-neglecting agents consists solely of workload (WL); thus, they only reach a low behaviour solution quality. In contrast, trust-considering agents know both, WL and the reputation (Rep) of possible cooperation partners. Therefore, they are able to reach better solutions than agents that possess less information. Trust-adaptive agents have access to the short-term situation description SD.S, which contains not only WL and Rep, but also the average reputation values in the agent group relevant to them and their own reputation value. Instead of just one static strategy, they possess different strategies suited to different situations. Thus, they are able to adapt their behaviour to the situation and reach an even better behaviour solution quality than do trust-considering agents. The best solutions are found by trust-strategic agents. These agents have access to the long-term situation description SD.L and, thus, are able to predict future situations. This prediction enables them to adapt not only reactively (like trust-adaptive agents), but also proactively before the predicted situation even occurs. The four different agent classes will be introduced in detail below.

6.5 Information Spaces of TDG Agents

This section provides an overview of the different parameters that agents can use in the TDG application scenario.

In general, the information space of agent A_i can consist of

- WL_i workload of agent A_i
- Number of jobs/WUs that have to be distributed FW_{own}
- Rep_i—i.e. its own reputation values in the eyes of the other agents

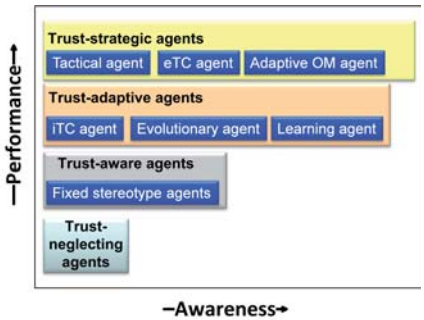


Figure 6.3: Agent hierarchy based on information and solution quality

- Rep_j values of agent A_j, j in the set of all available agents
- $Trust_{i,j}$—i.e. the trust value that agent A_i assigns to agent A_j, j in the set of all available agents
- WL_{total}—i.e. the average workload aggregated among all agents
- WL_{TC}—i.e. the average WL within the TC (all agents being above a predefined threshold)
- $Fitness_i$ or other performance metrics (e.g. from grid computing)
- Predict(Trust)—i.e. prediction of trust, expected behaviour in the next agent interaction
  - 2nd order exponential smoothing
  - Neural network
- Predict(WL)—i.e. prediction of WL (either total or TC)
- Predict(Rep)—i.e. predicted reputation of an agent based on former interactions.

It can be said that information causes overheads in terms of communication as well as calculation time. A communication overhead is produced whenever information is aggregated among several agents, because all of these agents need to be asked and need to answer in order to determine the requested information. Since we have a

global reputation database, we can also think about using this information source to store further data (like workload information), even though this data needs to be updated often (thereby causing communication overheads, too). The predictions used for the trust-strategic agent are computationally intensive. Therefore, we must carefully analyse in which situations and with which frequencies predictions are made available. Therefore, a feedback loop from the controller to the observer can be used to control which information is currently relevant to the agent and which information does not need to be perceived at the moment. Thus, the communication and computation overheads can both be reduced without losing relevant information.

6.6 Trust-neglecting Agents

Trust-neglecting agents do not have access to trust or reputation information. Therefore, they represent agents having a simple grid strategy—e.g. taking into account only the current workload and waiting queue of available agents. We use trust-neglecting agents only for reference and, therefore, do not present them further in this thesis but rather refer to the related work in the area of desktop grid computing as presented in Section 3.1.

6.7 Trust-aware Agents

An agent is called trust-aware, if the trust value (e.g. reputation, own experience or both combined) is considered in the agent's decision regarding with whom to cooperate. Therefore, the parameters TT^{sub} and TT^{acc} are simple constant values, as can be seen in Figure 6.4.

Examples of such a simple, trust-considering agent are static agents, such as the agent stereotypes: free-rider, egoistic agent, rational agent and altruistic agent. Each agent stereotype is defined by different TT_i^{acc} in the worker role as is depicted in Figure 6.4. An altruistic agent has a low TT_i^{acc}, which means that it accepts work units from nearly all agents. A rational agent accepts a work unit only if its owner has a certain trustworthiness (TT_i^{acc} is medium). An egoistic agent has a high TT_i^{acc}, which means that it accepts work units from only very trustworthy agents. free-riders have a TT_i^{acc} that is beyond the scope of the parameters. This means free-riders never accept any units, not even from highly trustworthy agents. Thus, the agent stereotypes have been defined solely in terms of different worker thresholds (see Figure 6.5). The submitter threshold is not regarded here.

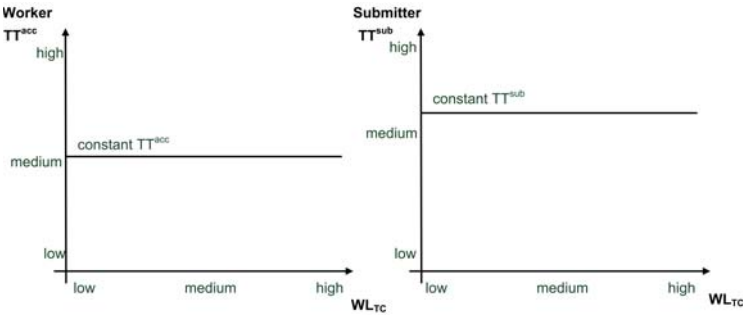


Figure 6.4: Trust-aware Agent

6.7.1 Egoistic Agent

In the TDG, egoistic agents accept work units but have a high probability to cancel them during computation. Therefore, a simplified trust threshold function cannot be drawn here. Moreover, Figure 6.5 indicates the approximated trust threshold of egoistic agents, which can be interpreted as follows. The probability that an egoistic agent accepts and completes a work unit is low. This can be performed on purpose as a strategy of egoistic agents. If other agents do not control which work unit has been successfully completed, its misbehaviour can remain undetected. Moreover, advanced strategies like delivering fake results can be performed by this agent type in order to reduce its own effort to become a member of the system.

Egoistic behaviour does not necessarily have to be on purpose. In the TDG, for instance, agents can represent a user that suddenly withdraws the resources from the agent. In the worst case, the user simply turns off the PC, leaving no possibility for the agent to inform the grid members that it is going offline and, therefore, cannot finish the queued work units. No matter how cooperatively the agent would like to parameterise its behaviour, the user as an external factor can force it into egoistic behaviour.

Technical problems can also result in egoistic agent behaviour. For instance, in the TDG, agents rely on the internet as a communication channel. Therefore, package loss, delays, low bandwidth or even lost connections can result in a kind of agent behaviour that outside agents rate as egoistic.

Regardless of whether the egoistic behaviour is a learned strategy or the result of factors external to the agent, it is a behaviour that disturbs the overall system

and, therefore, needs to be detected. Detected egoistic behaviour can be countered by isolating the misbehaving agent in order to protect the overall system from its actions or even by incentivising a change of behaviour to a more cooperative one.

6.7.2 Free-Rider

A free-rider is an agent that never cooperates with other agents. In the TDG, a free-rider asks other agents to compute work units, but does not accept work units from other agents for cooperation. This corresponds to the free-riders in the Gnutella network [19], which use data from other agents without giving back any data. In Figure 6.5, the free-rider has a trust threshold that is larger than the highest trust value defined. Hence, the free-rider never accepts work from others.

Adar [107] has evaluated that 70% of Gnutella users share no files and 90% of the users answer no queries. Therefore, free riding is a serious issue in open systems that needs to be tackled.

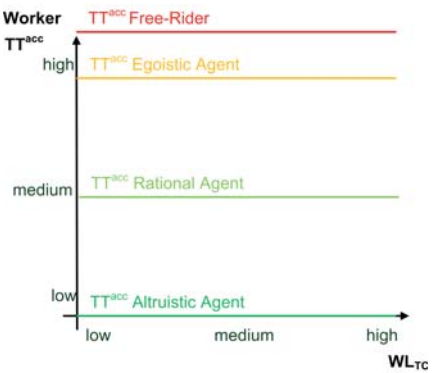


Figure 6.5: Agent stereotypes are instances of static agents

In this thesis, we present the adaptive agent architecture and trust-based self-organisation mechanisms that are able to prevent the system from being exploited by misbehaving nodes like egoistic agents and free-riders.

6.8 Trust-adaptive Agents

Agents are called trust-adaptive if they do not adhere to constant trust thresholds but are able to adapt these thresholds autonomously at run-time. This autonomous adaptation is implemented depending on the short-term situation description $SD.S$ that the agent has gathered from its observer's information space. In the TDG, $SD.S$ currently consists of the current workload in the TC (WL_{TC}), it's own reputation value Rep_i and the reputation value Rep_j of its possible cooperation partner A_j.

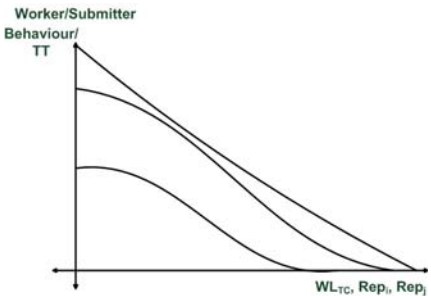


Figure 6.6: Adaptive agents can choose a suitable strategy

Based on its $SD.S$, the trust-adaptive agent A_i decides which behaviour graph is best suited to the current situation. The set of possible behaviour graphs is given by the system designer. Figure 6.6 shows an example of such possible behaviour graphs based on different parameters. These behaviour graphs depend on different parameters (WL_{TC}, Rep_i, Rep_j), which of the behaviour graphs is currently used by the agent is determined at runtime. This function deciding which behaviour graph is suited in which situation is also predefined. One example of the class of trust-adaptive agents is the iTC (implicit Trusted Community) agent used in the TDG, which will be illustrated in the following.

6.8.1 iTC Agent

The iTC agent is able to adapt its behaviour in both worker and submitter roles.[1]

In the worker role, the iTC agent A_i determines the acceptance threshold TT_i^{acc}

[1]Work on this has been conducted in part in cooperation with Lukas Klejnowski in the context of the DFG research unit OC-Trust (FOR 1085).

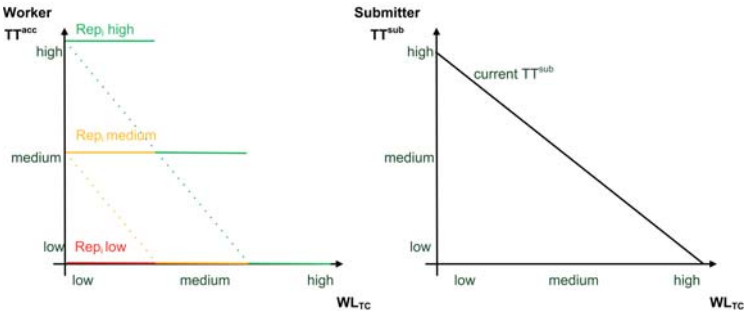


Figure 6.7: iTC agent thresholds

WL_{total}	Rep_i **low**	Rep_i **medium**	Rep_i **high**
low	TT^{acc} low	TT^{acc} medium	TT^{acc} high
medium	TT^{acc} low	TT^{acc} low	TT^{acc} medium
high	TT^{acc} low	TT^{acc} low	TT^{acc} low

Table 6.1: Discretised trust threshold matrix of adaptive agent A_i in worker role

based on Rep_i and WL_{TC}. This means that $TT_i^{acc} = f(WL_{TC}, Rep_i, Rep_j)$. This worker decision graph can also be modelled as a table:

Each agent A_i which has been offered a work unit by agent A_j reacts according to Figure 6.7 as described in Table 6.1.

In the submitter role, the decision is made regarding which agents are eligible for a work unit offer. We use an adaptive ranking mechanism that orders the available agents according to

1. their reputation Rep
2. their performance level PL
3. their workload WL.

This adaptive ranking mechanism of the iTC agent is done as follows:

The threshold TT_{sub} within the adaptive ranking mechanism is determined according to Fig. 6.7. Both, the worker and submitter strategies of the adaptive agents, realise local algorithms. A system where several agents act according to these local mechanisms is called an implicit Trusted Community (iTC). Each agent's view of

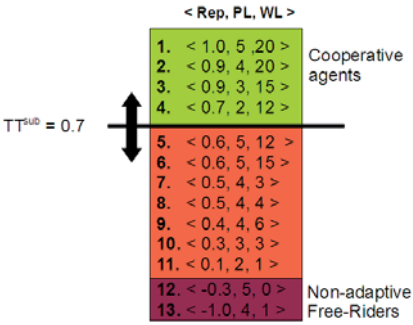


Figure 6.8: Adaptive ranking table

the TC is the set of agents above TT^{sub} (e.g. the green area in the adaptive ranking in Figure 6.8). The global view of the iTC is now the set of agents that is within all agents'TCs, as described in Section 5.2.2.

$$ETA_{WU,Worker} = Queuelength_{Worker} + \frac{Work_{WU}}{PerformanceLevel_{Worker}} \tag{6.1}$$

Equation 6.1 shows the estimated time of arrival (ETA) algorithm used by submitters to determine the most suitable agent for a work unit. All agents that are suitable workers according to their aggregated trust level, as presented in 6.7, are ordered by the time in which they are expected to deliver the work unit. This depends not only on the current queue length of the potential worker, but also on the potential worker's performance level (computation speed) and the computational complexity of the work unit. For instance, if an agent has 75 calculation steps to work off until its queue is empty and the work unit to be distributed needs 25 time steps with its performance level, the ETA of this agent is 100.

An example of such a rating is shown in Table 6.2, where the agents will be asked in the resulting order to calculate the work unit. This shows that, by first filtering the available agents by T^{sub} and then ranking the remaining agents according to the ETA strategy, the most suitable agent does not necessarily have to be the most trustworthy. Moreover, the ETA algorithm leads to a load balancing by evenly distributing the work units among a set of trustworthy agents rather than always demanding the most trustworthy agent. This load balancing leads to performance

Ranking Position	ETA	Trust
1.	50	0.8
2.	100	0.9
3.	150	0.9
4.	200	0.8

Table 6.2: ETA ranking table

improvement for the agents that use it and, if a sufficient number of agents use ETA, also for the overall system, as will be presented in Chapter 7.

6.8.2 Evolutionary Agent

Another example of the class of trust-adaptive agents is the evolutionary agent. The aim of this implementation was to be able to learn optimal decisions within the large solution space of trust-adaptive agents. Evolutionary agents are initialised with random behaviour functions. In contrast, learning agents 6.8.3 use the decision mechanism of iTC agents as starting points; therefore, they search the solution space in a different way.

An evolutionary agent is characterised by 10 genes defining the behaviour as well as the information perception (genes of other agents to be considered or ignored) of the agent. Based on this behaviour, agents interact with other agents. If two evolutionary agents meet, they exchange a certain set of their chromosome information. In general, the agent with the highest fitness is copied by the less successful agent. Additionally, mutations are possible in order to add some random changes in the agents' chromosome cycle. This agent type has proved highly successful, especially in very heterogeneous agent systems. A prerequisite of the evolutionary agent is that at least 25 percent of the agent population in the system needs to be evolutionary agents in order to achieve a dominant chromosome structure being spread along the set of all evolutionary agents.

The evolutionary agent is an approach to the run-time optimisation of trust-based interactions. Based on [108], we aimed at creating an evolutionary agent model that enables cooperation and trust-building. This model has been adapted for our application scenario TDG in order to make it the basis of the worker and submitter decisions.

In this section, we will introduce the general design of this agent type as well as focus on how this design is used to make the agent decisions in the application scenario TDG in both, the worker and submitter roles.

The evolutionary agent is defined by a chromosome structure that contains 10 genes. The bit values of the genes represent the alleles of the agent. The combinations of these genes influence the behaviour and decisions of the agent. The genes contain instructions that are interpreted as characteristics of the agents. Each agent follows its own strategy that is induced by the sequence of bits in its chromosome.

Table 6.3 describes the chromosome and the genes it contains, which encode the behaviour of the agent. The chromosome consists of 10 genes. Gene 1 is used to define the general character of the agent. Genes 2 to 5 define how the agent comes to trust decisions, whereas gene 10 marks the actual trust decision. Genes 6 to 9 define which characteristics of other agents are taken into account for decision making.

Each gene can have the value 1 or 0. Thus, there exist 2^{10} different types of evolutionary agents. Gene 1 defines the character of the agent. It decides whether the agent is an egoist (E) ($G_1 = 0$) or a cooperator (C) ($G_1 = 1$), and tries to do its job as well as possible. This means that an egoist will accept work units if it trusts its partner, but has a high probability of aborting them before they are finished. A cooperator will accept work units if the number of work units in its queue is not too high and it trusts its partner. It will then try to process the work units and avoid aborting them. Genes 2 to 5 influence the decisions of the agent regarding whether or not it will trust its partner. These genes determine the signals to which the agent must pay attention:

- Its own intentions (Gene 2)
- Reputation of the partner (Gene 3)
- Fitness of the partner (Gene 4)
- Workload of the partner (Gene 5)

Genes 6 to 9 determine how to interpret these signals. If the value of the signalling gene (2--5) is 0 then the corresponding gene (6–9) is ignored.

The following example (cf. Fig. 6.9) will illustrate the signals and the corresponding behaviour of an agent $A_i$as it decides how to interact with an agent A_j: We assume that the values of gene 2 and gene 4 are 1 and the values of gene 3 and gene 5 are 0. Since gene 2 has the value 1, its own intentions (Gene 1) will be included in the process of building trust. This depends on the value of gene 6. If

Genes	Alleles	Rules
1	0	E (Egoist: aborts Work Units (WUs) with high probability).
	1	C (Cooperator: tries to process WUs as well as possible).
2	0	Don't involve your own intentions (G_1) into building trust.
	1	Involve your own intentions (G_1) into building trust, given (G_6).
3	0	Ignore the partner's reputation.
	1	Pay attention to the partner's reputation, given (G_7).
4	0	Ignore the fitness of the partner.
	1	Pay attention to the fitness of the partner, given (G_8).
5	0	Ignore the workload of the partner.
	1	Pay attention to the workload of the partner, given (G_9).
6	0	Assume that others are the opposite of your gene (G_1).
	1	Assume that others are the same as your gene (G_1).
7	0	Distrust those who have a relatively high reputation. Trust those who have a relatively low reputation.
	1	Trust those who have a relatively high reputation. Distrust those who have a relatively low reputation.
8	0	Distrust those who have a relatively high fitness. Trust those who have a relatively low fitness.
	1	Trust those who have a relatively high fitness. Distrust those who have a relatively low fitness.
9	0	Distrust those who have a relatively high workload. Trust those who have a relatively low workload.
	1	Trust those who have a relatively high workload. Distrust those who have a relatively low workload.
10	0	Distrust everybody (reject all WUs).
	1	Trust everybody (try to process all WUs).

Table 6.3: Chromosome structure of the evolutionary agent

gene 6 = 0, then the agent assumes that the partner is the opposite of the agent's gene 1. If gene 6 has the value 1, the agent assumes that the partner's gene 1 has the same value as the agent's gene 1. The assumption that the partner is a cooperator will increase trust while the assumption that the partner is an egoist will decrease trust. Since gene 4 = 1, the agent pays attention to the partner's fitness. If gene 8 = 0, the agent will trust those that have a lower fitness than itself and distrust those with a higher fitness. If gene 8 = 1, then the behaviour is inverted. The number of signals that recommend trust are normalized with the total number of signals to which the agent pays attention, so that it results in a value between 0 and 1. This value represents the probability that the agent will trust a partner. If an agent pays attention to none of the four signals, then gene 10 decides if the agent will trust the partner. Based on total trust ($G_{10} = 1$) or total distrust ($G_{10} = 0$), the agent will accept all work units or reject all work units respectively.

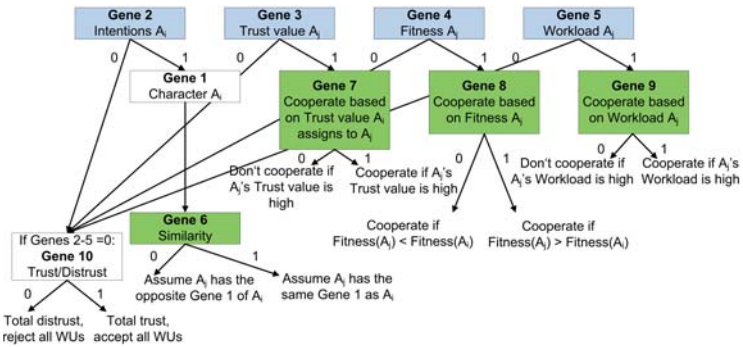


Figure 6.9: The chromosome structure of evolutionary agent A_i decides how to treat agent A_j

Gene Initialisation The genes of each agent are set randomly at creation. For each gene, there exists a parameter that has a value between 0 and 1. This parameter determines the probability that the corresponding gene is set to 1 when an evolutionary agent is created. Regarding the population of all evolutionary agents, the value of the parameter corresponds to the expected number of agents that have the corresponding gene equal to 1. The standard value is 0.5, so that each gene of half of the evolutionary agents takes the value 1.

Evolution and Spreading of the Genes To ensure the evolution of the genes

and the corresponding trust strategies, a gene exchange between the agents can occur when they come into contact. In this process, the genetic instructions will be transferred with a fixed probability from the agents with a higher fitness to those who have a lower fitness. If two agents interact, the partner with the lower fitness replaces a random part of his chromosome structure with a part of the fitter partner's chromosome. In this procedure, each bit in a chromosome can be replaced independently of the others. Whether a bit of the agent with a lower fitness is replaced by the fitter agent's bit is determined by the recombination probability. In this case, the recombination probability was 50%.

Furthermore, to increase heterogeneity, mutation occurs during the gene replacement. Thus, during the transfer of the genes from the fitter agent to the weaker agent, random copying errors (mutations) can arise. The probability that a copying error occurs during a gene replacement is 1%. This value allows for sufficient heterogeneity without affecting the stability of evolution. Thus, evolutionary agents are able to leave local optima in their fitness landscape and have a higher probability to reach the global optimum.

Worker: Acceptance of Work Units To decide whether or not a work unit is accepted, the chromosome structure is analysed. If the agent pays attention to more than one signal of the partner, each bit is considered equally. Partners that send out mixed cooperation signals will be trusted with a corresponding probability. Let us assume that three of the signal genes are used, where two show trust in the partner and the third distrust. In this case, the agent will trust the partner and accept the work unit with a probability of $\frac{2}{3}$. Here, the signal genes are weighted equally.

Submitter: Distribution of Work Units In our grid agent model, a ranking of the suited worker agents is created to distribute the work units [109]. This is achieved by calculating a score of reputation, fitness and workload whereby genes 3, 4 and 5 determine which of these characteristics are included and genes 7, 8 and 9 determine whether the total score will be increased or decreased. After creating the ranking, its own work unit is offered in order of ranking to the other agents until one of them accepts or the submitter has to process the work unit itself.

6.8.3 Learning Agent

The evolutionary agent uses the complete solution space of the class of trust-adaptive agents, which results in the potential of finding the perfect solution over time, but also in a less stable solution. This is because evolution does not stop but goes on at run-time, and the agents continuously adapt their behaviour. Therefore, the

learning agent is able to learn as well as to use solutions predefined by the designer. Therefore, the potential of learning can be used without loss of the stability of the solutions already found. In Section 4.1.1, we have analysed which learning techniques from related work are suitable for our application scenario. As the adaptive agent architecture is generic, we also want to make way for the usage of other learning techniques within this architecture. Therefore, the learning agent framework has been developed.

Learning Agent Framework

In order to to extend the trust-adaptive iTC agent [105] in a generic manner, we built a framework that is especially necessary because we tested different learning techniques and could reuse the essential parts of the code base. We encapsulate the learning function and the reward metrics in order to exchange them easily or to integrate further functions or metrics.

In this approach to learning agents, we concentrated on the worker role, thereby extending the adaptive mechanism used by iTC agents via a function that is learned and optimised by the agents at run-time.

The framework is built upon the abstract classes, *AbstractModularAgent*, *AbstractLearningFunction* and *AbstractRewardMetric*. New learning agents, learning techniques and reward metrics simply need to extend these classes in order to be used in the system.

The framework structure is depicted in Figure A.1.

Within the framework, the action sequence of a learning agent is as follows:

Initialisation

The agent class that extends the *AbstractModularAgent* is initialised at the beginning of the simulation. During the initialisation process, an instance of a reward metric that extends *AbstactRewardMetric* is created. Similarly, a learning mechanism instance extending *AbstractLerningFunction* is created. Both instances are referenced by the learning agent.

Worker Role

Each time that an agent is asked to process a work unit, the function *getTTack* of the learning agent is called. This function asks the learning mechanism of the agent for the parameter and returns it to the calling agent.

Reward Propagation

In each time step, the learning agent calls the *step* method of the reward metric instance. The statistical data of the agent is called, the reward is calculated, and finally returned to the learning mechanism using *propagateReward*. The learning

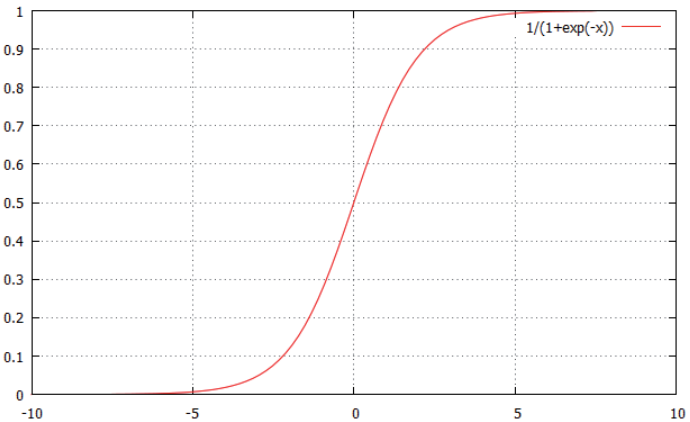


Figure 6.10: Sigmoid function

mechanism then updates its values accordingly.

CACLA Implementation

The core of CACLA are two neural networks (see Section 4.1.1) that store the V and Ac functions. According to Hasselt [5], the networks need 12 hidden neurons that are fully connected with the two input neurons and one output neuron. Neurons are represented by the class *neuron*, which implements the interface *ISynapseHandling*. Each neuron needs an activation function for initialisation. This has been realised as a sigmoid function (Equation 6.2). The input and output neurons use a simple linear function represented by the mean of the input values.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{6.2}$$

The sigmoid function is preferred for hidden neuron activation because it is continuous and differentiable. This is important if a neural network is used for backpropagation learning [110].

The *NeuralNetwork* class interconnects the created nodes using *Synapse* classes and sets the initial weights (used here: weight 1). The structure of the implemented neural network is depicted in Figure 6.11.

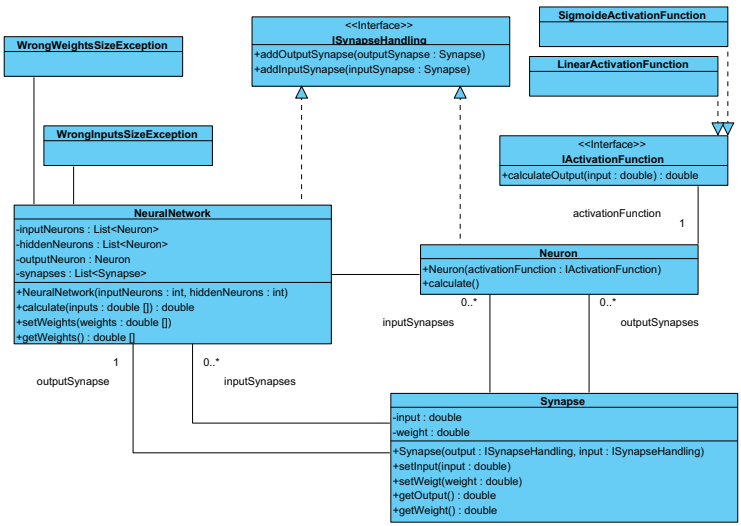


Figure 6.11: Structure of the CACLA neural network

The learning algorithm implemented in the *Learner* class, according to 4.1.1, is called each time that a reward is returned. In the TDG application scenario, this is usually each time that a job is finished. The weights of the two neural networks are read and modified according to the reward.

To integrate domain knowledge like the worker role acceptance matrix, we implemented a mechanism derived from Hedger4.1.1. With this extension, CACLA uses the worker acceptance matrix as the default function during the initial supervised learning period, and stores the received results in the *Learner* class. After a given amount of rewards, CACLA switches from supervised learning to the operational mode.

6.9 Trust-strategic Agents

Trust-strategic agents are an extension of the trust-adaptive agents. In addition to SD.S, they obtain a *long-term situation description SD.L*, which contains the trends of values like reputations or workload. These trend data can be used as a basis for predicting possible future situations. Therefore, trust-strategic agents are able to act proactively as soon as they expect the situation to change in a certain way (e.g. expect the workload to increase) before the situation change actually takes place. Therefore, trust-strategic agents do not simply react to changed situations, but react as soon as they expect such a situation to occur. We differentiate four types of trust-strategic agents:

- **Tactical agents**, which use predictions of other agents' future behaviour to adapt their behaviour towards them in a proactive manner.

- **eTC agents**, which have the ability to decide whether or not to form or join an eTC (see Chapter 5 for details).

- **Norm-aware agents**, which understand norms from an external institution and reason whether or not it is worthwhile for them to obey the norms.

- **Adaptive observation model agents**, which are able to adapt the parameters they observe and need for their currently used behaviour selection mechanism to the situation and, thus, save communication overheads.

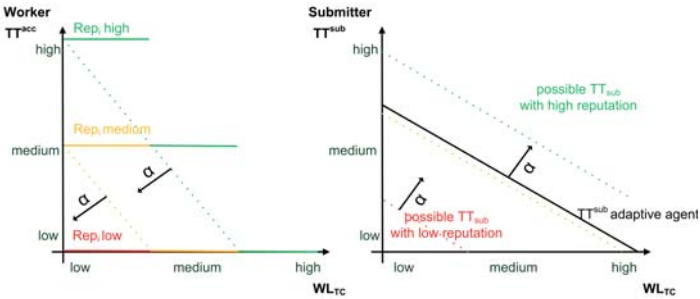


Figure 6.12: Trust-strategic agent

6.9.1 Tactical Agent: Situation Prediction

As depicted in Figure 6.12, in the TDG scenario, we are able to extend the notion of trust-adaptive agents (Figure 6.4) through a cooperation control loop. This enables agents to adapt not only if the situation has changed but also if there are cues that the situation is going to change. If such a future change in the situation is likely, the configuration space of the operational level is restricted by a strategic level in order to omit solutions that are not optimal in the current situation. Trust-strategic agents can belong to two classes: one-step tactical agents that determine the next action step, and multi-step agents that are able to determine a longer-term strategy (e.g. until the situation changes). The one-step tactical agent [111] is able to predict the cooperation partner A_j's next step by analysing previous interactions. Based on this predicted cooperation probability, the tactical agent decides how to treat A_j in the current time step.

Prediction of Agent Cooperation Based on Former Interactions

In order to take into account more than one past interaction for the cooperation decision, we enhanced our tactical agents with a history data structure storing the results of former interactions with all other agents.

We then implemented a simple prediction method based on double exponential smoothing. This prediction was used as an input for the novel prediction method based on neural networks. The results stored in the history data structure were transferred into a numeric value that was then used as input for the neural network.

The prediction approach is based on time series analysis and is used for a short-

term prediction of the cooperation partner's next behaviour. This approach is used for the initialisation of the training data set of the neural network introduced in the following. In the equations above and below, x is the value of the time series where x' is the value of the simple exponential smoothing function and x" is the value of the double exponential smoothing. x* denotes the values of the prediction by the exponential smoothing function used.

Here, α ($0 \leq \alpha \leq 1$) is a weight that determines the extent to which previous values are taken into account. A small α weights new values much higher than previous ones. Thus, the time series is smoothed to only a small extent and the adaptation occurs very fast. A high α weights old values higher, and new values are accounted for only to a small extent. This leads to a much smoother time series and, thus, to a very slow adaptation to changes in the situation. We have chosen $\alpha = 0.5$, which results in a medium adaptive and smooth function and leads to the best experimental results.

We use second-order exponential smoothing as shown in Equation 6.3.

$$x'' = \alpha * x'_t + (1 - \alpha) * x''_{t-1} \tag{6.3}$$

To compute this value, we need to determine the first-order exponential smoothing function with prediction given in Equation 6.4.

$$x^s_{t+1} = x'_t = \alpha * x_t + (1 - \alpha)x'_{t-1} = \alpha * x_t + (1 - \alpha)x^s_t \tag{6.4}$$

The reason for using second-order exponential smoothing is that, if the data implies a trend, this function is able to recognise this trend and predict the next value in the time series (see Eq. 6.5).

$$x^s_{t+1} = 2x'_t - x''_{t-1} = x'_t + (x'_t - x''_{t-1}) \tag{6.5}$$

Equation 6.5 now shows the actual prediction function of the double exponential smoothing used in the prediction function of trust-strategic agents. The tactical agent as a one-step function uses the prediction of aggregated trust values and workload to decide with which agent to cooperate in the next decision step. With this mechanism, the agent is able to act proactively, adapting its behaviour as soon as it expects a critical situation to evolve.

6.9.2 Norm-aware Agent: Consideration of Constraints

This section presents the implementation details of the norm-aware agent. This subtype of the class of trust-strategic agents is able to perceive norms from an institution. Currently, we use a norm manager as the institution that is authorised to legislate norms in case of a critical system state that cannot be observed and resolved via local view only.

Norm Formalisation

In order to consider the norms given by an institution, a norm-aware agent must 'speak the same language' as the institution. Therefore, we introduce a representation of norms and present an example of how this norm is used to solve a globally observable situation (overload). This normative framework can be used to formulate further norms and, thus, presents a basic building block for communication between institution(s) and agents. Here, we abstract from general institutional tasks by defining a norm manager. This global instance supervises the system state. Each possible norm that the norm manager is able to enact is encapsulated in a class implementing a norm interface. This ensures that the 'language' constraints necessary for a norm to be understood by all agents are met.

If the norm manager detects a crucial system state being present or approaching, he activates those norms that he expects to be suited to lead the system out of this state. The activation of the norm is broadcasted to all agents registered for the norm. The agents then decide whether or not to obey the norm.

If the system situation returns to the stable desired state, the norm manager deactivates the corresponding norm and broadcasts this norm to all agents.

Pre- and Post-selection

In general, the consideration of norms can be considered by the agent in two ways: *pre-selection* and *post-selection*.

Pre-selection means that the agent's solution space—i.e. the set of all possible solutions—is reduced before the agent enters its decision-making reasoning process. Thus, all solutions that the agent develops are compliant with the norm.

Using *post-selection*, the agent will first perform its standard decision-making process. In the next step, it will check whether its solution is compliant with the norm. If not, the agent needs to recapitulate its decision process with changed parameters in such a way that a new solution is computed. This solution is then checked again for compliance and so forth.

Both pre- and post-selection lead to norm-compliant behaviour if the agent decides to obey a norm. But, the designer must ensure that the solution space is not empty. Moreover, we do not want norms to restrict the agent's solution space to only one possible solution as this would be remote controlling of the agent by the institution. In order to enable agents to consider norms, we extend the adaptive agent architecture in such a way that the agent is able to receive norm activation or deactivation messages and to reason about its own actions based on this information. Here, we use a top-down approach, which means that the MAS and the normative system are connected but can be distinguished from each other. This is useful because the norm-aware agent extends the MAS that we already have developed. Thus, instead of having to start from scratch, we can extend the existing concepts and classes. We decided to use pre-selection in order to reach a fast computation of the agent's decision without repetition of the decision-making process if the decision does not match the norms.

Overload Workload Norm In order to demonstrate an example of how norms can be implemented in the norm-aware agent, we introduce the *overload workload norm*. The workload is the amount of work to be computed in the system. An overload situation occurs if the workload is higher than the amount of work that can be worked off by the agents in the system. This can either be a real overload where the computational capacity of all system members forms less power than is necessary to cope with the 'to-do pile' of the overall system or a relative overload where the parameterisation of the agents does not allow the system to cope with the overload. In the first case, we as system designers have no chance to change the agents in order to bring the system to stable workload state again. In the second case of relative overload, there is room for improvement by better adapting the agents to the system situation. In this case, we concentrate on situations where there is a possibility to bring the system into a stable state by adapting agents' parameters.

We consider a situation where the agents try to distribute more work units than can currently be worked off by the agents. Such a workload peak can be dealt with by adapting the agents' local interaction thresholds in such a way that the agents are more likely to accept work units. Such a mechanism will not only lead to a faster handling of the workload peak, but a trust breakdown can also be prevented: A trust breakdown is a system situation where agents do not trust each other any more and, therefore, do not find any cooperation partners. A longer overload of workload, in general, can lead to such an undesired system state. A denied request for cooperation (computation of a work unit) is rated as bad behaviour. Hence, the

greater the overload in the system, the more likely the possibility that agents are rated badly because they will decline if they are already occupied (with own or other agents' work units) and their queues are full. Therefore, if a norm manager finds a way to make agents more cooperative in overload situations, we can not only counter the overload situation, but also prevent a trust breakdown situation.

The overload workload norm is activated by the norm manager if it detects an overload situation. If the overload workload norm is active, agents change their cooperation threshold ($TT^a cc$) to cooperative behaviour, thereby making the acceptance of an offered work unit more likely.

Norm Life Cycle This chapter shows an example of a norm life cycle, starting with the prerequisites for activation, the active 'living' state and the deactivation phase. As an example, we use the trust breakdown norm. A trust breakdown exists in a system where no agent trusts any other agents and, thus, no cooperation can take place. Figure 6.13 shows the life cycle of a norm as a UML sequence diagram.

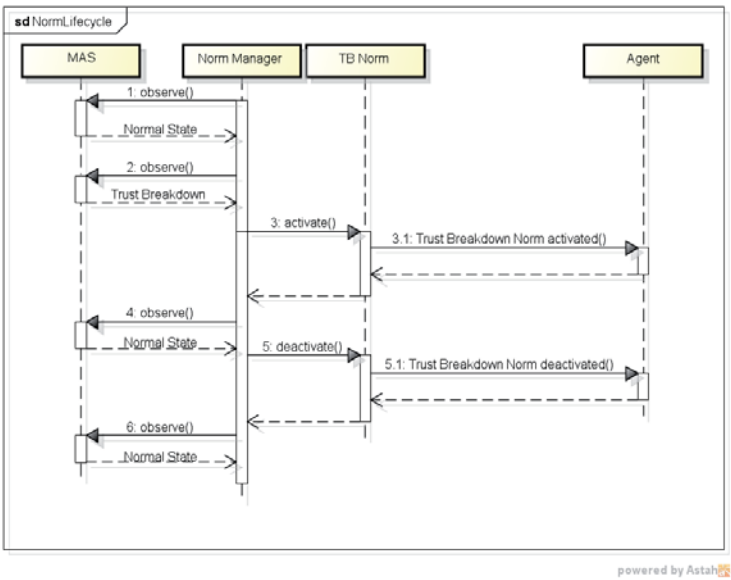


Figure 6.13: Norm life cycle

The norm manager monitors the global state of the MAS (1., 2., 4., and 6.). If a trust breakdown is observed, he will activate the trust breakdown norm. This is broadcasted to all agents in the system. Agents that want to act in a norm-compliant manner react to this norm, for instance, by changing their trust threshold $TT^a cc$ in such a way that they are more likely to accept work units. This is necessary in a trust breakdown situation because all agents have a trust value that is too low to find cooperation partners.

As soon as the norm manager observes that the undesired system state of a trust breakdown has been resolved, he deactivates the trust breakdown norm and communicates this to the agents. In turn, these agents do not have to obey the trust breakdown norm. Thus, they can once again change their parameters according to their own local strategies.

This norm life cycle is generic. If another norm (e.g. overload norm) needs to be activated, the mechanism is analogue.

Norm Manager The norm manager is a hierarchical institutional component that observes the system at run-time. This could, for instance, be the Trusted Community Manager of an eTC. In our current implementation, the norm manager is a singleton class, which means that there can only be one instance in the system. It may be possible to extend the system in such a way that more than one norm manager exists, but the designer will have to be even more careful with respect to conflicts. For instance, if there is no mechanism to ensure that norm managers do not have contradictory norms, the agents need to have a mechanism to decide whose norm to obey. Therefore, we concentrate on systems with a singleton norm manager class. The norm manager continuously observes the system and becomes active only if the system is in a critical situation.

The actions that the norm manager performs at each time step (tick) are best described as a pipe from the pipe and filters pattern, which means that they are worked off in chronological order, and the step function can easily be extended.

Listing 6.1: calculatoverloadWLVariables

```
1
2 private void calculatoverloadWLVariables
3         (Vector<Float> socialValueVector) {
4         ObservableWL observerWL =
5         new ObservableWL(1,socialAgents,
6         TEMAdapter.getInstance().getCurrentTime(),97);
7         Collection<ObservableValue> valuesWL =
```

```
8           observerWL.getValues();
9           Iterator iterWL = valuesWL.iterator();
10          while (iterWL.hasNext()) {
11                  ObservableValue value =
12                  (ObservableValue) iterWL.next();
13                  socialValueVector.add
14                  ((Float)value.getCurrentValue());
15          }
16          avgWL = 0;
17          for(float f : socialValueVector){
18                  avgWL = avgWL + f;
19          }
20          avgWL = avgWL/socialValueVector.size();
21  }
```

Usually, a norm manager will observe the different parameters that determine critical system situations. Therefore, in the following example, we will combine the OverloadWL norm (Listing 6.1) and the trust breakdown norm (Listing 6.2). ´

Listing 6.2: Calculate trust breakdown variables

```
1
2   private void calculateTrustbreakdownVariables
3   (Vector<Float> socialValueVector) {
4           ObservableReputation observerRep =
5           new ObservableReputation(1,socialAgents,
6           TEMAdapter.getInstance().getCurrentTime(),97);
7           Collection<ObservableValue> valuesRep =
8           observerRep.getValues();
9           Iterator iterRep = valuesRep.iterator();
10          while (iterRep.hasNext()) {
11                  ObservableValue value =
12                  (ObservableValue) iterRep.next();
13
14                  socialValueVector.add
15                  ((Float)value.getCurrentValue());
16          }
17          avgRep = 0;
```

```
18          for(float f : socialValueVector){
19                  avgRep = avgRep + f;
20          }
21          avgRep = avgRep/socialValueVector.size();
22  }
```

In order to evaluate the ObservableWL norm, the norm manager creates an observable of the workload values of the agents it wants to observe. It then iterates over the collection all workload values, calculates the average workload of these agents, and decides based on this aggregation if an overload situation exists. Similarly, it could also determine whether there is an imbalance of workload—for instance, if a part of the system has an overload situation whereas another part of the system is idle.

Analogously, the trust breakdown norm is activated if the average value of the global reputation value of the observed agents is below a threshold (see Listing 6.2).

Constraints

As introduced in Section 12, we implemented our own precondition version of constraints. On the one hand, this frees us from our dependence on proprietary libraries. On the other hand, our own version of constraint implementation is compliant with the norm can easily be extended to all kinds of constraints. Therefore, we realised the interface *IConstraint*, which is implemented by all our constraints. There exist two types of constraints which we use here: *ValueConstraint* and *BooleanConstraint*.

The ValueConstraint is given two values, A and B, and an operator, and it returns whether the operator (valueA, valueB) is true. We implemented the operations that we needed for the OverloadWL and the trust breakdown norm, but the method can easily be extended or overloaded for further operators.

A BooleanConstraint can be used if the constraint test has to determine whether two Boolean values are equal. Of course, this is a specialisation of the *valueConstraint*, but as it often occurs, it is worth implementing as an own class by extending the IConstraint.

The norm manager uses a ConstraintCollection of preconditions based on these constraint classes. According to the composite pattern, constraints can be composed of other constraints as shown in Figure 6.14.

The ConstraintCollection returns true only if all constraints within this set return true. As soon as one constraint is not met, the whole collection will return false. The norm manager creates the constraints and corresponding actions, as, for instance, listed in Listing 6.3. The OverloadWL and the trust breakdown norm both need a

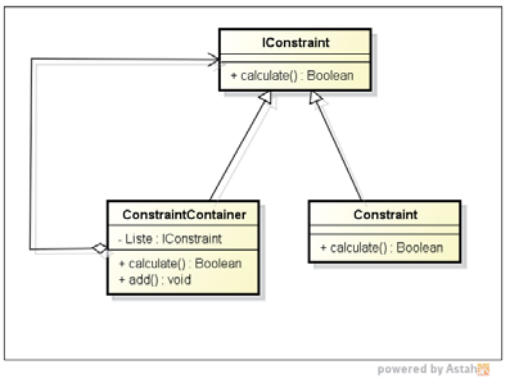


Figure 6.14: Composite pattern

creation and a deactivation function available to the manager.

Listing 6.3: Creation of constraints

```
1
2  ConstraintCollection createOverloadWLNormCollection =
3  new ConstraintCollection();
4
5  createOverloadWLNormCollection.add
6  (new ValueConstraint(avgWL,2,(float) 400));
7  createOverloadWLNormCollection.add
8  (new BooleanConstraint(overloadWLActive, false));
9
10 ConstraintCollection deactivateOverloadWLNormCollection =
11 new ConstraintCollection();
12 deactivateOverloadWLNormCollection.add
13 (new ValueConstraint(avgWL,1,(float) 400));
14 deactivateOverloadWLNormCollection.add
15 (new BooleanConstraint(overloadWLActive, true));
16
17 ConstraintCollection createTrustbreakdownNormCollection =
18 new ConstraintCollection();
```

```
19 createTrustbreakdownNormCollection.add
20 (new ValueConstraint(avgRep,1,(float) 0.1));
21 createTrustbreakdownNormCollection.add
22 (new BooleanConstraint(trustberakedownNormActive, false));
23
24 ConstraintCollection deactivateTrustbreakdownNormCollection =
25 new ConstraintCollection();
26 deactivateTrustbreakdownNormCollection.add
27 (new ValueConstraint(avgRep,2,(float) 0.1));
28 deactivateTrustbreakdownNormCollection.add
29 (new BooleanConstraint(trustberakedownNormActive, true));
```

According to the values used in this example, an OverloadWL norm is activated if the norm manager discovers that the average workload is larger than 400 and the norm is not yet active. Similarly, the TrustBreakdown norm is activated if the average reputation of the agents is below 0.1 and the norm is not active yet. As soon as a norm is activated, the agents are informed via broadcast. Analogously, a norm deactivation occurs as soon as the system returns to the desired states and the constraints are met again. All agents are then informed that they no longer have to follow the norm.

Agent Norm Consideration

System designers have to 'inform' the agents beforehandabout which norms could occur in the system. Agents must know the language in which the norm manager 'speaks' to them in order to be able to interpret the norms. The norm-compliant behaviour modification is then carried out as follows. If no norm is active, the norm-aware agent simply acts in a trust-adaptive manner. As soon as a norm is active, the agent changes its own view of the system. If the OverloadWL norm is active, then the agent does not change the designer-given thresholds, but rather pretends that its workload is lower than it actually is. This gives the designer the opportunity not to change the entire decision-making behaviour but only the state of the agent within this process (trust threshold plane). It also simplifies the combination with the trust breakdown norm: instead of changing the trust threshold plane in two dimensions (workload, trust) at the same time, the agents merely change their view and, thus, their position within the plane. This makes the combination of both norms very elegant. Conflicts do not occur in this case because a combination of changes in two dimensions is possible if we do not exchange the function in both dimensions but only the position within the function.

Of course, this elegant combination of norms regarding different parameters in one function is not possible in most situations. We restrict the abilities of norm-aware agents to the combination of norms that are generally not contradictory. This is easily applicable in the TDG scenario. If we have further norms that might lead to conflicting solutions, the agents' reasoning must be extended in such way that the agents are able to decide which norm is more important. Additionally, agents would need a mechanism to predict the outcome of norm violation. The system would also need mechanisms to detect and punish norm violations — e.g. using the already existing reputation system.

6.9.3 Adaptive Observation Model Agent

So far, we have assumed that each agent type is associated with a fixed scope of (social) awareness. In the more general case, agents need different information in different situations. If the agent can control the observations at run-time, we will spend only the minimum overhead that is absolutely necessary to make the currently vital decisions.

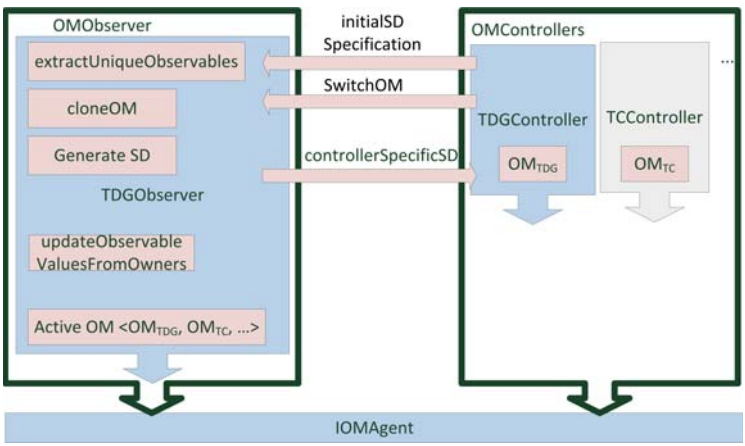


Figure 6.15: Adaptive observation model

We differentiate between calculation overhead, storage overhead and communication overhead. Calculation overhead, for instance, is necessary for aggregations (e.g. trust) or predictions (e.g. future trust). Storage overhead is necessary because

former values are the foundation for many calculations. The trust calculation as well as prediction methods need a certain amount of data to be stored as the basis of their calculation. Both types of overhead are not the crucial problem in PCs today. The communication overhead is a problem for current systems. Exchanging information about reputation or workload among agents results in management messages that decrease the bandwidth of the communication channel for the actual payload (the work units and their results).

A solution of the threat of too much management data with regard to payload is to enable agents to adapt the parameters they observe as well as their update intervals at run-time.

Therefore, we have extended the agent architecture with an adaptive observation model (AOM). The agent controller can specify exactly which information should be collected at any given moment and how this information is to be processed. This corresponds to the tunnel view effect experienced in life-threatening situations where our sensory equipment (as well as the motor equipment) is tuned on escape. Figure 6.15 shows how the adaptive observation model is used. Each agent implementing the interface IOMAgent has at least one controller and one observer. Each controller chooses the set of parameters (observables) that should be determined in order to enable its decision mechanism to function at the beginning phase (initialSDSpecification). An example of the data structure defining the initialSDSpecification is given in Figure 6.16. The observer collects the different initialSDSpecifications from all controllers and determines the unique observables (extractUniqueObservables) in order to prevent a parameter being asked for more than once. The observer then asks all agents assigned to an observable for the values. The raw observable values are then aggregated according to the requirements of the initialSDSpecification of each controller. Aggregation is a function applied to the set of raw data—e.g. average, minimum or maximum. If necessary, the aggregated observables can also be classified using an application-specific classificator class—e.g. to determine if a workload is low, medium or high according to designer-defined thresholds. As soon as the situation changes, a controller might want to change its awareness according to the requirements of the new situation. This is done using the switchOM method. As soon as an OM switch is triggered, the observer creates a new activeOM that fulfils the new controller requirements. In order to save calculation and communication time, values from the old OM that are still observed in the new OM are stored and used for further evaluation (cloneOM).

Which data should be observed and how often these should be updated must be

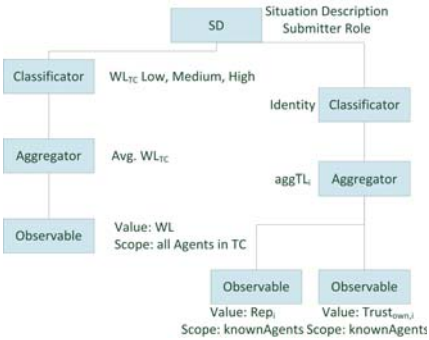


Figure 6.16: Observation model submitter

determined by the controller. For instance, in a situation where all agents behave well, trust information does not have to be updated as often as in a situation involving unknown or potentially malicious agents.

Moreover, the observer is able to merge data requests from different controllers for different roles. Potentially, these data can be identical or the data requests from one role may be a subset of the data requested from the other role. For instance, the submitter role is interested in the aggregated trust value of agents. Similarly, the worker role needs to know the aggregated trust value of requesting agents. It is possible that the information about requesting agents is already contained in the data requested in the submitter role.

Therefore, the AOM merges the initialSDs of all registered controllers, requests information for this data superset, and later dissolves this superset into the format requested by the controllers.

In Figure 6.17, the different observables of the different agent types presented in this thesis are listed. These observables apply to the TDG application scenario 5.2. Self-knowledge is knowledge the agent has about itself, whereas private knowledge adheres to information about other agents (like trust values) that the agent has gathered on its own. Community knowledge is information the agent has collected about other agents with the help of other agents or the reputation mechanism. It is the most expensive class of information with respect to communication costs.

The AOM agent is able to adapt the observables in the current situation. Therefore, these are displayed in brackets. Based on the selection of the observables, the

Observables of agent types in the TDG

General observables		Trust-aware agents	Trust-adaptive agents		Trust-strategic agents		
	Trust-neglecting agent	Fixed stereotype agent	iTC agent	Evolutionary agent	Tactical agent	Norm-aware agent	Adaptive OM agent
Self Knowledge							
WL_{Own}	x	x	x	x	x	x	(x)
Rep_{Own}			x	x	x	x	(x)
Fitness	x		x	x	x	x	(x)
Speedup/Flow $Time_{Own}$						x	(x)
Communication $overhead_{Own}$						x	(x)
Private Knowledge							
$Trust_{Agents}$*		x	x	x	x	x	(x)
Community Knowledge							
WL_{TC}	x	x	x	x	x	x	(x)
Rep_{Agents}		x	x	x	x	x	(x)
PL_{Agents}	x	x	x	x	x	x	(x)
$Norms_{TC}$	x	x	x	x	x	x	(x)

* Trust includes credibility and availability trust.

Figure 6.17: Observables in the TDG

AOM agent can take each behaviour of the other agent types, but also change the observables and, accordingly, its own behaviour at run-time as will be presented in Section 7.10.

6.10 Summary

In this chapter, the adaptive agent architecture has been presented. It decomposes the decision complexity of trust-strategic agents into short-term and long-term decision-making levels. The number of levels depends on the required complexity of the decision. The more complex the controller decision is, the more information has to be gathered by the Observer component.

For each class of levels (trust-neglecting, trust-aware, trust-adaptive, trust-strategic), we presented different techniques pertaining to how the controller side of the agent can be implemented in order to use the available information in the best possible way.

Trust-neglecting and trust-aware agents have been introduced as reference implementation and as disturbances within the system (egoistic agents, free-riders).

Trust-adaptive agents were able to identify and isolate these disturbing agents (iTC agents), thereby ensuring a good performance and enhancing the robustness of the system. We also presented two possibilities on how learning can be applied

to trust-adaptive behaviour. The evolutionary agent learns and optimises strategies from scratch and, thus, finds new solutions that might not always strike the system designer. The learning agent, on the other hand, uses the behaviour encoded in the already successful iTC agents as a starting point and finds new behaviour solutions in that area.

The class of trust-strategic agents is able to include long-term information in the decision-making process. Tactical agents are able to identify trends in situations, make predictions on how these will develop in the future, and adapt their behaviour proactively.

Norm-aware agents are able to consider institutional norms in their decision. Therefore, they can overcome situations that cannot be resolved solely using local knowledge.

AOM agents combine the different features and advantages of the agent types. They are able to adapt their observation model to the current situation and, therefore, also select the controller behaviour that is best suited for the situation from the set of all agent type behaviours. Moreover, by evaluating only that information which is relevant in the agent‘s current situation, AOM agents save communication overheads, thereby leading to a better usage of bandwidth.

Chapter 7

Evaluation

7.1 Evaluation Environment: Trusted Desktop Grid (TDG)

Our application scenario TDG (Section 1.2) is an instance of an open system as well as of common pool resource problems. We want to use this scenario to show situations in which trust and adaptivity can enhance the quality of solutions found by local agents participating in the system.

Trust is useful in situations where misbehaviour is possible. If all agents are cooperative and there are no agent-based attacks to be expected, trust causes more overheads than benefits. Therefore, we want to evaluate situations where trust or adaptivity plays a role:

- Misbehaving agents try to exploit the system.
- Attacks try to outsmart the trust mechanism.
- Global effects occur, which cannot be countered with the only local agent view.

We will first discuss the different local attacks on system performance as well as on the trust and reputation system, before debating about which situations need to be analysed on the system level.

7.1.1 Disturbances Caused by Misbehaving Agents

The commonest disturbance in peer-to-peer systems like the TDG is free-riding. According to the analysis of Hughes [112], about 66% of Gnutella network are free-riders, which consume common goods (files) without giving back to the community. Free riders are part of the system and ask other agents to compute work units for them, but refuse to work for other agents. Here, the trust-adaptive control loops are a perfect candidate measure to counter this threat. If an agent rejects a work unit, this results in a bad rating and, over time, these bad ratings result in a negative trust value. This trust value is used in the agents' cooperation decision. Agents accept work units only from agents with a good-enough trust value (Section 5.1). Therefore, the iTC (Section 5.2.2) evolves. We will show later in this chapter that the exclusion of free-riders by trust-adaptive agents is successful.

Another type of misbehaviour involves the egoistic agent. This agent works for others, but has a high probability of sending no result, a wrong result or an incomplete result. The verification of the correctness of a work unit must be performed by the application, a replication mechanism or the user. If the job can be validated, this checking can be carried out by the application. If the application does not produce jobs that can be validated, the decision of whether a calculation is correct must either be done by the user (which might be a lot of work) or the client can replicate the work units several times, distribute them and then determine the correct result using majority checking or similar approaches.

An overuse of replication, on the other hand, can also be seen as a misbehaviour in the submitter role. The trust-adaptive mechanisms presented in this thesis aim at minimising replication overheads by ensuring that only trustworthy agents are asked for their cooperation.

Tactical behaviour like building up trust, exploiting trust, building it up again and so forth can also be regarded as agent misbehaviour and is punished by some existing trust and reputation systems (Section 3.8). In the TDG, we use a positive version of this behaviour in order to make agents successfully trust-adaptive. As long as agents remain within positive trust parameters, we regard this behaviour as non-critical.

Other types of misbehaviour, like attacks on the reputation system or its stored values, attacks on messaging or authorisation, and authentication attacks are a matter of security and, therefore, not regarded in this thesis in particular.

7.1.2 Disturbances Analysable on System Level

Collusion attacks are attacks realised by a set of agents that arrange a common behaviour in order to harm or, at least, exploit the overall system. For instance, agents could compromise the reputation database by giving each other positive ratings without any real cooperation taking place. Thus, researchers at Cornell University discovered a theoretical collusion attack on the Bitcoin mining community [113].

Apart from different collusion attacks, there are also system states that are not easy to observe locally. For instance, an overload situation, where there is more workload in the system than agents can handle, is hard to obtain as a single agent. The agent itself only knows its own workload and may be able to derive from the answers of agents it asks for cooperation that the workload of other agents might be high as well. However, a system-wide overload situation must be observed on the system level rather than by each local agent trying to obtain a system-wide view of the situation.

Similarly, a trust breakdown situation [114], where the agent's mutual trust relations are broken, can hardly be observed locally. The trust breakdown can be due to agent decisions or external effects like an ongoing overload situation (where agents can no longer receive positive ratings) or message losses (which causes bad ratings because requests or answers are not received).

Overload as well as trust breakdown situations are severe problems that can result in agents deciding to leave the system and, therefore, destroy the system over time. As local observation alone is insufficient for such situations, we need a hierarchical component (institution) that is able to observe the system, detect these critical situations, and legislate norms telling the agents how to behave as long as this abnormal situation prevails. One example is the TCM observing on system level and being able to legislate norms to lead the system out of critical or undesired situations.

7.2 Evaluation Metrics

In this section, the evaluation metrics used in this thesis are presented. On the system level, we use metrics from the desktop grid computing area because these measures are used to determine successful strategies on a global scale. On the agent level, we also analyse metrics that are used to determine the agent behaviour. These metrics can lead to global success but are also interesting tools for the analysis of local behaviour decisions.

7.2.1 Evaluation Metrics on System Level

The evaluation metrics that we use to measure the performance of the adaptive agents on the system level are derived from performance metrics in grid computing literature. Additionally, there are some indicator metrics that we use for further analysis of the effects on a global scale.

Waiting Time

Jiang [27] describes the *average waiting time* as „*the average waiting time spent by a job in the grid.*“[27] Therefore, the average waiting time is a measure for the efficiency of work unit distribution. A bad load balancing, for instance, can result in long waiting times because the workload is concentrated on too few agents. Nonetheless, the average waiting time can also be high if the system is in an overload situation. The *total waiting time* (as the sum of all waiting times) can bring more clarity to the analysis because averaging can decrease the informational power of the single values. Therefore, we use the total waiting time to analyse system situations, but combine it with the flow time and speedup in order to determine comparable situations and settings for analysis.

Total Flow Time

In this thesis, we use the definition of Lee [115], which defines flow time (Eq. 7.1) and total flow time (Eq. 7.2) as follows:

$$F_i = C_i - r_i \tag{7.1}$$

The flow time F_i is the difference between the end time and the release time of a job J_i. The sum of the flow times of all jobs is then the total flow time (Eq. 7.2).

$$\sum F_i \tag{7.2}$$

The total flow time is a measure to analyse the degree of parallelism of job execution.

The flow time ratio is the total flow time normalised by the overall time which has passed. The lower the flow time ratio is, the less time an agent had to wait for the completion of its jobs.

Similar to the flow time, the turnaround time of a job is defined: It is the calculation time of a job from the submitter's perspective, starting with sending the

first work unit, including waiting times, until the last work unit is calculated and returned.

Speedup

The speedup used in this thesis follows the general definition of speedup from the area of parallel computing. It defines how much faster the algorithm can be executed in a parallel system than on a serial system. In this thesis, the *average speedup* describes how much faster each job j can be executed in parallel by other agents than serialised on the owner's machine (see Eq. 7.3).

The speedup can be defined either as a global value observed during the complete simulation time or as the average speedup which is evaluated after each job completion. Therefore, we measure both values.

$$S(x,y) = \frac{processingtime_{x,j}}{processingtime_{y,j} + waste} = \frac{\frac{W_j}{P_x}}{\frac{W_j}{P_y} + \sum\limits_{s} waste_s} \tag{7.3}$$

In the speedup according to Eq. 7.3, the size W_j of a work unit and the worker performance P_y are considered and contrasted with the *a posteriori* measure of the submitter performance P_x and the summed waste in all time steps s.

Scheduling Success Rate

There exist two definitions in literature for the scheduling success rate:

„*Scheduling success rate: the percentage of jobs successfully completed in the grid*“[27]

and

„*Success rate: The success rate for a run is defined as the ratio of the number of successful work unit completions (without rescheduling) to the total number of work units allocated.*“[22]

The general statement is similar. It is defined as the proportion of successfully completed work units of all work units. As the second definition is more detailed and does not controvert the first one, we use the second definition for our purposes. Moreover, the second definition enables us to use the scheduling success rate as a quality criterion for successful matchmaking.

Waste Ratio

Kondo [116] defines waste as follows:

„*Waste: of the CPU time that is used, the fraction used by jobs that miss their deadline.*“[116]

We transfer this definition into our simulation environment. Therefore, we also include all the times that an agent has computed a work unit, which has been cancelled later, as waste. Therefore, waste is the sum of all time ticks that haven been spent in useless calculations. We normalise this waste value by the overall time spent for computations in order to compute the *waste ratio*. This normalised value is better for comparison because the problem size is directly taken into account. This value represents the amount of unproductive computing steps in the whole system and is, therefore, a measure for the effectiveness of cooperation partner selection. The higher the number of misbehaving agents chosen for computation, the higher is the waste ratio.

7.2.2 Evaluation Metrics on the Agent Level

On the agent level, we can evaluate the system metrics for each agent, but are also interested in metrics to analyse the behavioural performance of an agent. Therefore, we added a fitness function, considering the benefit and the effort taken to reach this benefit. We also analyse the observation overhead by measuring the communication effort of the agents.

Fitness

The *fitness* of an agent is used to measure its success independently from the scenario. The fitness is used in two ways. On the one hand, the agent designer has a measure for the agent success, which also takes the effort used into account. For cooperation-based agent strategies, this is extraordinarily important and goes beyond the system metrics, which could also be used at agent level. On the other hand, agents like the evolutionary agent (Section 6.8.2) can also use this metric themselves to measure their success on-line, thereby optimising their behaviour accordingly. This is the third loop in the trust-based interaction mechanics presented in Section 5.1.

The fitness is calculated as a weighted sum of *benefit* and *1 - effort* (Eq. 7.4), where the term *1 - effort* is used to take into account the effort which has been spend in order to reach a certain benefit and also to enable users to define the minimisation of the effort as a goal for the agent.

$$fitness = \alpha * benefit + (1 - \alpha) * (1 - effort) \tag{7.4}$$

The fitness of an agent is evaluated each time one of its jobs has been completed in the TDG. The weight α is defined between 0 and 1, and can be determined by the user or the system designer. This weight defines whether benefit (cooperative behaviour) or the minimisation of effort (egoistic behaviour) is preferred.

The fitness is a normalised function in an interval between 0 and 1. The normalised benefit, which an agent receives from the calculation of its last job, is defined as:

$$benefit = \begin{cases} 0, & \text{if actualPTime} > \text{ownerPTime} \\ 1 - \frac{actualPTime - potentialPTime}{ownerPTime - potentialPTime}, & \text{else.} \end{cases} \tag{7.5}$$

The benefit is the proportion of time that an agent has saved by distributing its job in the system instead of calculating it on its own. The actual calculation time needed for the job, $actualPTime$ (including waiting times), is calculated as a ratio of the time it would have taken the agent itself to compute the work unit ($ownerPTime$). To normalise this ratio, the $potentialPTime$ is also taken into account. This is the time it would have taken to compute the job in a perfect system where all work units of the job can be computed in parallel by the fastest agents—i.e. the perfect parallelisation. Therefore, $ownerPTime > potentialPTime$ always holds. This normalisation transfers the benefit into an interval between 0 and 1.

In order to reach this benefit, the agent has to make some effort ($0 \leq effort \leq 1$). Due to the coupled trust and reputation loops (Section 5.1), this effort in the worker role can be necessary to reach a reputation that is high enough to receive a good benefit in the submitter role by finding suitable cooperation partners that rely on trust values as the basis for cooperation decisions.

$$effort = \frac{timeActiveAsWorker}{timeSinceLastJob} \tag{7.6}$$

The effort is computed from the time steps that the agent has been actively working for other agents ($timeActiveAsWorker$) normalised by the time which has passed overall since the last job ($timeSinceLastJob$). This fraction is already in an interval between 0 and 1. The effort is 0 if the agent did not work for others since the last job has been completed; it is 1 if it has been actively working for other agents throughout the time interval since the last job. By using the term *1 - effort* in the fitness definition, we make sure that a maximisation of fitness can be reached not only by maximising the benefit (which is not always possible for an agent), but also by minimising the effort an agent has to participate in the system (e.g. being more egoistic in certain situations).

Observation Overhead

The observation overhead is measured as the average number of messages per agent per tick. The less the overhead spent for observation, the more the payload that can be sent and received in the network. The TDG is a system involving communication over the internet. As the bandwidth here is limited, a minimisation of the observation overhead can result in a better usage of the available bandwidth. The observation overhead is a metric which we used to analyse how agents use their available bandwidth.

7.3 Comparison of TDG with State of the Art

In this section, we will compare our system to the state of the art. Our analysis of the state of the art (Chapter 3) has shown that the most promising candidates for comparison are H-trust and the Organic Grid. Therefore, we implemented both approaches and compared the efficiency of the agents and the robustness of the system regarding how misbehaving agents are evaluated. The changes to the simulation in order to create a fair evaluation environment for all approaches are presented in the following.

7.3.1 Organic Grid

This section presents our implementation of the Organic Grid according to [6]. As the H-Trust agent, the Organic Grid agent extends our basic agent class *ComputerAgent*. The Organic Grid uses pull mode, which means that agents advertise their free computing resources as soon as they are available rather than waiting for others to ask them for computation. Therefore, the *step* method of the ComputerAgent is overwritten in the *OrganicGridComputerAgent* in such a way that the agent uses the pull mode instead of the push mode.

The general behaviour of the step method *OrganicGridComputerAgent* can be seen in Figure 7.1.

In each time step, the step method of each agent is called and executes several methods, of which, we focus below only on those that add an extra functionality to the system:

updateAncestors: Each node needs to know which nodes are its parents; therefore, it continuously asks them if this is still the case. This is done recursively and the resulting tree overlay information is stored in the local ancestor list of each agent.

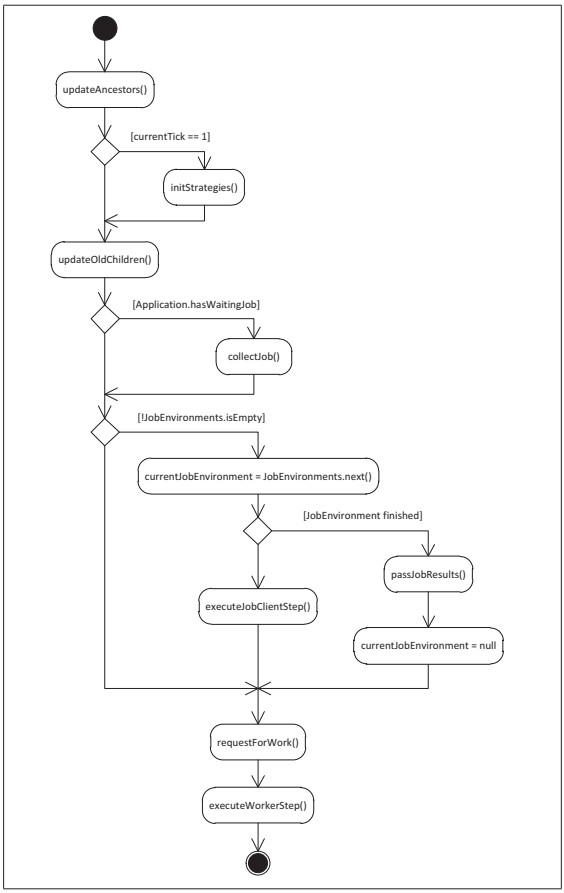


Figure 7.1: *Step*-method of the *OrganicGridComputerAgent*

Agents eliminate circles by searching for their own ID in the list and, if they find it, informing their parents that they are not child nodes any more.

updateOldChildren is used to remove nodes that have become too slow or that do not answer any more. These child nodes are removed from the tree overlay and stored in a list. After a predefined interval, agents from that list can be tested for cooperation and performance once again, which makes the Organic Grid a forgiving system.

requestForWork is an abstract method in our framework, but it is implemented for the *OrganicGrid* agent as follows. Agent B informs its parent node agent A if it is idle and able to compute work units for others. Agent A gives agent B work units to compute as long as the sum of work unit computation ticks does not exceed the $initialworkunitrequestsize$, which is dynamically adapted to agent B's performance at run-time. Due to the dynamic recursive structure of the overlay, agent B will always try to find new parent nodes if it has more computational power than required by its current parent node.

After having presented the key components of the Organic Grid agents, we will show how the two types of misbehaving agents are included in the Organic Grid system.

Egoist

In the *requestForWork* method, we needed to develop an adaptation to implement egoistic behaviour. With a $probabilityforegoistWUabort$, there is a communication error generated (which could also relate to misbehaviour on purpose), and the work units are aborted during computation and are not completed. The information about the aborted work unit is then forwarded to the submitting agent through the network.

Free-rider

The free-riding misbehaviour had to be transferred into the Organic Grid mechanics. In the submitter role, free-riders behave like all OrganicGrid agents: they distribute work units in the network. In the worker role, the *requestForWork* had to be overloaded with an empty method, which means that free-riders never request work from other agents. This is the translation of the free-rider behaviour 'never accepting a work unit' from push mode to pull mode ('never requesting work').

7.3.2 Comparison of Organic Grid with TDG

Undisturbed System State

We executed the same set of experiments with the TDG and the Organic Grid system using the parameters listed in Table A.4 for the Organic Grid and and Table A.2 for the TDG-specific parameter settings. The Organic Grid with its tree-based overlay structure reached a system state with a higher total flow time and a higher total waiting time (see Fig. 7.2).

The TDG has advantages over the Organic Grid in low- and medium-load situations like the one used in Figure 7.2. Under a high workload, however, the Organic Grid leads to a better performance in terms of total flow time and total waiting time. This can be explained by the push strategy, which leads to an efficient load balancing as agents will only ask to work if they are available. In particular, situations involving parallel distributions cannot occur. In the Organic Grid, an available agent asks only its parent for work, whereas in the TDG, the best-ranked agent according to trust and workload can be asked by several submitters in parallel for computation in the same time step. As the Organic Grid concentrates on adaptivity and does not use trust information, it is very promising that the TDG approach (with its overhead of trust mechanisms) led to similar results as the Organic Grid in a system without disturbances.

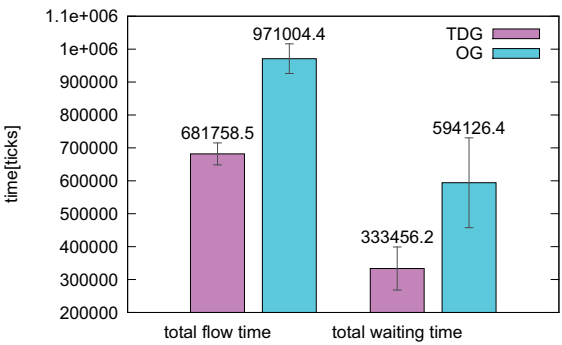


Figure 7.2: Average total flow time and waiting time of TDG and Organic Grid (OG) in undisturbed system state

Disturbed System State

Figure 7.3 shows the total flow time and the total waiting time of the TDG and the Organic Grid approach with a mean workload of 3,000 ticks (i.e. medium in this scenario), and a disturbance caused by 15 egoists and 15 free-riders to 70 agents (see A.5 for the other parameters used in this disturbance setting). This means that, in this disturbance and workload situation, the TDG has a lower flow time and waiting time; thus, it is able to compute work units in a faster and more effective way.

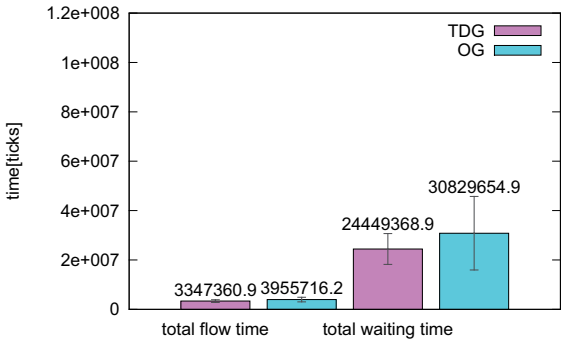


Figure 7.3: Average total flow time and waiting time of TDG and Organic Grid (OG) in disturbed system state

Moreover, the TDG was able to prevent the production of waste (see Figure 7.4 by making agents concentrate on trustworthy cooperation partners. The scheduling success rate of the Organic Grid shows that the agents find workers in the first place, but the higher waste shows that these workers are untrustworthy (e.g. egoists) and not suited as cooperation partners. Once again, this is explained by the self-organising mechanisms of the iTCs in the TDG, whereas the Organic Grid tries to rely on the untrustworthy agents as well.

Summary: Organic Grid vs. TDG

In overload situations, the Organic Grid was better able to perform well due to the distribution restrictions within the scheduling mechanism [117]. In situations involving a high workload, the Organic Grid reaches a better total flow time than does the TDG. This is, as in the undisturbed situation involving a high workload,

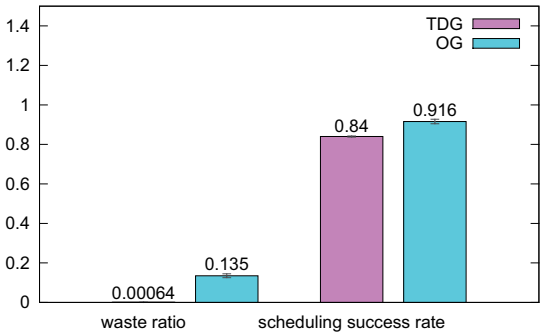


Figure 7.4: Average waste ratio and scheduling success rate of TDG and Organic Grid (OG) in disturbed system state

due to the efficient push mechanism in the Organic Grid. In situations with low or medium workload, the TDG was comparable or even superior to the Organic Grid. In a disturbed system state, the TDG reached better total flow times, total waiting times and a minimisation of waste compared to Organic Grid. This is due to the self-organised iTCs that build up in the TDG and, thus, effectively minimise the total waiting time. In the TDG, free-riders and egoists are recognised and isolated over time, resulting in a minimisation of flow times, waiting times and waste.

7.3.3 H-Trust

This section presents the implementation of H-Trust, which we have conducted for the sake of comparison with our mechanisms. The H-Trust system consists of five phases. The first one is the trust recording phase [25], in which the former interaction results of workers are stored. The local trust evaluation phase is used to compute the local trust value of an agent. Since an agent in a desktop grid cannot have complete trust information about all agents, in the trust query phase, an H-trust based trust value is computed based on trust ratings and credibility information. To keep the trust and credibility tables updated, the tables are periodically renewed in the spatial-temporal update phase. Finally, in the group reputation evaluation phase, the reputation of agent groups is computed based on thresholds and the H-trust function.

The implementation of H-Trust is based on the publications of the inventing author in [25] and [57]. Nonetheless, we had to make some assumptions due to lacks of information in these publications. These assumptions are presented in the following.

Each H-trust agent owns three tables called the local service history table, the local trust rating table and the local credibility rating table [25].

The service history table is updated with a new entry for each interaction that the agent has with another agent. This entry includes the agent (remote Peer ID), the time step at which the interaction has ended, the service quality that has been provided, and the importance of the interaction (service importance). We transferred this into our TDG simulation environment by storing an entry for each work unit calculation, rejection and cancellation with a constant service importance of 3 (as the authors did not define how these values are initiated, and as all services are equal in our simulation). A work unit rejection results in a service quality of 1, and a finished work unit leads to a service quality of 5.

The trust values of known agents are stored in the *local service history*. According to Zhao [25], these trust values are computed using the vector-based approach presented by Selcuk [65]. A positive experience is rated with 1, a negative one with 0. The number of experiences used is based on the quantity of significant bits and then used to compute the trust value according to Eq. 7.7 (for example, vector 11101000 with five experiences).

$$Trustrating = \frac{(11101000)_2}{2^5} = 0.90625 \tag{7.7}$$

The trust vector is shifted once to the left with each new experience. In the example above, four out of five experiences are positive, and the first five digits are taken into account, which makes $11101 = 29$.

An agent stores such a vector for each agent that it has used as a worker and uses this local trust rating information to decide with whom to cooperate if it is asked for cooperation (worker role).

In the submitter role, the agent uses the trust rating table. The agent uses a quantity of agents defined by the *queryPercentage* parameter. These agents are filtered in such a way that only agents with a trust rating greater than the *selectionThreshold* are selected and asked for cooperation. This behaviour is derived directly from the Netlogo implementation given in [118] and [25]. As can be derived from the details of this implementation, the H-Trust approach is designed to deal with incomplete knowledge about other agents and their behaviour.

In our implementation of the H-Trust approach, we used all the simulation parameters presented in [57] (Table 5). These include initial trust and credibility values, expectation values and standard deviation of the Gaussian distribution.

Having presented the details of our H-Trust agent implementation, we will provide an overview of the implementation details of the disturbing agents, the egoistic agent and the free-rider. These had to be adapted to the functionality of H-Trust agents in order to enable interactions among all agent types.

Egoist

Egoistic behaviour in an H-Trust MAS is implemented in the *HTrustEgoisticAgent*. If it is asked by a submitter, the HTrustEgoisticAgent in the worker role decides, based on its *selectionThreshold* and the submitter's trust value, whether or not to accept the work unit. If the work unit is accepted, the HTrustEgoisticAgent will cancel the work unit with a *probabilityforegoistWUabort*. This represents exactly the egoistic behaviour that we model as a disturbance within the TDG in general.

In the following section, our implementation of free-rider behaviour in an H-Trust-based system is presented.

Free-rider

The disturbance caused by free-riders is implemented in the *HTrustFreeRidingAgent*. Free riders do not accept work units. Thus, they model system members that consume computational power without providing any in return and exploit the system in this way. In this implementation, the *HTrustFreeRidingAgent* always declines when asked to be active as a worker. In the submitter role, however, the agent acts like any H-Trust agent and uses the same distribution strategy.

7.3.4 Comparison of H-Trust with TDG

The trust-based mechanics of H-Trust are a promising candidate for comparison with the TDG. Therefore, we compare the two systems in different situations.

Here, we concentrate on the metrics that help us to explain the main performance differences and effects; a more detailed analysis is presented in [117].

Undisturbed System State

In a system situation that has no disturbing agents, we expect high performance in both systems. The trust mechanism here is an overhead because all agents behave

in a trustworthy manner. Here, we compare the load-balancing mechanism of both trust-based systems. In order to compare both approaches, we used the parameters defined in A.3.1 for both simulations, and A.3.2 for the TDG-specific parameter settings.

Figure 7.5 shows the average total flow time and the total waiting time of the agents in a scenario with the above-mentioned settings in a long-term experiment conducted with the TDG and H-Trust.

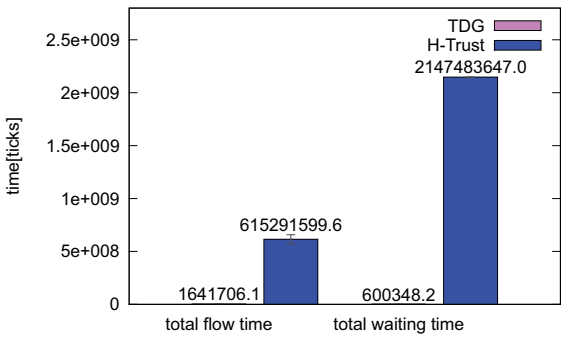


Figure 7.5: Average total flow time and total waiting time of TDG and H-Trust in undisturbed system state

The variance between the different experiments, especially regarding the total waiting time, is caused by the parallelism in some of the randomly initialised workload situations: several agents had similar jobs in similar time intervals, but all of the jobs could not be distributed at the first attempt. Therefore, these jobs were postponed for a time interval and then distributed. This delay is a part of the waiting time; therefore, in some experiments, it leads to larger waiting times than the average. Nonetheless, these values can be used for comparison with the H-Trust.

It is clear that H-trust leads to a lower performance, as agents spend much more time waiting in queues than they did in the TDG. This is due to the distribution mechanism, which is based on trust and random selection, but does not take the performance of the agents into account. In contrast, TDG agents adapt to both, trust and workload; therefore, it is more successful than H-Trust in finding fast workers.

Disturbed System State

Figure 7.6 shows the total flow time and total waiting time of the H-Trust approach in the same disturbance setting as used above. It is obvious that H-Trust agents in a medium workload situation reach a massively higher total waiting time and an extremely high flow time. This shows that, despite the benefits of trust, H-Trust is inferior to both, the Organic Grid and the TDG, due to the lack of load balancing in the scheduling approach.

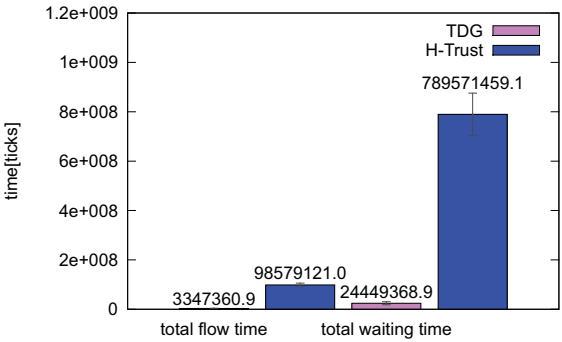


Figure 7.6: Average total flow time and waiting time of TDG and H-Trust in disturbed system state

Figure 7.7 shows the waste ratio and scheduling success rate in the same scenario. It is interesting to see that, despite the slightly better waste ratio, H-Trust still reaches a worse scheduling success rate than does the Organic Grid. This can be explained by a successfully working trust mechanism (resulting in less waste production), which is unfortunately overcome by the still insufficient load balancing in H-trust: agents know which agents are trustworthy workers, but still cannot make good cooperation decisions.

The TDG again reaches a better waste ratio and scheduling success rate than do the H-Trust and the Organic Grid.

Summary: H-Trust vs. TDG

Due to the better load balancing, the TDG clearly outperforms H-Trust in undisturbed system states as well as under disturbance. This shows that agents reach a

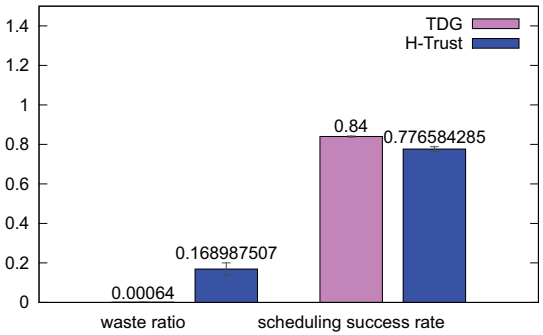


Figure 7.7: Waste ratio and scheduling success rate of TDG and H-Trust in disturbed system state

better performance if they do not use trust as the only cooperation criterion, but take a combination of trust and workload as well adaptivity into account.

7.3.5 Summary: Comparison with Related Work

The general idea of the agent approaches in this thesis is to combine trust and adaptivity in order to increase their performance in open MAS like open desktop grids. Therefore, we develop an architecture that allows for the combination of trust, adaptivity, learning, and even a long-term strategy component.

In this section, we compared trust-adaptive iTC agents that adapt their behaviour to workload and trust situations, to the state of the art in trust-based or adaptive grid systems—H-Trust and Organic Grid. The Organic Grid had advantages in systems involving a high or even overloaded workload situations: the push-mechanism ensured that work units are always given to the fastest available agents. This proved successful in situations without disturbances, but the approaches of the trust-adaptive iTC agents were competitive, too. In situations involving disturbances caused by free-riders or egoists, the lack of trust consideration in the Organic Grid was visible and the Organic Grid was outperformed by the TDG agents. Only in overload situations under disturbances, could the Organic Grid succeed due to the stable push mechanism.

H-Trust showed drawbacks regarding the load balancing mechanisms: work units

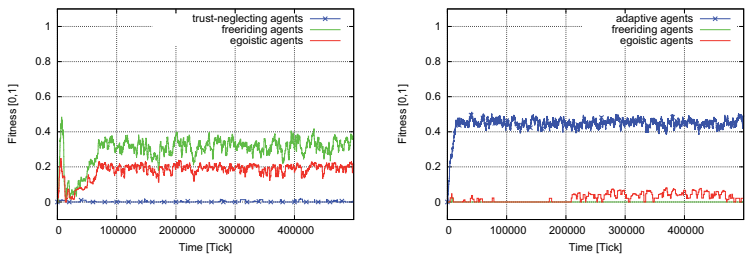
were given to the most trustworthy agents, rather than to the fastest ones. Therefore, despite trust consideration, H-Trust was outperformed by both, Organic Grid and TDG agents.

To sum up, by comparing our approaches to the state of the art, we could show that trust alone does not suffice, but a combination of trust and adaptivity to workload situations as implemented by our agents leads to both, high performance and robustness of the system.

7.4 Trust-awareness Used to Reduce Information Uncertainty

In this experimental setting, we will demonstrate that agents can enhance their performance by using trust as additional information.[1]

In this scenario, 30 agents are confronted with five egoistic agents and five free-riders, which try to exploit the system. Figure 7.8(a) shows an example of the performance of trust-neglecting agents confronted with both types of misbehaving agents. It is obvious that trust-neglecting agents reach a very low fitness. This is because they are exploited by egoistic agents and free-riders, both of which achieve a good fitness.



(a) Performance of trust-neglecting agents

(b) Performance of trust-adaptive iTC agents

Figure 7.8: Performance comparison of trust-neglecting and trust-adaptive agents.

In Figure 7.8(b), we repeated the same experiment using trust-adaptive iTC agents. These only cooperate with agents that are rated as trustworthy. Since

[1] Work on this has been conducted in part in cooperation with Lukas Klejnowski in the context of the DFG research unit OC-Trust (FOR 1085).

egoistic agents and free-riders are rated negative as soon as they reject or cancel a work unit, both types of misbehaving agents have low trust values; therefore, they are not regarded as cooperation partners by trust-adaptive iTC agents.

Egoistic agents and free-riders reach a very low fitness because they are avoided by the trust-adaptive iTC agents. Therefore, we define the set of agents that mutually trust each other as iTC agents (Section 5.2.2).

This can also be seen in the desktop grid metrics that we evaluated as Figure 7.10 in the following section will show.

7.5 Trust-adaptivity to Improve Robustness

In this experimental setting, the job generation pause was 1,500 to 4,500 ticks. [2]

In compliance with Schmeck [119], we have examined the behaviour of the TDG in case of sudden, unexpected disturbances. We focused on the introduction of additional free-riding agents into a system consisting of 30 adaptive agents, five egoists and five free-riders.

[119] defines target space as the set of ideal system states. The acceptance space contains all acceptable system states and, therefore, also includes the target space. The survival space contains all system states that are not acceptable, but from which a reconfiguration is still possible without damage to the system. According to [119], a weakly robust system leaves target space if a disturbance occurs, but remains in acceptance space and will return to the target space after a recovery phase. We adhere to these definitions and define our system as a weakly robust one. Therefore, we define the target space of our system as the system states where the overall fitness in the iTC, which is the bottom-up formed set of all agents being trusted by the other agents, is above a threshold of $T_{Target} = 0.45$. This value is chosen based on the current system configuration and influenced especially by the workload existing in the stable phase without further disturbances. The system is in the target space for as long as $Fitness_{System} >= T_{Target}$. If a disturbance occurs, it decreases the fitness function and, thus, pushes the system into acceptance space. We define the acceptance space of our scenario to be above the threshold between acceptance space and survival space. Here, survival space means that the system fitness is below a threshold determined by system configuration parameters in general and the current workload in particular. Here, we chose $T_{Acceptance} = 0.1$ as the threshold between

[2] Work on this has been conducted in part in cooperation with Lukas Klejnowski in the context of the DFG research unit OC-Trust (FOR 1085).

acceptance space and survival space. As the fitness is an average of all agents, an overall system fitness of 0.1 or less would lead to a state where a number of agents might have a fitness that is nearly 0. A fitness value of 0 means that the agent does not profit from using the grid for its calculation and could have processed its work units on its own. Therefore, we define the acceptance space in such a way that the overall fitness still indicates a positive benefit from the system. Thus, our system is in the acceptance space if $Fitness_{System} >= T_{Acceptance}$. The system is in the survival space if $Fitness_{System} < T_{Acceptance}$. This value indicates that some of the agents might as well leave the system as they do not benefit from it.

In the experimental setup that was used here, the initial agent population consisted of 25 percent uncooperative agents (i.e. egoistic agents and free-riders). In Figure 7.9, the disturbance doubled the number of uncooperative agents, which is a substantial disturbance.

In traditional systems that lack trust-based detection mechanisms for agent misbehaviour, this arrival of free-riders would raise the average workload of the system for the above-mentioned reasons, would lower the average reputation of the agents as they are often forced to reject the processing of additional work units because of their high load, and consequently, the average fitness of the agents would decrease. Although we first see exactly these characteristics in the TDG at the time the disturbance is introduced (cf. tick 35,000 in Figure 7.9), the system is in a self-organising recovery phase just after the occurrence of the disturbance. The system recovers from the mentioned low performance state in the acceptance space to return to the target space as defined in [119]. After the recovery phase, the workload is reduced to the level that prevailed before the disturbance although the additional free-riders remain in the system. This is an indicator that these agents have been isolated by the iTC.

The behaviour of the new agents entering the system is not known. They start with a reputation of 0.1, which makes them potential cooperation partners for the adaptive agents. These free-riders are therefore able to delegate the processing of their work units to the trust-adaptive iTC agents, which shows as a reduction of the workload of the free-riders. Additionally, these work units are returned correctly and the free-riders in the beginning benefit from their participation in the grid. However, in the further progression, the cooperative agents realise that the free-riders do not return the favour of being a worker, which results in negative trust ratings and thus decreasing reputation. The consequence for the adaptive agents is that they do not accept further processing requests from the free-riders. However, they do not cancel

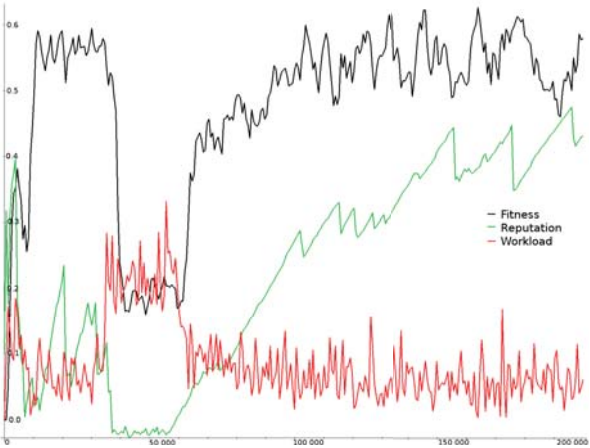


Figure 7.9: Performance of trust-adaptive iTC agents with disturbance of 25 percent free-riders

the processing of already accepted work units, mainly because of the negative ratings that this would imply. Thus, in Figure 7.9, we see a slow reduction in the workload of the adaptive agents.

In Figure 7.10, we present the evaluation of the self-organised iTC formation in the system consisting of 70 trust-adaptive iTC agents and 30 misbehaving agents (15 free-riders and 15 egoistic agents) using metrics from the grid computing domain.

The most important metric here is the average speedup. It is obvious that trust-adaptive iTC agents reach a good speedup (8.73 on average) whereas the mean speedup of misbehaving agents is below 1.0 (0.86), which means that these agents have no benefit from distributing their work units in the grid. This decrease of the performance of misbehaving agents is due to the fact that their misbehaviour is over time recognised by the iTC agents and punished with low reputation. Due to the low reputation, misbehaving agents are less likely to find cooperation partners and thus reach a low performance. This is also visible in the flow time ratio, which, for cooperative agents is about 0.13 whereas misbehaving agents have a flow time ratio of 1.32, again being interpreted as having no benefits from the grid. The scheduling success rate indicates that, due to their low reputation, misbehaving agents do not find agents to compute their work units (scheduling success rate 0.01). In contrast

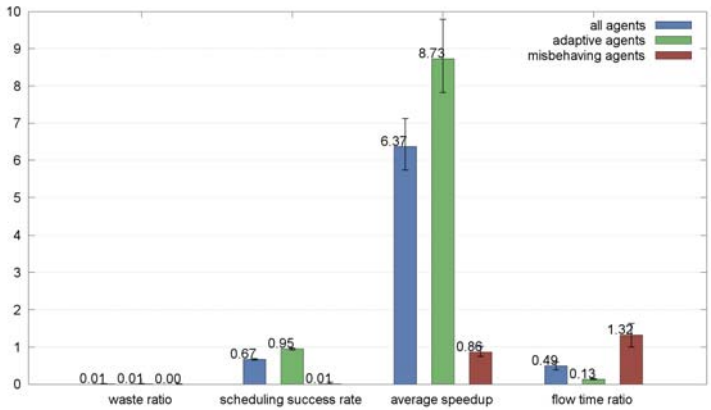


Figure 7.10: Performance comparison of trust-adaptive iTC agents and misbehaving agents

to that, trust-adaptive iTC agents are regarded as trustworthy (i. e. have positive reputation values) and thus reach a high scheduling success rate of 0.95.

Further evaluation results of this experimental settings supporting the analysis are depicted in Figure A.4.

7.6 Evolutionary Approach for Continuous Run-time Adaptation

So far, we have analysed trust-adaptive agents [109]. These agents are able to adapt their behaviour to the current situation that they observe based on predefined thresholds. These thresholds are tailored to the situation—e.g. if there is a high workload, an agent needs to ask more (and, occasionally, even less trustworthy) agents for cooperation. For each situation, the system designer defines a suitable threshold based on his knowledge of the system at design-time. However, we want these agents to learn and optimise at run-time the threshold best suited in a given situation. Thus, optimising agent behaviour at run-time is crucial for a successful adaptation to changing environmental conditions as requested in the open, dynamic systems that we consider. Therefore, we develop evolutionary agents that are able to find new agent

configurations at run-time without prior knowledge of the system and its dynamics. One possibility for optimisation is an evolutionary approach where a new population arises during agent interaction, continuing life with the dominant genes of the successful agents from the last generation. This completely distributed way of learning and optimisation seems to be worth considering for the agents in our application scenario. Therefore, we here evaluate the evolutionary agents, which optimise decision making in both, worker and submitter roles, at run-time through imitation of the fitter agents in combination with mutation. We will investigate which strategies evolve as an emergent phenomenon of the interaction of the evolutionary agents.

In the experiments presented in this section, we investigated how the evolutionary agents behave in interaction with other types of agents and with each other. In Experiment 1 (Sect. 7.6.1), we analysed how evolutionary agents behave in a heterogeneous system featuring both, evolutionary and adaptive agents, without evolutionary learning approaches. Experiment 2 (Sect. 7.6.2) has been conducted in order to evaluate the behaviour of evolutionary agents in a homogeneous system. In Experiment 3 (Sect. 7.6.3), we investigated the behaviour of evolutionary agents when they interact with egoistic ones. This is a misbehaviour we would like the agents to recognise in a distributed fashion and adapt to at run-time.

7.6.1 Evolutionary Agents vs. Adaptive Agents

Figure 7.11 shows the average fitness of the evolutionary agents and the adaptive agents. The average fitness of the evolutionary agents is much higher than the fitness of the adaptive agents. The average reputation of the adaptive agents shown in Fig. 7.12 is higher than that of the evolutionary agents, but the latter still have a good reputation that is greater than 0.5. Thus, a good reputation can help to reach a higher fitness, but a high reputation does not necessarily imply a high fitness. Already after tick 30,000, a dominant chromosome structure has evolved: genes 1, 4, 5, 6, 7, 8 and 9 have the value 1 and genes 2, 3 and 10 the value 0. Figure 7.13 shows that the workload of the evolutionary agents decreases and the workload of the adaptive agents increases, which means that the evolutionary agents successfully distribute the work units to the adaptive agents. In other words, the evolutionary agents are able to exploit the adaptive agents. This behaviour can also be observed in systems involving other agent types. Evolutionary agents are successful regardless of what the other agents in the system might be because their behaviour continuously adapts to the system configuration. The agents with the highest fitness are copied; thus, the most successful strategy for a given situation evolves and spreads over

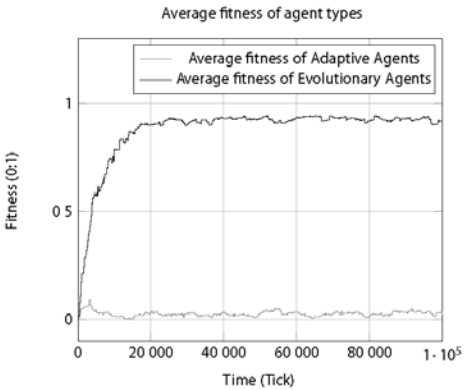


Figure 7.11: Results of experiment 1: Average fitness of evolutionary agents and adaptive agents

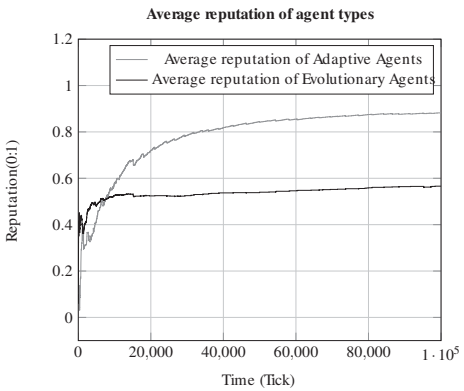


Figure 7.12: Results of experiment 1: Average reputation of evolutionary agents and adaptive agents

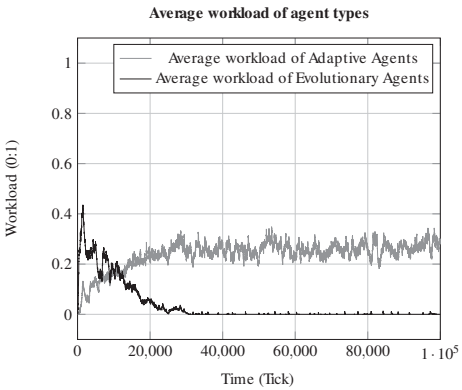


Figure 7.13: Results of experiment 1: Average workload of evolutionary agents and adaptive agents

time. Therefore, in unknown system configurations, an evolutionary approach is worthwhile. This holds as long as there are enough agents using this strategy. Our further experiments have shown that the enforcement of successful chromosomes needs about 25% of the system population to be evolutionary agents in order for the system to be fast enough to adapt to the environment successfully.

7.6.2 Homogeneous System of Evolutionary Agents

It can be seen from Fig. 7.16 that value 1 for gene 1 wins from almost immediately after the start of the simulation. This shows that being cooperative is a more successful strategy than being egoistic. In Fig. 7.14, it is particularly noticeable that at tick 20,000, the fitness strongly drops and rises again at tick 30,000. This matches with the fact that, during the same period, the value 0 of gene 4 in Fig. 7.17 has been established and so the majority of the agents will pay no attention to the fitness of the partner when accepting work units. In Fig. 7.15, it is noticeable that due to the increased workload in this period and due to the lack of trust, the agents cannot distribute their work units any more and are forced to process them on their own.

As soon as the value 1 for gene 4 prevails, the fitness increases and the workload decreases again because new trust is created between the agents. Thus, it is important to note to which of the chromosome signals the evolutionary agents pay

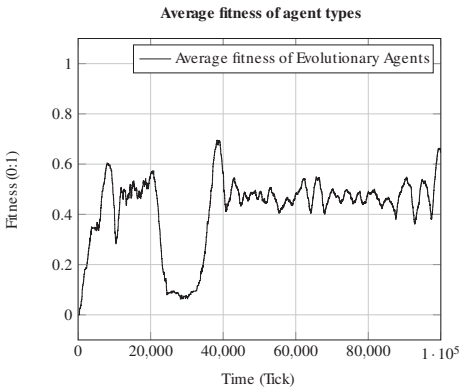


Figure 7.14: Results of experiment 2: Average fitness of evolutionary agents

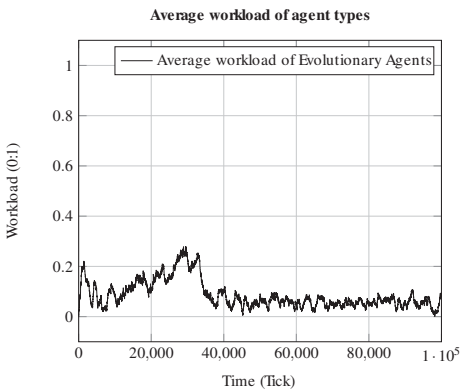


Figure 7.15: Results of experiment 2: Average workload of evolutionary agents

attention: paying attention to the others' fitness is crucial to an agent's success. After gene 4 wins through, the chromosome structure stabilizes and it becomes obvious that, in this experiment, it is not important for the evolutionary agents to pay attention to their own intentions based on gene 1 and to the reputation of the partners. Furthermore, it can be seen in Fig. 7.17 that the number of agents with a gene 8 develops to a high value. This means that agents trust agents with a fitness higher than their own, which also leads to a stable population of agents with a high fitness.

Thus, agents pay attention to the fitness of other agents (percentage of gene 4 near 1) and imitate agents with a high fitness (percentage of gene 8 near 1).

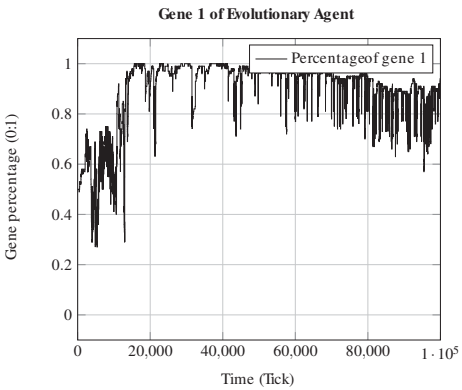


Figure 7.16: Results of experiment 2: Amount of gene 1 of evolutionary agents

7.6.3 Evolutionary Agents vs. Egoistic Agents

In Fig. 7.18, the average fitness of the evolutionary agents and the egoistic agents is plotted. Comparing the two fitness curves, it can be seen that the fitness of the egoistic agents does not exceed a value of 0.1. The fitness of the evolutionary agents increases at the beginning and reaches a value of 0.5, but the fitness starts to decrease at tick 23,000. After a stabilisation in the lower range (< 0.2), at tick 65,000, the fitness starts to increase again. Hence, the fitness curve of the evolutionary agents can be separated into three sections. The first section goes from tick 0 to tick 30,000. In this section, the evolutionary agent has a high fitness. The second section is the

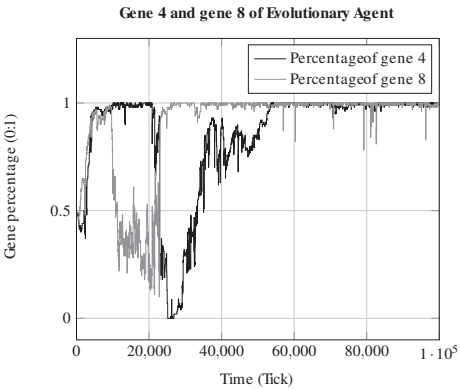


Figure 7.17: Results of experiment 2: Amount of gene 4 and gene 8 of evolutionary agents

interval from tick 30,000 to tick 80,000 in which the evolutionary agents have a low fitness. But in the third section from 80,000 to 100,000, the fitness is high again. The decreasing fitness in the second interval is because the strategy of the evolutionary agents changed at tick 30,000. This can be seen in Fig. 7.19 in the distribution of gene 5. At tick 30,000, the evolutionary agents stop paying attention to the workload. Therefore, they start to accept WUs from egoistic agents, which have a very low workload during the whole simulation. This causes the drop in the fitness of the evolutionary agents because they process the WUs of the egoistic agents but do not get their distributed WUs processed in return. Later, the strategy changes back and the evolutionary agents are able to detect the egoistic agents because of their low workload.

Overall, evolutionary agents are also quite successful in homogeneous systems. However, the amplitude of the fitness is large, as there are changes in the genes while trying to adapt the chromosome structure to the self-referential fitness landscape in a continuously changing environment (cf. [79] (Chap. 2)).

This section introduces the evolutionary agent as an approach to a self-optimising trust-adaptive agent. The results of experiment 1 show that, in a heterogeneous system, evolutionary agents can achieve a higher fitness than trust-adaptive iTC agents, with a good reputation and low workload. In other words, evolutionary agents learn

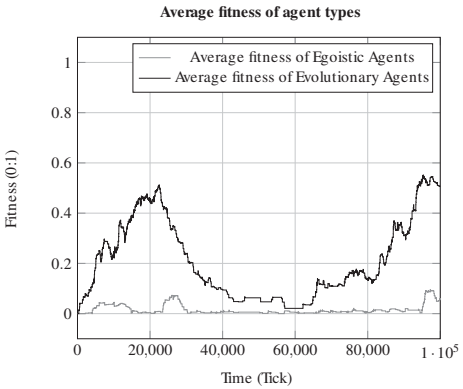


Figure 7.18: Results of experiment 3: Average fitness of evolutionary agents and egoistic agents

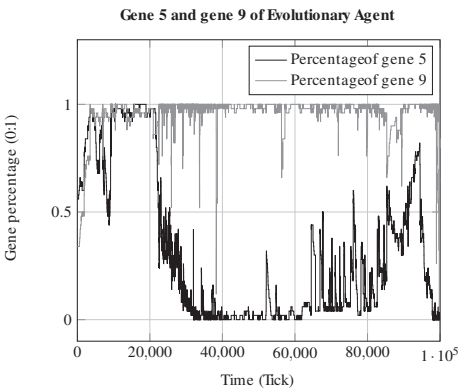


Figure 7.19: Results of experiment 3: Amount of gene 5 and gene 9 of evolutionary agents

to exploit trust-adaptive iTC agents without any malicious behaviour coded in their genes. Experiment 2 shows that evolutionary agents are able to interact in homogeneous systems as well. In experiment 3, we have seen that evolutionary agents are also successful in a system where disturbances are caused by egoistic agents. The former are able to detect the misbehaving egoistic agents, although the fitness is not always stable. Surprisingly, evolutionary agents do not learn trust-based strategies; they even ignore the trust values to which they have access. This is because the observation of trust is relevant only if it leads to advantageous behaviour modification, which is not necessarily the case in the current experimental setting of our simulation. Therefore, we aim at investigating further learning techniques like that of the learning agent (Section 6.8.3), whose results are presented in the following section.

7.7 Learning Optimal Behaviour at Run-time

In this section, we evaluate, how agents can learn to optimise their behaviour if they use predefined behaviour settings as a starting point of the learning algorithm. We used the implementation of our adaptive agent architecture and replaced the TT^{acc} decision table with an extended version of CACLA (as presented in Sections 4.1.1 and 6.8.3), learning the decision plane of trust-adaptive agents at run-time.

We conducted long-term experiments in order to give the learning function the ability to tune in.

7.7.1 Metrics for Learning Reward Functions

In order to implement the learning mechanism CACLA, we must determine which metrics (out of all the metrics that we use in our application scenario) are best suited as a reward function for the learning algorithm. Three requirements have to be fulfilled for a metric to be a good reward function:

- The metric has to create relative values in an interval $(0, 1)$. Absolute values have to be normalised before they can be used.
- A maximisation or minimisation of the metric value must be useful for the agent and the system. If maximising or minimising leads to undesired behaviour in the system, the metric is not applicable.
- The metric has to be calculated in fixed time intervals (e.g. after each job

completion) because the learning algorithm needs the value for run-time adaptation.

Table 7.1 shows the taxonomy used to define suitable reward function metrics.

Metric	Criteria			No. of Fulfilled Criteria
	Relative Values	Mini-/Maximisation Useful	Run-time Evaluation	
Makespan	–	√	–	1
Flow Time	–	√	√	2
Flow Time Ratio	√	√	√	3
Scheduling Success Rate	√	√	√	3
Average Resource Utilisation	√	–	√	2
Waiting Time	–	√	√	2
Throughput	–	–	√	1
Waste	–	–	√	1
Waste Ratio	√	√	√	3

Table 7.1: Taxonomy of grid metrics as reward functions

According to these criteria, we found three candidates for reward function metrics:

- The *flow time ratio* is the sum of interval lengths from job activation to job completion of all jobs normalised by the calculation times of all jobs. Minimising the flow time ratio means maximising the agent's performance. The flow time ratio can be evaluated at run-time and is a relative value.

- The *scheduling success rate* defines how many work units an agent has submitted to other agents during the first attempt. It is a relative value, computable

at run-time, and maximising this value leads to performance improvement on the agent and system levels.

- The *waste ratio* defines the amount of wasteful computation (e.g. cancelled work units). It is a relative value, can be evaluated during run-time, and minimising the waste ratio optimises both agent and system performance.

We evaluated CACLA with the three reward functions that we have analysed to be suitable for our system in order to determine which reward function is best in which situation and overall. For each reward function, we have evaluated five different agent populations. The agents in these populations are

- Modular agents: Agents that show trust-adaptive behaviour in the beginning, but that switch to the threshold plane they have learned at run-time. In the implementation used for this evaluation, modular agents use CACLA with different reward metrics.
- Free riders: Agents that submit work units, but refuse to work for others.
- Egoistic agents: Agents that have a high probability to cancel a work unit they have accepted in the first place—for instance, if the user of the agent's machine suddenly withdraws the resources from the agent. free-riders and egoists are disturbances within the system.
- Adaptive agents: For comparison, we repeated all of the experiments using our standard trust-adaptive agents with the TT^{acc} decision table instead of modular agents.

The quantity of modular agents varies from 100%, 80%, 60%, 40% to 20%. The remainder of the population consists of *egoists* and *free-riders*. Table 7.2 shows an overview of the different situations that we have evaluated.

Each experiment has been conducted 20 times, and the resulting graphs show the mean average of these 20 runs in order to eliminate random effects.

7.7.2 Performance of Learning Modular Agents in Different Disturbance Situations

In this section, we present the experimental results of CACLA, measured using the metrics we analysed as useful for the reward function as well as the fitness, which is not one of the metrics used to optimise the function learned, and is thus independent from the reward functions.

Combination	Population		
CACLA &	Modular	Egoists	free-riders
Flow Time Ratio	100	0	0
	80	10	10
	60	20	20
	40	30	30
	20	40	40
Scheduling Success Rate	100	0	0
	80	10	10
	60	20	20
	40	30	30
	20	40	40
Waste Ratio	100	0	0
	80	10	10
	60	20	20
	40	30	30
	20	40	40

Table 7.2: Evaluation setup

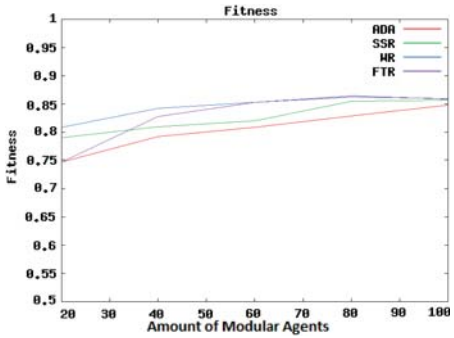


Figure 7.20: Fitness of CACLA (ADA) agents with SSR, WR and FTR as reward functions compared to adaptive agents in different disturbance situations

In Figure 7.20, the mean fitness of CACLA agents with three different reward metrics (*flow time ratio (FTR)*, *scheduling success rate (SSR)*, and *waste ratio (WR)*) is given and compared to the fitness of non-learning adaptive agents. Fitness is a weighted sum of the performance that the agents gain from the system and the effort that they expend to reach this performance. We regard this metric because it is not one of the metrics used in the reward functions; thus, it measures the performance independently from the reward, which is used in the learning function's feedback loop. This has been done for each population combination from table 7.2.

It can be seen that, in all cases, learning leads to a performance improvement. In particular, the combination of CACLA with *waste ratio* or *scheduling success rate* is successful.

As fitness is an aggregation of success and effort, we are also interested in the effects that the different reward functions for CACLA show in other metrics, especially those used as reward metrics.

Figure 7.21 shows the mean *flow time ratio* of the agents in the different disturbance situations. Again, the combinations of CACLA and *SSR* as well as *WR* lead to the most successful configurations. CACLA with *FTR* as a reward function delivers worse results than the adaptive agents in situations with many (60% to 80%) disturbing agents.

Similarly, the *scheduling success rate* (see Fig. 7.22) of modular agents is best when CACLA combines with *SSR* and *WR*; CACLA and *FTR* show a lowered per-

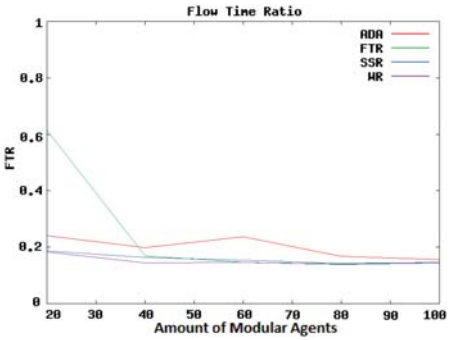


Figure 7.21: Flow time ratio of CACLA agents with SSR, WR and FTR as reward functions compared to adaptive agents in different disturbance situations

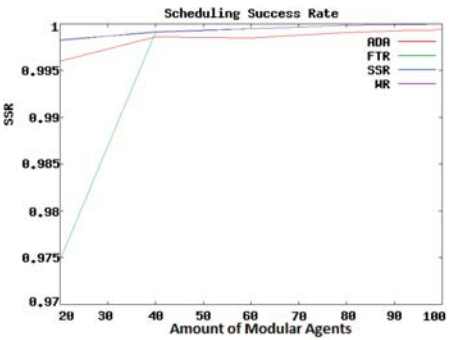


Figure 7.22: Scheduling success rate of CACLA agents with SSR, WR and FTR as reward functions compared to adaptive agents in different disturbance situations

formance under high disturbance than do adaptive agents.

In Figure 7.23, the *waste ratio* evaluation of this scenario can be seen. Under high disturbances, adaptive agents have a slightly better performance; low to medium disturbances are again best dealt with by CACLA + *SSR* and CACLA + *WR*.

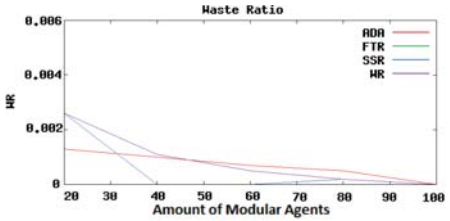


Figure 7.23: Waste ratio of CACLA agents with SSR, WR and FTR as reward functions compared to adaptive agents in different disturbance situations

In order to take a look into the agents, we plotted the decision plane the agents have learned. An example is given in Figure 7.24. In general, it can be said that agents learn quite similarly to the TT^{acc} decision planes. If the workload increases, the variance between the agents' decision planes grows, and they learn different strategies to cope with tincreased load.

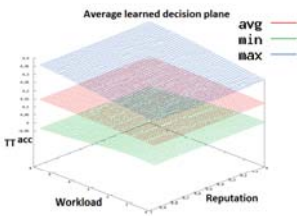


Figure 7.24: Decision plane learned at runtime

7.7.3 Summary: Learning Agent

After having analysed which learning techniques are best suited to improve the decision mechanisms of our trust-adaptive agents at run-time. In this section, we

evaluated which effect the application of our extended CACLA learning algorithm had on the performance of the agents, compared to a designer-given threshold decision table. In previous experiments, we realised that the run-time behaviour of Hedger was unsuitable for the complexity of our system. Hence, we concentrated on an enhanced version of CACLA. We evaluated which reward metrics are best suited for use by CACLA to determine an action's success. All in all, it can be said that CACLA in combination with the waste ratio metric led to the best results, directly followed by CACLA using scheduling success rate as the reward metric. CACLA combined with flow time ratio was unable to cope with high disturbance (60% and more free-riders and egoists).

Analysing the decision planes that the agents have learned, we found that the more critical the situation becomes (e.g. high workload), the greater the number of different solutions for decision planes that agents learn at run-time.

7.8 Using Predictions to Act Proactively

In this section, the functionality of tactical agents 6.9.1 is evaluated. We here regard multi-step tactical agents, which means that agents are able to analyse a set of former incidents and thus predict future developments of their situation.

In this experimental setting, we confronted 70 tactical agents with 15 egoistic agents and 15 free-riders. Jobs to be distributed by the agents occurred according to a normal distribution within 2000 and 5000 ticks.

Aim of this experiment was to evaluate, whether the inclusion of the ability to predict future workload situations in agent decisions improves the performance of agents. As work units usually appear in bursts (each time a job has to be distributed), it is useful to adapt the agent behaviour as soon as a high workload is likely to appear.

Figure 7.25 shows the performance of agents without the usage of workload predictions. We conducted the same set of experiments with identical seeds, but this time enabled agents to adapt their behaviour earlier, which is as soon as the prediction indicates that the workload will rise in the future. The results of the experiments with usage of workload prediction are shown in Figure 7.26.

It can clearly be seen that the inclusion of workload prediction in agent adaptivity can be used to increase agent as well as system performance. Adapting their behaviour early based on prediction increases the speedup of agents. Due to the proactivity enabled by workload prediction consideration, agents reach a faster calculation of their work units. Analogously, the flow time ratio decreases, which means

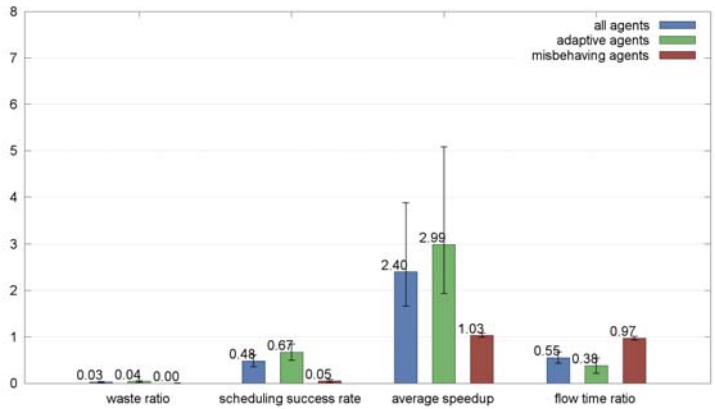


Figure 7.25: Performance without workload prediction

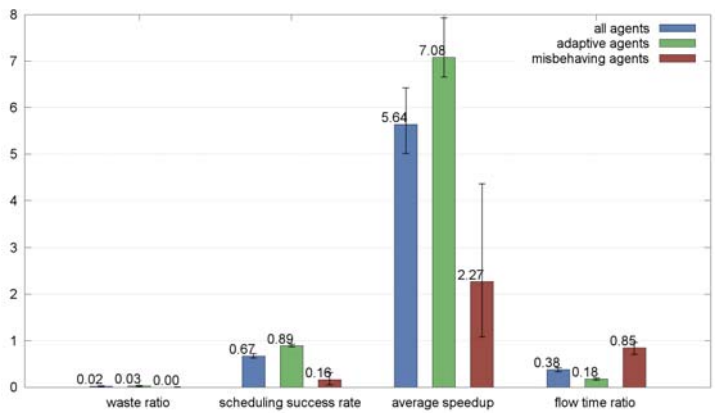


Figure 7.26: Performance with workload prediction

that agents reached a faster calculation of their jobs.

The scheduling success rate has been increased by using workload prediction, which means that agents are more likely to find a suited cooperation partner.

The performance improvements reached by the consideration of workload prediction in agent decisions are further underlined by the comparison of Figures A.5 and A.6 in the appendix. These figures show that the waiting time as well as the turnaround time could be decreased, work units could be retrieved faster if agents adapted proactively to future workload increase.

7.9 Inclusion of Norms into Local Agent Decision Making

In this section, we present the evaluation results reached using norm-aware agents (Section 6.9.2). In order to show that the inclusion of institutional norms can lead to performance improvement on the agent and system levels, we triggered a situation that cannot be resolved using local knowledge alone. In this experiment setting, the system is in an overload situation, where agents have to cope with a greater workload than their standard parameters can handle. Such a situation cannot be observed locally. Therefore, a norm manager observes the system and, as soon as it detects the overload situation, legislates a norm recommending that the agents adapt their parameters so that it leads to a rise in their cooperative behaviour and, thus, an increase in their likelihood of accepting work units from other agents.

We will first show how the performance of the agents and the system suffers from this overload situation and how an overload can lead to a trust breakdown over time. We will then show how the application of norms leads to slight performance improvements and, more importantly, how a trust breakdown can be prevented by norms.

The experiments presented here are conducted in a scenario involving 70 agents (either simply trust-adaptive or norm-aware), 15 egoistic agents and 15 free riders. The job generation pause has been set as the range from 1,500 to 3,000 ticks, which in this setting is an extremely high load situation in which agents cannot reach good performance values by design. The simulation lasted 30,000 ticks, which is roughly the time during which such an undesired system state can occur. Therefore, longer experiments with such special system states would be unrealistic.

7.9.1 Overload Workload Situation without Norms

In this experiment, we show how the system behaves in a situation where no norm tries to counter the agent behaviour in overload.

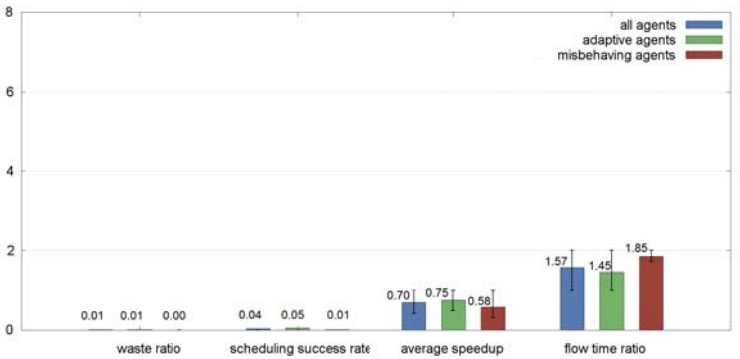


Figure 7.27: Performance of trust-adaptive agents without overload workload norm

Figure 7.27 shows the performance of trust-adaptive agents in an overload situation without the overload workload norm. Of course, we cannot expect agents to achieve a good performance in this critical situation. Despite the low performance, the iTC formation still works, and misbehaving agents record a worse performance than do adaptive agents.

In Figure 7.28, we take a deeper look into the agent behaviour and reputation of the different agents. It is obvious that the overload situation causes a complete loss of trust relations in the system; all agents reach a negative reputation. Thus, an overload situation can lead to a trust breakdown from which the system cannot recover using only local knowledge. Even if the overload situation is resolved (for instance, if the applications do not produce too many jobs any more), the agents still remain in a situation where no cooperation is possible and, thus, no good performance can be reached.

We will now repeat the experiments in the same situation, but with an institutional overload workload norm in place.

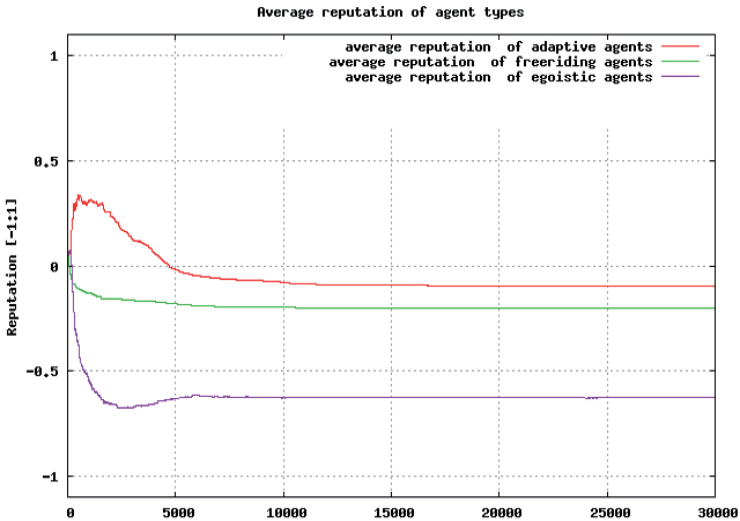


Figure 7.28: Reputation of agents without overload workload norm

7.9.2 Overload Workload Situation with Norm

In this section, we repeat the above experiment, but enable the norm manager to legislate an overload workload norm. Since the workload is beyond the amount that can be handled by the system, we do not expect a speedup that is greater than 1, but rather, a slight performance improvement in combination with the prevention of a trust breakdown, which was inevitable in this situation without institutional norms.

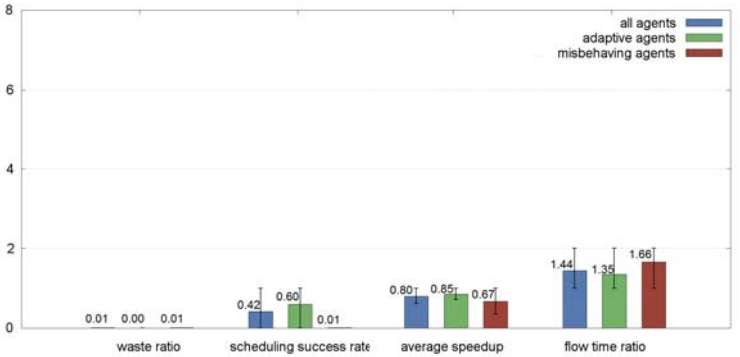


Figure 7.29: Performance in overload situation with overload workload norm

Figure 7.29 shows the performance metrics of the overload situation with norms activated by the norm manager and considered by the norm-aware agents.

We can see that the performance values involving norms are higher than in the identical situation without norms (Fig. 7.27). Regarding the speedup, for instance, we can see that the improvement by considering norms in this situation is more than 12%. As stated above, we could not expect the speedup to become greater than 1 in this setting because it is an overload situation in which agents cannot gain speedup by distributing their jobs in the grid.

Apart from a slight performance improvement in highly critical situations, we want the norms to maintain the system in a state from which it can recover. In order to evaluate this, we regard the reputation of the agents, which, in this situation without norms (Fig. 7.28), had been negative for all agent types.

Figure 7.30 shows the reputation of the agents in the overload situation with an overload workload norm activated.

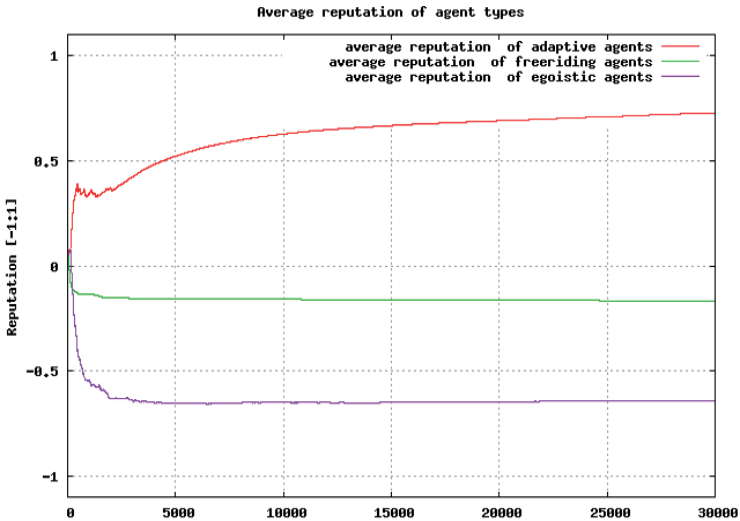


Figure 7.30: Reputation in overload situation with overload workload norm

The misbehaving agents (free-riders and egoistic agents) still achieve a negative reputation, which is valid and aimed for by the system designer. But the overload workload norm leads to a situation where agents know that the system is in an overload situation. They try to be more cooperative, but are also warned that despite more cooperation, other agents might still not be able to compute all requested work units. Therefore, although the trust and reputation mechanism has not been manipulated by the overload workload norm, agents are able to maintain a good reputation (nearly 0.7). Therefore, as soon as the overload situation is overcome, agents are in a state where they can easily find cooperation partners, thereby reaching a good performance. Thus, the overload workload norm was used to prevent the system from running into a trust breakdown situation.

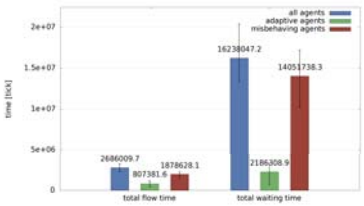
7.10 Overhead Reduction by Using the Adaptive Observation Model

The aim of the Adaptive Observation Model (Section 6.9.3) is to enable agents to manipulate the information they observe. This can affect the parameters they observe, the scope from which they receive the parameters (e.g. a set of agents, all agents), and the frequency with which the parameters are updated. Reducing the parameters, scope and frequency leads to a reduction of communication effort and, therefore, especially in a scenario like the TDG, a better usage of the available bandwidth.

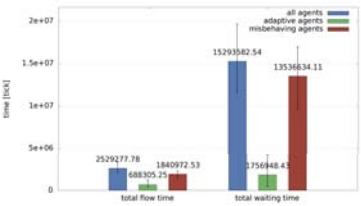
7.10.1 Variation of Parameter Type and Scope

Here, we present an experimental setting conducted in order to show how the variation of parameter types and the set of agents from which they are obtained can be performed.

Figure 7.31 shows a comparison of two different observation models used by the agents. In the first set of experiments (Fig. 7.31(a)), the agents used the standard observation model of iTC agents (see Fig. 6.17). One observable of this model is WL_{TC}, the average workload of the agents that are regarded as trustworthy by the agents and, thus part of its iTC. Obtaining this parameter is expensive because all trustworthy agents have to be asked as often as possible in order to obtain the correct value. This value is used as an input in the worker role to adapt the agent's cooperative behaviour to the current workload situation. In order to minimise the communication effort for this value, in Figure 7.31(b), we conducted the same set of



(a) High communication effort to retrieve average workload from all agents of the iTC



(b) Reduced communication effort by using agent's own workload as input for the decision mechanism

Figure 7.31: This awareness adaptation not only reduces communication effort, but can also improve performance (flow time and waiting time).

experiments, but used the agent's own workload instead of determining the average of all trustworthy agents.

This slight adaptation of the observable (private knowledge instead of community knowledge) not only led to a similar performance, but even to a slight improvement of the average flow time and waiting time. This indicates that a smart reduction of observed information can lead to comparable or even better agent decisions. This adaptation of the observation model led to a reduction of communication costs by about 14%.

7.10.2 Variation of Sampling Distance

In this experimental setting, we evaluated how the sampling distance of the observation model can be adapted to the available bandwidth in the system. The less bandwidth is used for management data like reputation or workload information exchange, the more bandwidth is available for the user data, e.g. the work units and their results. In volunteer desktop grid systems, the bandwidth of the agents is usually defined by the internet connection of the underlying PC. As the grid client software only uses the idle computing resources, the foreground tasks of the user should have a higher priority and therefore, the bandwidth available to the agent is only the remaining bandwidth, which is not used by the foreground tasks. Therefore, limited bandwidth is an issue such systems need to cope with. The sampling distance of the observation model defines, how often the observed parameters are updated. On the one hand, updating more often than necessary leads to an un-

necessary reduction of the available bandwidth. On the other hand, be updating too seldom, agents can miss important information. The types of information which might be missed by a sampling distance, which is too high for the agent to make a good decision, are:

- new agents (esp. potential workers) entering the system
- agents leaving the system
- changes in the system's workload situation, which might need behaviour adaptation
- changes in the behaviour of an agent (e.g. becoming more altruistic and thus a potential worker or becoming egoistic and thus untrustworthy)

Therefore, the ability to adapt the sampling distance at runtime to the current situation's needs is an important ability for agents and thus one of the key advantages of the adaptive observation model.

In the simulations conducted for these experiments, we use a simple bandwidth model, which, as soon as the available bandwidth of an agent in a time sliding window is exceeded, has a probability of a collision leading to message extinction. Despite several collision avoidance techniques, collisions are still an issue, especially in UDP (User Datagram Protocol)-based connections like used by Wireless LAN (e. g. discussed in [120]. Moreover, several foreground applications run by users (e.g. TV streaming) exploit a lot of bandwidth and thus the grid client agent needs to adapt its bandwidth usage accordingly.

In Figure 7.32, we evaluated which sampling distances are best suited for a scenario of 70 AOM agents, 15 free-riders and 15 Egoistic agents with a normal distribution of workload.

We varied the bandwidth from 10000 messages per tick interval to 300000 messages per interval. Additionally, we evaluated the setting with unlimited bandwidth, which used to be the standard setting at design time of the TDG and the agent algorithms. In the scenario setting with unlimited bandwidth, a sampling distance of 1, which means updating the observed parameters each tick, reached the best speedup, followed by sampling distance 10. In this setting, a sampling distance of 100 still reached quite good results. This is reasonable because, in this setting, neither the agent society nor the workload situation changed severely. Therefore, the agents did not "'miss"' many information between two sampling time points.

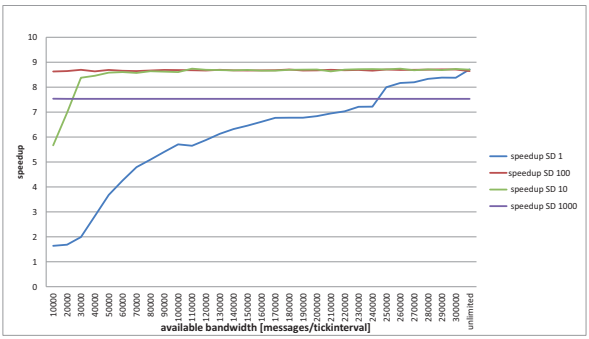


Figure 7.32: Speedup of different sampling distances used in different bandwidth limitation scenarios

In general, especially if the system is dynamic (new, unknown agents enter the system), a smaller sampling distance is usually better, but as soon as the bandwidth is limited, our evaluations have shown that the timeliness of the information is less important than the reduction of message overhead, because lost messages lower the performance severely.

As soon as the bandwidth is limited, the formerly optimal sampling distance 1 shows to reach lower speedup (see Fig. 7.32), because too many messages collide and information like reputation updates are lost due to collisions. Here, the sampling distances 10 and 100 both show a constantly high speedup despite bandwidth reduction. Nonetheless, a sampling distance of 1000 led to a low speedup with unlimited bandwidth, because, despite the rather static scenario, the information retrieved from the observation was too deprecated to allow for optimal behaviour decisions. In situations with very low bandwidths (up to 20000), the sampling distance 100 reaches a higher speedup than sampling distance 10, collisions are prevented, but still the information is more accurate than in the scenario using sampling distance 1000.

This evaluation shows, that there is not one sampling distance optimal for each situation. Moreover, a continuous autonomous runtime adaptation of the sampling distance enables agents to always have the best-suited view for the current situation.

The sampling distance should be chosen by the agent based on:

- the system size (number of agents),

- the system dynamics (volatility of agents),
- the available bandwidth,
- the expected behaviour of the agents in the system,
- the system workload,
- the own workload,
- and the amount of work units to be distributed.

The system size influences the number of messages the agent needs to send and receive, both for the actual cooperation communication (which is the payload, e.g. requests to compute a work unit, sending results) and for the management information like ratings.

The system dynamics, in the TDG defined by the volatility, is a measure how likely it is that agents join or leave the system. The available bandwidth is an issue as soon as the system size and dynamics require more messages than can be send through the channel in order to have a full view of the occurrences in the system.

The expected behaviour can, for instance, be deviated from the reputation of the agents (e.g. by using the prediction methods presented earlier in this chapter): If an agent has shown constant behaviour in the past, he is more likely to stay constant. The higher the workload in the system is, the more communication needs to be preserved for payload messages.

The more own workload the agent has or will have in the future, the more payload messages he will send, but the more important an up-to-date view of the available agents and their reputation is for him.

Based on this analysis, our AOM agents define their observation model as follows: in general, they choose a sampling distance which is rather high in situations where agents are known and remain constant. As soon as the situation changes and requires the agents to stay more alarmed (e.g. unknown agents enter the system), the sampling distance is minimised to a value, which is small, but still does not overuse the available bandwidth. This adaptive design of the observation model, especially regarding the sampling distance, enables agents to always have the necessary information to make optimal decisions in every situation.

7.11 Summary

The evaluation has underlined the advantages of the different agent aspects in this thesis. We have seen that trust-enhanced interaction leads to performance improvements and a higher system robustness. Self-organised mechanisms on agent levels lead to the creation of iTCs, which are able to detect and isolate misbehaving agents.

We compared our trust-based algorithms to the state of the art of trust-based and adaptive grid scheduling systems. The self-organised iTC creation led to a better grid performance (total flow time, total waiting time) than H-Trust in general due to the better implementation of fairness within the distributed algorithms. Organic Grid reached a better performance in undisturbed system states, but could be outperformed by our algorithms in systems under disturbance.

On the Controller level, we have shown that evolutionary algorithms and learning can lead to further performance improvement. Especially our extended adaptation of the CACLA learning algorithms show that agents are able to learn better behaviour at runtime despite the complexity of continuous situation and action spaces.

In order to make agents able to adapt proactively before a situation change occurs, we implemented and evaluated prediction techniques on agent level. These prediction techniques lead to a higher horizon of agent behaviour and, therefore, to performance improvement in changing situations.

By enhancing the agents with norm consideration, we could show that hierarchical institutional components within self-organised systems are worthwhile and able to overcome situations where local agent knowledge alone does not suffice. We could evaluate that these mechanisms are able to prevent the system from being drawn into a trust breakdown situation.

The different ways of using the Adaptive Observation Model for communication reduction have been shown to improve the bandwidth usage within the system and therefore make agents adapt their overhead to the allowance and needs of their situation.

Chapter 8

Conclusion

In this thesis, we have shown how agents in open systems can be enhanced with a trust and reputation mechanism and trust-based interaction algorithms in order to cope with the information uncertainty caused by the open nature of the systems.

We mainly focused on the class of open organic computing systems, which means that solutions containing self-x properties and observer/controller-pattern-based agents are extended with the ability to observe and use trust information in complex cooperation decisions.

8.1 Discussion

Our application scenario TDG was chosen due to its open nature and the relevance of the system to current research. Moreover, we analysed the state of the art of desktop grid systems as well as the scheduling and matchmaking techniques in such systems, and compared our approaches to comparable candidates from literature.

We presented our own trust and reputation system on which all trust information of the agents are based. This system covered both, the requirements of our adaptive agent architecture and of the application scenario TDG, which were not met by the current state of the art of trust and reputation models.

The adaptive agent architecture is a general framework where the complexity of agents in open systems is decomposed into different decision levels. For each agent complexity level, a class of agents exists, which is able to make decisions based on

the available awareness on this level.

For each class of decision complexity levels (trust-neglecting, trust-aware, trust-adaptive, trust-strategic), we presented different techniques pertaining to how the controller side of the agent can be implemented in order to use the available information in the best possible way.

Trust-neglecting agents have been introduced as reference implementation and showed that trust-consideration leads to a performance increase. Two different implementations of trust-aware agents (egoistic agents, free-riders) have been used as disturbances within the system. These disturbances enabled us to demonstrate the robustness of our agents and algorithms.

Trust-adaptive agents were implemented in three different design examples.

ITC agents are able to identify and isolate these disturbing agents using local algorithms; thus, they ensure a good performance and enhance the robustness of the system.

Based on an analysis of the state of the art, two possibilities regarding how learning techniques have been applied to trust-adaptive behaviour have been presented. The evolutionary agent uses a bit-code representation of its behaviour and evolutionary algorithms strategies from scratch, thereby finding new solutions. The learning agent uses the behaviour encoded in the already-successful iTC agents as a starting point and finds new behaviour solutions in that area by applying a modified version of the CACLA algorithm.

Trust-strategic agents include long-term and institutional information in their decision-making process.

One implementation of this class involves tactical agents that identify trends in situations, make predictions on how these will develop in the future, and adapt their behaviour proactively.

Norm-aware agents as a second implementation of trust-strategic agents are able to consider institutional norms in their decision. Therefore, they can overcome situations that cannot be resolved with local knowledge alone. We have shown that norm consideration can help agents to overcome overload situations and thus prevent the system from further damages, e.g. a trust breakdown.

AOM agents as the last implementation of trust-strategic agents combine the different features and advantages of the agent types. They are able to adapt their observation model to the current situation. Therefore, they also select the controller behaviour that is best suited for the situation from the set of all agent-type behaviours. We have shown that by evaluating only the information that is relevant

for decision making in the agent's current situation, AOM agents save communication overheads, thereby leading to a better usage of bandwidth.

On the system level, we have presented two types of TCs. iTCs are formed as an emergent effect of locally interacting iTC agents. This bottom-up formation led to an exclusion of misbehaving agents and thereby, to a performance improvement in well-behaving agents. Moreover, this self-organised formation of the TC leads to an enhancement of system robustness.

eTCs have been presented as an example of TCs with higher-level components. In contrast to the implicit definition of a TC, these are managed by an institutional component. Moreover, membership information is available to and from all agents. Therefore, this agent organisation allows for algorithms that require a closed environment. The eTC consists of only trustworthy agents. Therefore, it is possible to use algorithms without trust-enhancement within the closed subsystem of the actual system. For instance, agents could rely on a pull mode like in the Organic Grid, which might further enhance the load balancing in this undisturbed subsystem.

To sum up, we have presented a combined architecture for an agent hierarchy and different implementations of classes of this agent hierarchy. We have shown the benefits of the agent classes and each of their characteristics by evaluating them in a desktop grid scenario. Nonetheless, the concepts of the agents presented in this thesis can be applied to many different domains.

8.2 Generalisation

The agents that we have introduced and analysed in this thesis are able to use trust information and adapt their behaviour in a self-organised way. Therefore, they can be used in each system that is suited for Organic Computing and self-organisation techniques.

Due to the encapsulation of the agent decision making process, is also possible to adapt specific aspects of the agent abilities to other application scenarios and domains, even without implementing the complete adaptive agent architecture. For instance, the adaptive observation model as a general concept to enable agents to adapt their awareness at run-time and thus minimse observation overhead and maximise the performance in situations with limited bandwidth is applicable in various domains and settings. Similarly, the algorithms developed to consider trust, enable prediction, learn in dynamic environments and consider norms on agent level can be transferred to other agent-based applications as well.

Moreover, the concepts presented in this thesis have outstanding abilities with respect to open systems. They overcome the information uncertainty by using trust and reputation information, thereby improving the robustness and performance of such systems. Similarly, an open MAS can be enhanced by the trust-based algorithms presented here.

We want to point out that desktop grids, in which the agents in this thesis are evaluated, are only one example of the applications that could benefit from trust-enhanced agents. Desktop grids are an instance of the general class of CPR problems. One example of such open CPR problems is energy markets. The so-called *prosumer*, which produces and consumes energy (based on the availability of its energy resource), is growing in importance and self-organised distributed solutions to match energy resources and requirements are needed. Therefore, the agent-based algorithms, architectural frameworks and results presented in this thesis can be applied to any instance of CPR problems. In particular, CPR problems that have to cope with the openness and volatility of subsystems are candidates that benefit from the trust-aware agents presented in this thesis.

Bibliography

[1] A. Jøsang and S. J. Knapskog, "A metric for trusted systems (short paper)," in *Proceedings of the 1998 IFIP/SEC International Information Security Conference, Kluwer*, 1998, pp. 16–21. iv, 39, 43, 53

[2] Emre Cakar, Moez Mnif, Christian Müller-Schloer, Urban Richter, and Hartmut Schmeck, "Towards a quantitative notion of self-organisation," in *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC 2007)*, September 2007, pp. 4222–4229. iv, 58

[3] SungJin Choi, HongSoo Kim, EunJoung Byun, MaengSoon Baik, SungSuk Kim, ChanYeol Park, and ChongSun Hwang, "Characterizing and classifying desktop grid," in *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, May 2007, pp. 743–748. 3

[4] R. Falcone and C. Castelfranchi, "The human in the loop of a delegated agent: the theory of adjustable social autonomy," in *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 2001, vol. 31, pp. 406–418. 3

[5] BOINC Berkeley Open Infrastructure for Network Computing, "https://boinc.berkeley.edu/," [Online; accessed 21-March-2014]. 3

[6] Seti@home, "http://setiathome.berkeley.edu," [Online; accessed 20-Sept.-2013]. 3

[7] Hado van Hasselt and Marco A. Wierling, "Reinforcement learning in continuous action spaces," *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)*, pp. 272–279, 2007. 4, 59, 62, 110

[8] A.J. Chakravarti, G. Baumgartner, and M. Lauria, "The organic grid: self-organizing computation on a peer-to-peer network," *IEEE Transactions on*

Systems, Man and Cybernetics, Part A: Systems and Humans,, vol. 35, no. 3, pp. 373–384, May 2005. 4, 23, 34, 137

[9] E. Ostrom, *Governing the Commons*, Cambridge University Press, 1990. 4, 67

[10] Sungjin Choi, Rajkumar Buyya, Hongsoo Kim, and Eunjoung Byun, "A taxonomy of desktop grids and its mapping to state of the art systems," Tech. Rep., Grid Computing and Distributed Systems Laboratory, The University of Melbourne, 2008. 5

[11] David P. Anderson, "Public computing: Reconnecting people to science," in *Conference on Shared Knowledge and the Web. Residencia de Estudiantes, Madrid, Spain*, 2003. 5

[12] Ian Foster, Carl Kesselman, and Nicholas Jennings, "Brain meets brawn: Why grid and agents need each other," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS*, 2004, pp. 8–15. 7, 19

[13] T. Schöler and C. Müller-Schloer, "An observer/controller architecture for adaptive reconfigurable stacks," *Proceedings ARCS 05*, pp. 139–153, 2005. 8, 60, 87

[14] S. Varrette, E. Tantar, and P. Bouvry, "On the resilience of [distributed] eas against cheaters in global computing platforms," in *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011s*, May 2011, pp. 409–417. 17

[15] A.L. Beberg, D.L. Ensign, G. Jayachandran, S. Khaliq, and V.S. Pande, "Folding@home: Lessons from eight years of volunteer distributed computing," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, May 2009, pp. 1–8. 17

[16] D.P. Anderson, "Boinc: a system for public-resource computing and storage," in *Fifth IEEE/ACM International Workshop on Grid Computing, 2004. Proceedings.*, Nov. 2004, pp. 4–10. 17, 18

[17] Michael R. Garey and David S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness.*, W. H. Freeman and Company, New York, 1979. 18

[18] M. MadadyarAdeh and J. Bagherzadeh, "An improved ant algorithm for grid scheduling problem using biased initial ants," in *Computer Research and Development (ICCRD), 2011 3rd International Conference on*, 2011, vol. 2, pp. 373–378. 18

[19] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," in *IEEE Transactions on Software Engineering*, 1989, pp. 1427–1436. 18

[20] R. Sharma, V. Kant Soni, M. Kumar Mishra, and P. Bhuyan, "A survey of job scheduling and resource management in grid computing," vol. 4, no. 4, pp. 418–424, 2010. 18

[21] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, Aug. 2001, pp. 99–100. 19, 100

[22] Li Lin and Jinpeng Huai, "Qgrid: An adaptive trust aware resource management framework," *Systems Journal, IEEE*, vol. 3, no. 1, pp. 78–90, march 2009. 21

[23] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, Chapter 6.5, 2005. 21

[24] Krishnaveni Budati, Jason Sonnek, Abhishek Chandra, and Jon Weissman, "Ridge: Combining reliability and performance in open grid platforms," in *Proceedings of the 16th international symposium on High performance distributed computing - HPDC '07*. 2007, p. 55, ACM Press. 21, 134

[25] Dayi Zhou and Virginia Lo, "Wavegrid: a scalable fast-turnaround heterogeneous peer-based desktop grid system," in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*. 2006, IEEE. 22

[26] A.E. El-Desoky, H.A. Ali, and A.A. Azab, "A pure peer-to-peer desktop grid framework with efficient fault tolerance," in *International Conference on Computer Engineering Systems. ICCES '07.*, Nov. 2007, pp. 346–352. 22, 27

[27] Huanyu Zhao and Dr. Xiaolin (Andy) Li, "H-trust simulator, http://cs.okstate.edu/ huanyu/h-trust/h-trust.html," [Online: accessed on 10-Feb-2012], 2008. 22, 142, 143, 209

[28] J. E. Hirsch, "An index to quantify an individual's scientific research output," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 46, pp. 16569–16572, 2005. 22

[29] Congfeng Jiang, Cheng Wang, Xiaohu Liu, and Yinghui Zhao, "A survey of job scheduling in grids," in *Advances in Data and Web Management*, Guozhu Dong, Xuemin Lin, Wei Wang, Yun Yang, and Jeffrey Yu, Eds., vol. 4505 of *Lecture Notes in Computer Science*, pp. 419–427. Springer Berlin / Heidelberg, 2007. 25, 133, 134

[30] Kirk Pruhs, Jiri Sgall, and Eric Torng, "Online scheduling," in *Handbook of Scheduling: Algorithms, models, and performance analysis*, Joseph Y-T. Leung and James H. Anderson, Eds., pp. 196–231. CRC Press, Boca Raton, 2004. 25

[31] Arjav J. Chakravarti, Gerald Baumgartner, and Mario Lauria, "Application-specific scheduling for the organic grid," in *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, Washington, DC, USA, 2004, pp. 146–155, IEEE Computer Society. 26

[32] HongSoo Kim, SeockIn Kim, EunJoung Byun, ChongSun Hwang, and Jang-Won Choi, "Agent-based autonomous scheduling mechanism using availability in desktop grid systems," in *Proceedings of the 15th International Conference on Computing*, Washington, DC, USA, 2006, pp. 174–179, IEEE Computer Society. 26

[33] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, "A scalable content-addressable network," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 161–172, Aug. 2001. 26

[34] A.A. Azab and H.A. Kholidy, "An adaptive decentralized scheduling mechanism for peer-to-peer desktop grids," in *International Conference on Computer Engineering Systems. ICCES 2008.*, Nov. 2008, pp. 364–371. 27

[35] L. Canon, E. Jeannot, and J. Weissman, "A scheduling and certification algorithm for defeating collusion in desktop grids," in *31st International Conference on Distributed Computing Systems. ICDCS 2011.*, June 2011, pp. 343–352. 27

[36] J. Abawajy, "Fault-tolerant dynamic job scheduling policy," in *Distributed and Parallel Computing*, Michael Hobbs, Andrzej Goscinski, and Wanlei Zhou, Eds., vol. 3719 of *Lecture Notes in Computer Science*, pp. 165–173. Springer Berlin / Heidelberg, 2005. 27, 28

[37] Shanyu Zhao, Virginia Lo, and Chris GauthierDickey, "Result verification and trust-based scheduling in peer-to-peer grids," in *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, Washington, DC, USA, 2005, pp. 31–38, IEEE Computer Society. 27

[38] Virginia Lo, Daniel Zappala, Dayi Zhou, Yuhong Liu, and Shanyu Zhao, "Cluster computing on the fly: P2p scheduling of idle cycles in the internet," in *In Proceedings of the IEEE Fourth International Conference on Peer-to-Peer Systems*, 2004, pp. 227–236. 28

[39] Dayi Zhou and Virginia Lo, "Wave scheduler: Scheduling for faster turnaround time in peer-based desktop grid systems," in *11th Workshop on Job Scheduling Strategies for Parallel Processing. ICS 2005.* 2005, pp. 194–218, Springer. 28

[40] Chang-Qin Huang, De-Ren Chen, and Hua-Liang Hu, "Intelligent agent-based scheduling mechanism for grid service," in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics.*, Aug. 2004, vol. 1, pp. 16–21. 28

[41] D. Cenk Erdil and Michael J. Lewis, "Supporting self-organization for hybrid grid resource scheduling," in *Proceedings of the 2008 ACM symposium on Applied computing*, New York, NY, USA, 2008, SAC '08, pp. 1981–1986, ACM. 28

[42] Kai Lu and A.Y. Zomaya, "A hybrid policy for job scheduling and load balancing in heterogeneous computational grids," in *Sixth International Symposium on Parallel and Distributed Computing. ISPDC '07.*, July 2007. 29

[43] J.O. Melendez and S. Majumdar, "Matchmaking with limited knowledge of resources on clouds and grids," in *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, July 2010, pp. 102–110. 29

[44] R. Islam, Z. Islam, and N. Leyla, "A matchmaking algorithm for resource discovery on grid," in *International Conference on Information and Communication Technology. ICICT '07.*, March 2007, pp. 193–196. 29

[45] Md. Rafiqul Islam, Md. Zahidul Islam, and Nazia Leyla, "A tree-based approach to matchmaking algorithms for resource discovery," *International Journal of Network Management*, vol. 18, no. 5, pp. 427–436, September 2008. 29

[46] R. Raman, M. Livny, and M. Solomon, "Matchmaking: distributed resource management for high throughput computing," in *The Seventh International Symposium on High Performance Distributed Computing. Proceedings.*, Jul 1998, pp. 140–146. 30

[47] M. Imran Shaik, S.M. Saira Bhanu, and N.P. Gopalan, "Distributed grid resource discovery with matchmakers," in *Second International Conference on Semantics, Knowledge and Grid. SKG '06.*, nov. 2006, p. 28. 30

[48] Adriana Iamnitchi, Ian Foster, and Daniel C. Nurmi, "A peer-to-peer approach to resource discovery in grid environments," in *In High Performance Distributed Computing*. 2002, IEEE. 30

[49] Chang Liu, Zhiwen Zhao, and Fang Liu, "An insight into the architecture of condor - a distributed scheduler," in *International Symposium on Computer Network and Multimedia Technology. CNMT 2009.*, Jan. 2009, pp. 1–4. 30

[50] Zar Lwin Phyo and A. Thida, "Best resource node selection using rough sets theory," in *3rd International Conference on Computer Research and Development (ICCRD)*, March 2011, vol. 2, pp. 461–464. 31

[51] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert, "Centralized versus distributed schedulers for multiple bag-of-task applications," in *20th International Parallel and Distributed Processing Symposium. IPDPS 2006.*, april 2006, p. 10 pp. 31

[52] Victor Shafran, Gal Kaminka, Sarit Kraus, and Claudia V. Goldman, "Towards bidirectional distributed matchmaking," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 3*, Richland, SC, 2008, AAMAS '08, pp. 1437–1440, International Foundation for Autonomous Agents and Multiagent Systems. 31

[53] Raffaele Montella, Giulio Giunta, and Angelo Riccio, "An integrated classad-latent semantic indexing matchmaking algorithm for globus toolkit based computing grids," in *Parallel Processing and Applied Mathematics*, Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Wasniewski, Eds., vol. 4967 of *Lecture Notes in Computer Science*, pp. 942–950. Springer Berlin / Heidelberg, 2008. 32

[54] Jik-Soo Kim, Bobby Bhattacharjee, Peter Keleher, and Alan Sussman, "Matching jobs to resources in distributed desktop grid environments," Tech. Rep.

Technical Report CS-TR-4791 and UMIACS-TR-2006-15, University of Maryland, April 2006. 32

[55] J.-S. Kim, B. Nam, M. Marsh, P. Keleher, B. Bhattacharjee, D. Richardson, D. Wellnitz, and A. Sussman, "Creating a robust desktop grid using peer-to-peer services," in *IEEE International Parallel and Distributed Processing Symposium. IPDPS 2007.*, March 2007, pp. 1–7. 32

[56] Jik-Soo Kim, Beomseok Nam, M. Marsh, P. Keleher, B. Bhattacharjee, and A. Sussman, "Integrating categorical resource types into a p2p desktop grid system," in *Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, Washington, DC, USA, 2008, GRID '08, pp. 284–291, IEEE Computer Society. 32

[57] S. Esteves, L. Veiga, and P. Ferreira, "Gridp2p: Resource usage in grids and peer-to-peer systems," in *IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum. IPDPSW 2010.*, April 2010, pp. 1–8. 33

[58] Sungjin Choi, Hongsoo Kim, Eunjung Byun, and Chongsun Hwang, "A taxonomy of desktop grid systems focusing on scheduling, technical report: Ku-cse-2006-1120-01," Department of Computer Science and Engineering, Korea University, November 2006, Email: lotieye@csse.unimelb.edu.au. 33, 34

[59] Huanyu Zhao and Xiaolin Li, "H-trust: a group trust management system for peer-to-peer desktop grid," *J. Comput. Sci. Technol.*, vol. 24, pp. 833–843, September 2009. 34, 143, 144

[60] J Dunn, "The concept of trust in the politics of john locke," in *Philosophy in History: Essays on the Historiography of Philosophy*, 1984, pp. 279–301. 35

[61] M. Wang, F. Tao, Y. Zhang, and G. Li, "An adaptive and robust reputation mechanism for p2p network," in *Communications (ICC), 2010 IEEE International Conference on*, May 2010, pp. 1–5. 36, 51, 53, 54

[62] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *WWW '03: Proceedings of the 12th international conference on World Wide Web*, New York, NY, USA, 2003, pp. 640–651, ACM. 37, 39, 53, 82

[63] Asimina Vasalou, Astrid Hopfensitz, and Jeremy V. Pitt, "In praise of forgiveness: Ways for repairing trust breakdowns in one-off online interactions," *Int. J. Hum.-Comput. Stud.*, vol. 66, pp. 466–480, June 2008. 37, 42

[64] Junsheng Chang, Huaimin Wang, and Yin Gang, "A dynamic trust metric for p2p systems," in *Grid and Cooperative Computing Workshops, 2006. GCCW '06. Fifth International Conference on*, October 2006, pp. 117–120. 37, 42, 44, 53, 54

[65] T. Beth, M. Borcherding, and Birgit Klein, "Valuation of trust in open networks," in *Proceedings of the European Symposium on Research in Computer Security (ESORICS), Brighton, UK*. 1994, pp. 3–18, Springer-Verlag. 37, 53

[66] A.A. Selcuk, E. Uzun, and M.R. Pariente, "A reputation-based trust management system for p2p networks," in *Cluster Computing and the Grid. CCGrid 2004. IEEE International Symposium on*, April 2004, pp. 251–258. 40, 53, 143

[67] Li Xiong and Ling Liu, "Peertrust: supporting reputation-based trust for peer-to-peer electronic communities," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 16 (7), pp. 843–857, July 2004. 41, 49, 53

[68] Audun Jøsang, Ross Hayward, and Simon Pope, "Trust network analysis with subjective logic," in *Proceedings of the Australasian Computer Science Conference (ACSC'06), Hobart*, January 2006. 43, 53

[69] Junsheng Chang, Huaimin Wang, Gang Yin, and Yangbin Tang, "A new reputation mechanism against dishonest recommendations in p2p systems," in *WISE'07 Proceedings of the 8th international conference on Web information systems engineering*. 2007, Springer-Verlag Berlin / Heidelberg. 44, 45, 53, 54

[70] Gang Li, Sheng Ge, and Zhenhai Yang, "An affair-based interpersonal trust metric calculation method," in *The 9th International Conference for Young Computer Scientists. ICYCS 2008.*, November 2008, pp. 1938–1943. 45, 53

[71] L. Alboaie and T. Barbu, "Reputation system user classification using a hausdorff-based metric," in *Computational Intelligence for Modelling Control Automation, 2008 International Conference on*, December 2008, pp. 1035–1040. 46, 53

[72] D. Glynos, P. Argyroudis, C. Douligeris, and D. O'Mahony, "Twohop: Metric-based trust evaluation for peer-to-peer collaboration environments," in *Global*

Telecommunications Conference. IEEE GLOBECOM 2008. IEEE, December 2008, pp. 1–6. 47, 53

[73] Yunchang Zhang, Shanshan Chen, and Geng Yang, "Sftrust: A double trust metric based trust model in unstructured p2p system," in *Parallel Distributed Processing. IPDPS 2009. IEEE International Symposium on*, May 2009, pp. 1–7. 48, 53, 54

[74] Jianli Hu, Xiaohua Li, Bin Zhou, and Yonghua Li, "A reputation based attack resistant distributed trust management model in p2p networks," in *Third International Symposium on Electronic Commerce and Security (ISECS)*, July 2010, pp. 237–241. 49, 53, 54

[75] Asimina Vasalou, Jeremy Pitt, and Guillaume Piolle, "From theory to practice: forgiveness as a mechanism to repair conflicts in CMC.," in *Proceedings of the 4th International Conference on Trust Management (iTrust 2006)*, Ketil Stølen, William H. Winsborough, Fabio Martinelli, and Fabio Massacci, Eds., Pisa, Italy, May 2006, vol. 3986 of *LNCS*, pp. 397–411, Springer. 51, 72

[76] Junsheng Chang, Huaimin Wang, Gang Yin, and Yangbin Tang, "Icrep: an incentive compatible reputation mechanism for p2p systems," in *WISA'07 Proceedings of the 8th international conference on Information security applications*. 2007, Springer-Verlag Berlin / Heidelberg. 53, 54

[77] Oodes, Krisp, and Müller-Schloer, "On the combination of assertions and virtual prototyping for the design of safety-critical systems," in *ARCS*, Schmeck, Ungerer, and Wolf, Eds. 2002, vol. 2299 of *Lecture Notes in Computer Science*, pp. 195–208, Springer. 58

[78] Richter, Mnif, Branke, Müller-Schloer, and Schmeck, "Towards a generic observer controller architecture for organic computing," in *INFORMATIK 2006 Informatik für Menschen*, 2006, pp. 112–119. 58

[79] Cakar, *Population-Based Runtime Optimisation in Static and Dynamic Environments*, Ph.D. thesis, Leibniz Universität Hannover, 2011. 58, 65, 158

[80] Jorge Casillas, Brian Carse, and Larry Bull, "Fuzzy-xcs: An accuracy-based fuzzy classifier system," *Congreso Espanol Sobre Tecnologias Y Logica Fuzzy*, vol. 12, pp. 369–376, 2004. 59, 61

[81] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini, "Reinforcement learning in continuous action spaces trough sequential monte carlo methods," 2007. 59, 61

[82] Jose Del R. Millan, Daniele Posenato, and Eric Dedieu, "Continuous-action q-learning," *Machine Learning*, vol. 49, pp. 247–265, 2002. 59

[83] William D. Smart and Leslie Pack Kaelbling, "Practical reinforcement learning in continuous spaces," 1999. 59, 61

[84] Stewart W. Wilson, "Zcs: A zeroth level classifier system," *Evolutionary Computation*, vol. 2, pp. 1–18, 1994. 59

[85] Christopher J.C.H. Watkins, "Learning from delayed rewards," 1989. 59

[86] J. Mendel, "Fuzzy logic systems for engineering: a tutorial.," *Proceedings of the IEEE*, vol. 83, pp. 345–377, 1995. 61

[87] Simon Haykin, "Feedforward neural networks: An introduction," 1998. 62

[88] Richard P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, April 1987. 62

[89] Michael J. Wooldridge and Michael Wooldridge, *An Introduction to MultiAgent Systems (2. ed.)*, Wiley, Chichester, UK, 2 edition, 2009. 65

[90] The foundation for intelligent physical agents, "Fipa," . 65

[91] Fabio Bellifemine, A Poggi, and Giovanni Rimassa, *JADE - A FIPA-compliant agent framework*, pp. 97–108, The Practical Application Company Ltd., 1999. 65

[92] Gerrit Anders, Florian Siefert, Nizar Msadek, Rolf Kiefhaber, Oliver Kosak, Wolfgang Reif, and Theo Ungerer, "Temas - a trust-enabling multi-agent system for open environments," Tech. Rep. 2013-04, Informatik, 2013. 66

[93] "Repast simphony," 2013. 66, 77

[94] Michael E. Bratman, "Intention, plans, and practical reason," in *Center for the Study of Language and Information (CSLI) Publications, Stanford, CA, US*, 1999. 66

[95] Adam Cheyer David L. Martin and Douglas B. Moran, "The open agent architecture: A framework for building distributed software systems," in *Applied Artificial Intelligence*, 1999, vol. 13, pp. 91–128. 66

[96] Sanjeev Kumar, "The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams," in *In Proceedings of the Fourth International Conference on Multi-Agent Systems*. 2000, pp. 159–166, IEEE Computer Society. 66

[97] "Cognitive agent architecture (cougaar) open source project," . 66

[98] Sven Tomforde, *An Architectural Framework for Self-configuration and Self-improvement at Runtime*, Ph.D. thesis, Leibniz Universität Hannover, 2011. 66

[99] J. Pitt, D. Ramirez-Cano, M. Draief, and A. Artikis, "Interleaving multi-agent systems and social networks for organized adaptation," *Computational and Mathematical Organization Theory*, vol. 17, no. 4, pp. 344–378, 2011. 68

[100] T. Khopkar, X. Li, and P. Resnick, "Self-selection, slipping, salvaging, slacking, and stoning: the impacts of negative feedback at ebay," in *Conference on Electronic Commerce*. 2005, pp. 223–231, ACM Press. 71

[101] Guido Boella, Leendert van der Torre, and Harko Verhagen, "Introduction to normative multiagent systems," in *Normative Multi-agent Systems*, Guido Boella, Leon van der Torre, and Harko Verhagen, Eds., Dagstuhl, Germany, 2007, number 07122 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. 72

[102] Guido Boella, Gabriella Pigozzi, and Leendert van der Torre, "Normative systems in computer science - ten guidelines for normative multiagent systems," in *Normative Multi-Agent Systems*, Guido Boella, Pablo Noriega, Gabriella Pigozzi, and Harko Verhagen, Eds., Dagstuhl, Germany, 2009, number 09121 in Dagstuhl Seminar Proceedings, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. 72, 73

[103] Inc. (OMG) Object Management Group, "Omg object constraint language (ocl)," Tech. Rep., 2012. 74

[104] Yoonsik Cheon Gary T. Leavens, "Design by contract with jml," 2006. 76

[105] Yvonne Bernard, Lukas Klejnowski, Jörg Hähner, and Christian Müller-Schloer, "Towards trust in desktop grid systems," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 637–642, 2010. 82, 109

[106] Lukas Klejnowski, *Trusted Community: A Novel Multiagent Organisation for Open Distributed Systems*, Ph.D. thesis, Leibniz Universität Hannover, 2014. 87

[107] Eytan Adar and Bernardo A. Huberman, "Free riding on gnutella," *First Monday*, vol. 5, pp. 2000, 2000. 100

[108] Michael W. Macy and John Skvoretz, "The evolution of trust and cooperation between strangers: A computational model," Tech. Rep., Oct. 1998. 104

[109] Y. Bernard, L. Klejnowski, E. Cakar, J. Hähner, and C. Müller-Schloer, "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*, 2011. 108, 152, 209

[110] R. Rojas, *Theorie der neuronalen Netze: Eine systematische Einführung*, Springer, 1996. 110

[111] Yvonne Bernard, Lukas Klejnowski, Ronald Becher, Markus Thimm, Jörg Hähner, and Christian Müller-Schloer, "Grid agent cooperation strategies inspired by game theory," in *4. Workshop Grid-Technologie für den Entwurf technischer Systeme, Dresden, 21.-22. September*, 2011, vol. ISSN 1862-622X. 113

[112] Daniel Hughes, Geoff Coulson, and James Walkerdine, "Free riding on gnutella revisited: The bell tolls?," *IEEE Distributed Systems Online*, vol. 6, no. 6, 2005. 131

[113] I. Eyal and E. G. Sirer, "Majority is not Enough: Bitcoin Mining is Vulnerable," *ArXiv e-prints*, Nov. 2013. 132

[114] Christiano Castelfranchi and Rino Falcone, *Trust Theory: A Socio-Cognitive and Computational Model*, Wiley Publishing, 1st edition, 2010. 132

[115] Kangbok Lee, Joseph Leung, and Michael Pinedo, "Makespan minimization in online scheduling with machine eligibility," *4OR: A Quarterly Journal of Operations Research*, vol. 8, pp. 331–364, 2010. 133

[116] D. Kondo, D.P. Anderson, and J. McLeod, "Performance evaluation of scheduling policies for volunteer computing," in *IEEE International Conference on e-Science and Grid Computing*, Dec. 2007, pp. 415–422. 134, 135

[117] Christopher Seifert, "Analyse und vergleich von adaptiven und trustbasierten desktop grid matchmaking-verfahren," M.S. thesis, Leibniz Universität Hannover, 2012. 141, 144, 211

[118] Huanyu Zhao and Xiaolin Li, "H-trust: A robust and lightweight group reputation system for peer-to-peer desktop grid," in *28th International Conference on Distributed Computing Systems Workshops. ICDCS '08.*, June 2008, pp. 235–240. 143

[119] Hartmut Schmeck, Christian Müller-Schloer, Emre Çakar, Moez Mnif, and Urban Richter, "Adaptivity and self-organization in organic computing systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 5, pp. 10:1–10:32, September 2010. 149, 150

[120] Ashikur Rahman and Pawel Gburzynski, "Hidden problems with the hidden node problem," in *Proceedings of 23rd Biennial Symposium on Communications*, 2006, pp. 270–273. 176

List of Publications

1. J. Kantert, Y. Bernard, L. Klejnowski and C. Müller-Schloer: "Estimation of reward and decision making for trust-adaptive agents in normative environments", in *Proceedings of the 27th International Conference on Architecture of Computing Systems (ARCS 2014)*, 978-3-319-04890-1, Feb. 25–28, 2014.

2. Y. Bernard, L. Klejnowski, D. Bluhm, J. Hähner and C. Müller-Schloer: "Self-organisation and Evolution for Trust-adaptive Grid Computing Agents", in *Evolution, Complexity and Artificial Life, Springer*, ISBN: 978-3-642-37576-7 (Print), 978-3-642-37577-4 (Online), 2014.

3. J. Kantert, L. Klejnowski, Y. Bernard and C. Müller-Schloer: "Influence of Norms on Decision Making in Trusted Desktop Grid Systems - Making Norms Explicit, Poster at ICAART 2014, in *Proceedings of the 6th International Conference on Agents and Artificial Intelligence*, March 6–8, 2014.

4. Y. Bernard, J. Kantert, L. Klejnowski, N. Schreiber and C. Müller-Schloer: "Application of learning to trust-adaptive agents", Workshop on Social Concepts in Self-Adaptive and Self-Organising Systems, in *Proceedings of the Seventh IEEE International Conference on Self-Adaptive and Self-Organising Systems Workshop (SASOW)*, IEEE, Philadelphia, USA; Sept. 9–13, 2013.

5. J. Kantert, Y. Bernard, L. Klejnowski and C. Muller-Schloer: "Interactive Graph View of explicit Trusted Communities in an open Trusted Desktop Grid System", 2013 Demo Entry, in *Seventh IEEE International Conference on Self-Adaptive and Self-Organising Systems (SASO)*, IEEE, Philadelphia, USA; Sept. 9–13, 2013.

6. G. Anders, J.-P. Steghöfer, L. Klejnowski, M. Wissner, S. Hammer, F. Siefert, H. Seebach, Y. Bernard, W. Reif, E. André and C. Müller-Schloer: "Reference Architectures for Trustworthy Energy Management, Desktop Grid Computing

Applications, and Ubiqitous Display Environments", *Technical Report 2013–05*, Universitätsbibliothek der Universität Augsburg, Universitätsstr. 22, 86159 Augsburg.

7. L. Klejnowski, S. Niemann, Y. Bernard and C. Müller-Schloer: "Using Trusted Communities to improve the speedup of agents in a Desktop Grid System", in *Proceedings of the 7th International Symposium on Intelligent Distributed Computing - IDC 2013*, Prague, Czech Republic, Springer, vol. 511, ISBN: 978-3-319-01570-5.

8. L. Klejnowski, Y. Bernard, G. Anders, C. Müller-Schloer and W. Reif: "Trusted Community - A Trust-Based Multi-Agent Organisation for Open Systems", in *ICAART 2013 - Proceedings of the 5th International Conference on Agents and Artificial Intelligence*, Barcelona, Spain, 15–18 Feb., 2013.

9. M. Pacher, C. Müller-Schloer, Y. Bernard and L. Klejnowski: "Social Awareness in Technical Systems, The Computer After Me", *Imperial College Press/-World Scientific Book*.

10. L. Klejnowski, Y. Bernard, C. Müller-Schloer and J. Hähner: "Using Trust to reduce wasteful computation in open Desktop Grid Systems", TSOS 2012, in *Proceedings of the 10th Annual Conference on Privacy, Security and Trust*, ISBN: 978-1-4673-2323-9, IEEE 2012, pp. 250–255.

11. Y. Bernard, L. Klejnowski, C. Müller-Schloer, J. Pitt and J. Schaumeier: "Enduring Institutions and Self-Organising Trust-Adaptive Systems for an Open Grid Computing Infrastructure", 2nd Awareness Workshop, in *Proceedings of the 2012 IEEE International Conference on Self-Adaptive and Self-Organising Systems Workshop (SASOW 2012)*, IEEE, pp. 163–168, ISBN: 978-1-4673-5153-9.

12. Y. Bernard, L. Klejnowski, D. Bluhm, J. Hähner and C. Müller-Schloer: "An Evolutionary Approach to Grid Computing Agents", in *Proceedings of the Italian Workshop on Artificial Life and Evolutionary Computation WIVACE 2012*, pp. 1–12, ISBN: 978-88-903581-2-8.

13. Y. Bernard, L. Klejnowski, Emre Cakar, J. Hähner and C. Müller-Schloer: "Efficiency and robustness using Trusted Communities in a Trusted Desktop Grid", in *Proceedings of the 2011 Fifth IEEE International Conference on Self-*

Adaptive and Self-Organising Systems Workshop (SASOW 2011), IEEE, October 2011, pp. 21–26, ISBN 978-1-4577-2029-1.

14. Y. Bernard, L. Klejnowski, R. Becher, M. Thimm, J. Hähner and C. Müller-Schloer: "Grid agent cooperation strategies inspired by Game Theory", in *Proceedings 4. Workshop Grid-Technologie für den Entwurf technischer Systeme Grid4TS*, Dresden, ISSN 1862-622X, September 21–22, 2011.

15. J.-P. Steghöfer, F. Nafz, W. Reif, Y. Bernard, L. Klejnowski, J. Hähner and C. Müller-Schloer: "Formal Specification and Analysis of Trusted Communities", in *Proceedings of the 2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organising Systems Workshop (SASOW 2010)*, IEEE; September 2010, pp. 190–195.

16. J.-P. Steghöfer, R. Kiefhaber, Karin Leichtenstern, Y. Bernard, L. Klejnowski, W. Reif, T. Ungerer, E. André, J. Hähner and C. Müller-Schloer: "Trustworthy Organic Computing Systems: Challenges and Perspectives", Proceedings of the 7th International Conference on Autonomic and Trusted Computing (ATC 2010), Springer, ISBN: 978-364216575-7, pp. 62-76.

17. L. Klejnowski, Y. Bernard, J. Hähner and C. Müller-Schloer: "An architecture for Trust-adaptive Agents", Proceedings of the 2010 Fourth IEEE International Conference on Self-adaptive and Self-organising Systems Workshop (SASOW 2010), IEEE, pp. 178-183, ISBN 978-1-4244-8684-7.

18. Y. Bernard, L. Klejnowski, J. Hähner and C. Müller-Schloer: "Towards Trust in Desktop Grid Systems", ccgrid, pp. 637–642, 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE Computer Society.

19. Sven Tomforde, M. Hoffmann, Y. Bernard, L. Klejnowski and J. Hähner: "POWEA: A System for Automated Network Protocol Parameter Optimisation Using Evolutionary Algorithms", Beiträge der 39. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 2009, pp. 3177–3192.

20. M. Hoffmann, M. Wittke, Y. Bernard, R. Soleymani and J. Hähner: "DMCtrac: Distributed Multi Camera Tracking", ICDSC 2008. Second ACM/IEEE International Conference on Distributed Smart Cameras.

Appendix A

Appendix

A.1 UML Diagrams

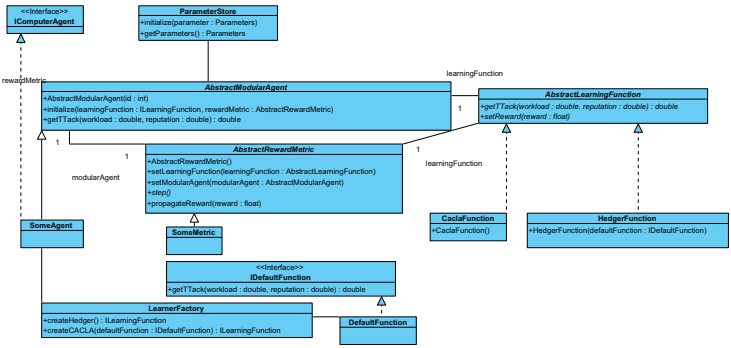


Figure A.1: The Learning agent framework enabling system designers to exchange the learning algorithm of trust-adaptive learning agents

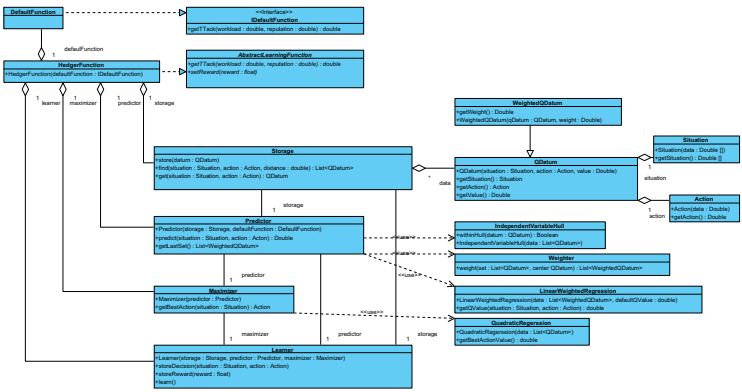


Figure A.2: The Hedger implementation based on learning agent framework

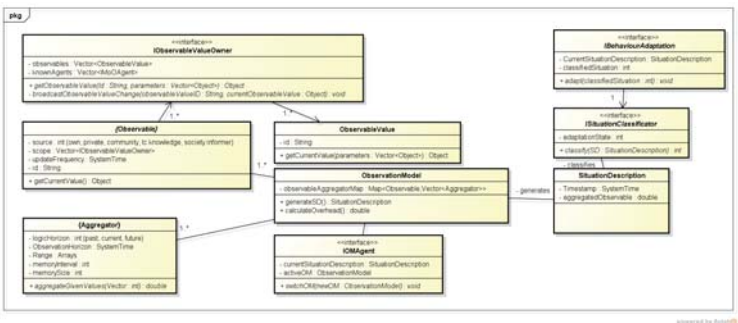


Figure A.3: UML representation of Adaptive Observation Model

A.2 Further Evaluation results

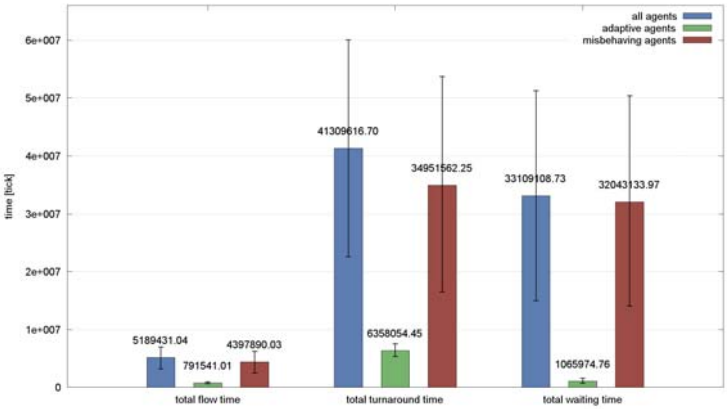


Figure A.4: Further performance results of trust-adaptive iTC agents. Misbehaving agents reach severely higher flow times, turnaround times and waiting times, have therefore no benefit from participating in the system and are thus excluded from the TC.

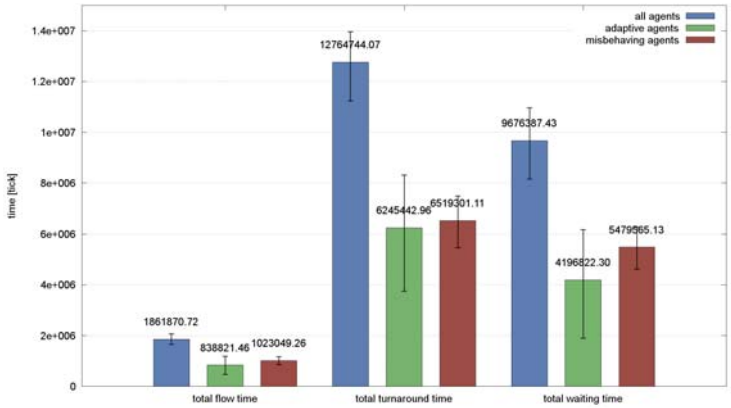


Figure A.5: Further performance results of tactical agents without workload prediction.

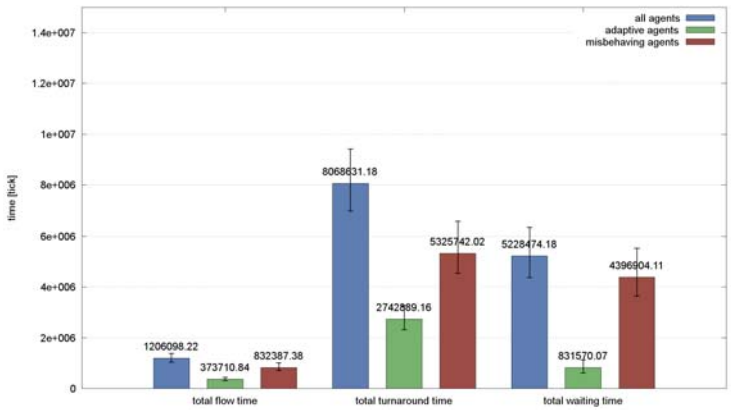


Figure A.6: Further performance results of tactical agents with workload prediction. The performance has been improved compared to the results in Figure A.5. The flow times, turnaround times and waiting times of tactical adaptive agents have been decreased, which means that agents reach a better usage of the system if they adapt proactively to workload predictions.

A.3 Parameter settings

A.3.1 General simulation parameters

The general parameters of the experiment are listed in Table A.1.

Parameter	Value
min_work_unit_number_for_job	10
max_work_unit_number_for_job	15
min_computing_costs_for_workunit [Tick]	450
max_computing_costs_for_workunit [Tick]	650
min_job_generation_pause [Tick]	2000
max_job_generation_pause [Tick]	4000
throughput_Interval [Tick]	10.000
Job Generation Length [Tick]	50.000
Simulation length [Tick]	200.000
Number of agents	100

Table A.1: General simulation parameters

The number of work units for each generated job is randomly chosen in the interval between 10 and 15. The computing costs (number of computation ticks) for each work unit are uniformly distributed between 450 and 650. The computation time on an agent is then computed by dividing the computing costs by the Performance level of the agent, which in this experiment has been randomly initialised between 2 and 5. For instance, a work unit with 500 ticks computing costs which is computed on an agent with Performance level 5 needs 100 ticks computation time.

The interval, in which the next job is generated, is uniformly distributed between 2000 and 4000 ticks. This interval can be varied in order to increase or decrease the workload of the system. The throughput interval is used to define the interval in which the throughput is evaluated.

The last parameters define the maximal time step at which new jobs in the system are generated (Job Generation Length [Tick]), the simulation length and the number of agents in the simulation.

A.3.2 TDG parameters

The results of the TDG runs have been conducted with the tdg specific parameters listed in Table A.2. Basically, these are responsible to parameterise the worker and submitter thresholds of the iTC agent decision mechanism 6.8.1.

Parameter	Value
ada_work_own_high_rep	0.3
ada_work_own_low_rep	-0.2
ada_work_others_high_aggTL	0.2
ada_work_others_med_aggTL	-0.01
ada_work_others_low_aggTL	-0.3
ada_sub_min_trust_req	-0.3
ada_sub_med_trust_req	-0.005
ada_sub_high_trust_req	0.3
ada_workload_low_med	0.05
ada_workload_med_high	0.3

Table A.2: TDG parameters

The parameters *ada_work_own_high_rep* and *ada_work_own_low_rep* define the TT^acc boundaries for the low, medium and high classification. The agent decisions whether or not to accept a work unit are based on its own reputation and the workload in the system defined by *ada_workload_low_med* and *ada_workload_med_high* as low, medium or high.

The trust value of an agent in submitter role is also discretised into low, medium and high, which is defined by *ada_work_others_low_aggTL*, *ada_work_others_med_aggTL* and *ada_work_others_high_aggTL*.

The sampling in submitter role is defined using *ada_workload_low_med* and *ada_workload_med_high* for work and *ada_sub_high_trust_req*, *ada_sub_med_trust_req* and *ada_sub_min_trust_req* for trust thresholds [109].

A.3.3 H-trust parameters

In our experiments, we used the parameters given in [25] where these were defined. The parameters of our experiments are given in Table A.3.

Parameter	Value
selectionThreshold	10.0
queryThreshold	7
queryPercentage	0.1
initialTrustRatingMean	75.0
initialTrustRatingStdDev	10.0
initialCredibilityFactor	5.0
minimumCredibilityRating	1
maximumCredibilityRating	10
maximumServiceHistoryTableSize	30
localTrustUpdatePeriod	1.000
incrementalQualityRating	0
performanceLevelThreshold	-1
initialtrustcount	1
probabilityofinitialTrustvalue	0.0

Table A.3: Simulation parameters of H-Trust

Parameter	Wert
initialFriendListSizePercentage	1.0
resultBurst-BurstSize	3
resultBurstIntervalsCount	3
childPropagation	1
initialworkunitrequestsize	1
maximumchildren	5
maximumpotentialchildren	5
bestPerformingChildPropagationPeriod	2.000
oldChildrenTimeoutInterval	1.000

Table A.4: Simulation parameters of Organic Grid

A.3.4 Organic Grid parameters

We used the parameters listed in A.4, which we either derived from literature or optimised in our simulation [117] in order to reach the best performance.

A.3.5 Disturbance parameters

Table A.5 lists the parameters used for the disturbance situation for the comparison of H-Trust, Organic Grid and the TDG.

Parameter	Wert
min_work_unit_number_for_job	10
max_work_unit_number_for_job	15
min_job_generation_pause [Tick]	2000
max_job_generation_pause [Tick]	4000
min_computing_costs_for_workunit [Tick]	450
max_computing_costs_for_workunit [Tick]	650
throughput_Interval [Tick]	10.000
Job Generation Length [Tick]	100.000
Simulation Length [Tick]	200.000
Adaptive Agents	70
Egoists	15
Free Riders	15
probability for egoist WU abort	0.9

Table A.5: Simulation parameters for disturbance experiments

Curriculum Vitae

Personal Information

Name:	**Yvonne Bernard**
Date of Birth:	March 28, 1982
Place of Birth:	Hannover
Nationality:	German

Education

10/2005-06/2008	**Degree in Computer Science** at the Leibniz University in Hannover, Germany (M. Sc.), Thesis: "Distributed Object Tracking in Smart Camera Networks"
10/2001-10/2005	**Degree in Computer Science** at the Leibniz University in Hannover, Germany (B. Sc.), Thesis: "Implementierung einer Testumgebung mit Emergenzeigenschaften anhand eines agentenbasierten Simulators"
08/1994-06/2001	**Grammar school** (Gymnasium Burgdorf), Burgdorf; Abitur 2001 (A-levels)

Experience

06/2008 - 12/2013 **Research Assistant at Institut für Systems Engineering, Fachgebiet System- und Rechnerarchitektur**

- Research project "OC-Trust": Development of Trustworthy Multiagent Systems
- Tutorial Parallel Computing, Tutorial Organic Computing, Seminar Complex Systems

11/2003 – 05/2008 **Student Assistant at Institut für Mensch-Maschine-Kommunikation, Fachgebiet Graphische Datenverarbeitung**

- Tutorial Datastructures and Algorithms, Conference Management, 3D Workshops

10/2006 – 03/2007 **Student Assistant at Institut für Kartografie und Geoinformatik**

- Programming of graphic rectification (C++, Qt)

06/2002 – 08/2002 **Student Assistant in Project "Mentoring für Schülerinnen"**

Languages

German	**Native language**
English	**Fluent both orally and in writing**
French	**Basic knowledge**