Opening the Black Box of Trust: Reasoning about Trust Models in a BDI Agent

ANDREW KOSTER,
Artificial Intelligence Research Institute IIIA-CSIC, Bellaterra, Catalonia, Spain
e-mail: andrew@iiia.csic.es
Universitat Autònoma de Barcelona, Bellaterra, Catalonia, Spain

MARCO SCHORLEMMER,
Artificial Intelligence Research Institute IIIA-CSIC, Bellaterra, Catalonia, Spain
e-mail: marco@iiia.csic.es
Universitat Autònoma de Barcelona, Bellaterra, Catalonia, Spain

JORDI SABATER-MIR,
Artificial Intelligence Research Institute IIIA-CSIC, Bellaterra, Catalonia, Spain
e-mail: jsabater@iiia.csic.es

Abstract

Trust models as thus far described in the literature can be seen as a monolithic structure: a trust model is provided with a variety of inputs and the model performs calculations, resulting in a trust evaluation as output. The agent has no direct method of adapting its trust model to its needs in a given context. In this article we propose a first step in allowing an agent to reason about its trust model, by providing a method for incorporating a computational trust model into the cognitive architecture of the agent. By reasoning about the factors that influence the trust calculation the agent can effect changes in the computational process, thus proactively adapting its trust model. We give a declarative formalization of this system using a multi-context system and we show that three contemporary trust models, BRS, ReGReT and ForTrust can be incorporated into a BDI reasoning system using our framework.

Keywords: trust, BDI, multi-context systems.

1 Introduction

Research into trust models is currently an active topic in the domain of multi-agent systems (MAS). Many computational trust models have been proposed [27], based on theoretical foundations from many different disciplines. For example, some trust models have cognitive foundations [31, 14, 8], others are based on mathematical methods, such as statistical models [36, 37] or game theory [2] and still others use a social network oriented approach [11] or are oriented towards specific applications, such as negotiation [35] or the semantic web [34]. These trust models have in common that they are computational methods for calculating an agent's trust in a trustee based on the agent's own interactions with the trustee, as well as on information that is available in the environment about the trustee. Such information may be direct communications from other agents in the system, giving their own trust evaluations of the trustee; it may be reputation information; or it may be any other source of information available in the system. The

trust model then aggregates this information, using the chosen mathematical method, and calculates the evaluation of the trustee.

The problem with the trust models discussed so far in the literature is that an agent is unable to change its trust model if it discerns a change in the environment. If we were to view the trust model from the agent's perspective, it would appear to be a "black box" with as input the various information sources and as output an evaluation of how trustworthy the trustee is. However, as argued in [4], trust is not just an evaluation of a trustee, but an integral part of the decision making process of an agent in a social environment. For a trust evaluation to be meaningful in this process it may be necessary to customize the evaluation process to the decision that is being made. This is especially so in an open MAS, where the environment may change.

As an example, consider the following situation. An agent has the task of routinely buying items on an electronic marketplace and to decide, by using its trust model, which seller to interact with. In general the agent's owner requires the agent to buy items as cheaply as possible and does not mind if, to achieve this, the item is delivered late. However, one day the agent is assigned specifically to buy an item prioritizing speedy delivery. This is problematic if the agent's trust model is hard coded to disregard delivery speed when evaluating salesmen. Traditionally, evaluating single interactions is not considered to be a part of a trust model, which is generally defined as an aggregation method to determine the trust evaluation of an agent based on various different information sources. However, at this level the same thing can happen. For instance, if the agent's environment contains mostly truthful agents it will be able to use reputation information. However, if the environment changes and most agents are liars, the agent using reputation information is misguided. Some models are equipped to deal with these changes, but still do so reactively [36, 37]. If the agent *knows* the environment has changed it cannot proactively adapt its trust model. Such issues arise at all levels during trust computation and can be triggered by changes in the beliefs and the goals of an agent. Contemporary trust models are not equipped to deal with this type of proactive adaptation, in which the agent's goals and beliefs can precipitate a change in the way an agent calculates its trust evaluations. Furthermore, because trust models are treated as a black box, their integration into a cognitive agent also does not allow for proactive adaptation.

In this paper we present an agent model capable of reasoning about trust and proactively changing its trust model to reflect this reasoning. We do not present a new trust model, but rather propose an extended BDI framework designed to work with existing models. We provide a method for integrating computational trust models into a cognitive agent model. We do this by considering the trust model in as declarative a way as possible, while still relying on the underlying computational process for calculating trust evaluations. While this does not provide the agent with introspection into the actual computational mechanism its trust model uses, it opens the black box of trust sufficiently to allow the agent to proactively adapt its model. Because of the way that trust is fitted into the multi-context model, it is possible to plug in different trust models quite easily.

In Section 3 we present our abstract view of what a trust model is and the properties we assume a computational trust model has. This is needed to understand how we incorporate trust models into an extended BDI framework, which forms the main body of this paper. In Section 4 we present the various logics we use and the basic BDI framework we extend. Section 5 describes the first part of this extension: a manner of specifying a trust model to allow the agent to reason about it, while Section 6 introduces the mechanics for performing this reasoning. Finally we demonstrate how the system allows an agent to reason about its trust model, using three of the best known contemporary trust models, in Section 7.

2 Background

The idea of integrating the trust model into the cognitive process of an agent is not new. There are various different methods for representing this cognitive process, but the integration of trust into this process is done mainly for BDI agents. BDI stands for Beliefs-Desires-Intentions and is a logical model

for designing intelligent agents [28]. The beliefs represent the agent's knowledge about its environment, the desires the state of the environment the agent desires to achieve and the intentions represent the plans it intends to perform to achieve its desires. The BDI model has met with considerable success and many systems for implementing intelligent agents follow this model to some degree or another [1, 6]. From the point of view of trust, it offers clear advantages: by providing a crisp definition of the agent's beliefs and its goals, the role trust plays can be made explicit, incorporating it into the logical framework. There are different ways of doing so, and thus trust is treated differently by different models.

The work that deals with an integration of trust into a BDI-agent can be divided roughly into two different approaches. The first is computational trust models that, for the most part, take a belief-centric approach to trust. These allow for direct integration into the agent. The second approach is a more logic-oriented approach and these works provide formal logics for reasoning about trust, although they do not go into detail on how a computational model should compute these evaluations.

2.1 Cognitive Computational Trust Models

Castelfranchi and Falcone [4] describe the socio-cognitive underpinnings of trust and show its functioning through experimentation using a computational model. This is a very belief-centric model in which trustworthiness is treated as a belief with a specific method for updating it. Decisions are taken in a normal manner, choosing whom to interact with based on the agent's beliefs. The belief base contains trust evaluations as well as the intermediate beliefs resulting from the computational process. Because the trust evaluations and decision process are all based upon the belief base, adapting the model to a changing environment, for which the evidence will also be collected in the agent's belief base, seems possible, however this step is not addressed in [4]. This model is one of the inspirations for our work, and the two should be largely compatible: Castelfranchi and Falcone's model could serve as the adaptive trust model, while the work we present here allows for explicit, proactive adaptation. Burnett et al. take a very different approach and emphasize trust as a tool in fulfilling plans [2]. They assume agents need to delegate tasks and discuss the role trust and reputation play in deciding whom an agent should optimally delegate tasks to. They assume different agents will perform differently at the various tasks and thus agents' trustworthiness depends partially on what task they need to perform. This is quite similar to the idea of role-based trust, but is integrated into the decision-making process of the agent. This is somewhat similar to the approach we take, in which the goal and plan of the agent can influence the computation of the trustworthiness of an agent, but their approach focuses purely on the decision of delegating and do not deal with other cognitive aspects of trust. Specifically they do not use the beliefs of an agent.

BDI+Repage [25] attempts to take a more comprehensive approach, by integrating the Repage reputation model [31] into the cognitive process of a BDI agent. While this is an interesting approach, it leaves the Repage model as a black box; specifying only how Repage should be connected to the various inputs and how the outputs of the calculation should be dealt with in the belief base. As such it does not allow for reasoning about how to evaluate reputation, just about what to do with the evaluations after Repage has calculated them. This system presents the basic idea we build upon: it uses a multi-context system [10] to represent the BDI agent and reputation model, which provides a clear way of integrating a trust or reputation model into the cognitive agent architecture. However, it does not provide the mechanism needed to actually reason about the trust model and represents the trust computation in a single, monolithic context. We propose a more sophisticated method of representing the trust model in a BDI-agent which allows an agent to reason about and adapt its trust model.

2.2 Logics for Reasoning about Trust

Liau presented a modal logic for explicitly reasoning about trust [19]. The modal logic used is specifically intended for reasoning about beliefs and information and shows how interesting and desirable properties of trust emerge from basic axioms about the logic. Dastani et al. [7] extend the logic to deal with topical

trust, leading to the logic being able to infer trust evaluations about specific topics, or tasks. However, these logical models do not give a definition of the computational process. Trust is an abstract concept and, if certain axioms about it are true, then other properties can be proved. This makes it possible to show the consequences of specific types of trust, such as if trust is transitive, or symmetric. However, it says nothing about whether actual methods of computing trust fulfill such properties. An approach that attempts to rectify this somewhat is ForTrust [20]. This logic integrates trust into a BDI logic and, while using a different logic, also allows for the proving of certain properties of trust from the basic axioms of the logic. However, in this case, the axioms used are quite practically oriented and there are implementations of ForTrust [14, 18]. We show that these implementations can also be modeled within our framework in Section 7.2.

An entirely different approach to reasoning about trust is taken by Parsons et al. [24], who use argumentation as their method for reasoning about information and integrate trust into the argumentation framework, in order to decide whether or not an argument is acceptable. The work then describes some axioms of argumentation and trust to prove desirable properties of the reasoning, similar to Liau's approach. These logic-based approaches differ significantly from our approach, in which we assume a computational trust model is given and provide a methodology for adapting this computational method. Specifically our aim is to give a declarative description of a computational trust model and allow the agent to reason about this: we take a bottom-up approach, starting with a computational trust model, rather than a top-down approach in which desirable properties are derived, regardless of whether these are based on realistic assumptions or not.

3 Trust Models

In this paper we focus on how an individual agent can reason about its trust model and as such we require the agent to have access to its own trust model. *Centralized* reputation systems, such as eBay [22] or SPORAS [40] are not considered within this paper, despite specifying a function for calculating the trust evaluation, because they do not allow the agent to change the way this calculation is done and thus reasoning about it is pointless. This is not to say that we disregard all reputation mechanisms: those that do not rely on a centralized approach, but are designed so reputation is calculated in a distributed manner, such as Yu & Singh's model [39], can be incorporated into a reasoning system. In this approach we coincide with the view adopted by Pinyol [26], who argues that the difference between computational trust and reputation models is not clear, and that many reputation models in fact give an evaluation of agents that can equally well be interpreted as a trust evaluation.

3.1 Algorithmic Nature of Trust Models

When talking about trust in a multi-agent system, the first thing to note is that we are talking about trust in *computational* entities and not humans. As a result, any model of trust we take under consideration should be a computational model. With this we mean that the description of the trust model must define an *algorithm* for obtaining a trust evaluation from a set of inputs. An algorithm is a computational method for computing the output of a *partial function*: given an input in the function's domain for which the function is defined, the algorithm computes a unique output in the function's range. In the most abstract sense, a trust model is thus a *Turing computable function*.

PROPERTY 1 (Functional nature of trust models):
A trust model, as we understand it, is a computational process defined by a Turing machine and as such is a method for calculating the output of a *function*.

3.2 Parametrization

The goal of this paper is to allow the agent to reason about its trust model. As such, the above criteria that the model is a Turing computable function is not enough. The agent needs some way of reasoning and *adapting* the function. However in this work we are not interested in adapting the algebraic operations used, but rather, within the limitations of the algebraic operations, adapt the available parameters. For example, if the trust model uses a weighted average to calculate the output, we do not wish to replace this operation: the model should use the weighted average. However, what we might want to adapt are the weights it uses.

In general, this description begs the question of which parameters can be adapted. To this question we can answer that it very much depends on the actual trust model used, but an example that recurs in many trust models is that some parameter(s) defines the importance of different *sources of information*, such as direct and indirect experiences [4]. We require a function representing a trust model to not just be any function, but a *parametric* function with at least one parameter. Such parameters are, for instance, the weights in a weighted average, the decay rate at which older evidence is discarded, or the distance in a social network at which point to disregard opinions.

Many trust models use a number of different parameters and each of these parameters influences the effect different variables have on the final trust evaluation. We therefore say that the value of a parameter should be "governed" by the importance the agent wants to assign to these variables: the more important the variable, the larger its influence should be upon the output of the trust model. For instance, consider a trust model that has a parameter which governs the relative importance of direct experiences and communicated, indirect, experiences. If the agent believes it is surrounded by liars it should give little importance to communicated, indirect, experiences. The value assigned to the parameter should reflect this, ensuring that direct experiences are given more importance in the trust calculation than communicated ones. In general, we require that the agent is capable of instantiating a trust model with different values for its parameters, which reflect the importance the agent assigns to the different factors that play a role in calculating trust evaluations.

A further requirement is that for any combination of values for the parameters, the function is valid: the parameters are linked to the importance of different factors taken into account in calculating trust. Thus if some factor becomes more important this can precipitate a change in the parameters. Any such change should result in a valid trust model.

> PROPERTY 2 (Parametric models of trust):
> We only consider trust models that have at least one parameter such that a change in the parameter changes the output of the trust model in a predictable manner. Any combination of values for a trust model's parameters describes a valid trust model.

3.3 Functioning of Trust Models

All trust models are methods of aggregation: they combine and merge information from several different sources into a single value. It is possible to distinguish two separate parts in this process: the first part in this process is to combine different aspects of information from a single source into a single value and the second part consists of several stages of combining many values representing different sources into, eventually, a single evaluation. Intuitively we see that parameters in the first part *mean* something different to parameters in the second part: in the former case a parameter specifies which aspects of a piece of information obtained from a single source are more important to the agent, whereas in the latter a parameter specifies which sources of information are more important. In an example of wishing to buy items at an auction, the first part may for instance evaluate how happy an agent is with the outcome of an auction and a parameter might encode that the cost of an item is more important than its delivery time when performing this evaluation. In the second part we could aggregate the evaluations of single auction

transactions into a final evaluation and have a parameter which encodes that the more expensive an item is, the more important it is in this aggregation.

We see that, while most literature on trust models describes the second part in great detail, the first part is often left open. This is justifiable as this part is very domain dependent. However, we see that even those trust models which completely define the aggregation of evaluations from various sources of information without any explicit parameters, for example Vogiatzis et al.'s model [37], can be adapted by our proposed reasoning model if the implementation of the first part, evaluating independent interactions, is parametrized. Furthermore there may be *implicit* parameters in the way the designers make certain choices. For instance Vogiatzis et al. state that "in the absence of direct experience of a service provider that leads to direct trust, it is essential to be able to enquire about that service provider's reputation", thereby implying that reputation information should only be used if the agent has had no direct contact with the service provider. However, this is just one possibility for combining reputation and direct trust: the cut-off for reputation could be placed at an arbitrary value of direct experiences. There are many other methods proposed for the combination of direct trust and reputation, such as ReGReT's weighted average [30] or Falcone et al.'s belief-based approach to combining different sources [8].

Another important aspect of trust, which is not dealt with by all computational models, is that trust is inherently multi-faceted. An agent is evaluated with respect to some role, or action, which it is expected to perform. The evaluating agent requires this action to be performed to achieve a specific goal, which is why it requires the trust evaluation in the first place. This goal, and the action the trustee is required to perform to achieve it, can change the parameters of the trust model: an agent selling an item in an auction may be evaluated differently to an agent buying an item.

The cognitive framework we present in this paper is designed in such a manner that a large number of current computational trust models can be incorporated into a BDI reasoning system by considering the parameters they take into account and specifying what factors can influence these parameters. The system described here allows these factors to influence the trust model automatically, thus integrating the model into the reasoning system of the agent. Furthermore it is inherently multi-faceted, as the entire cognitive stance of the agent can influence the trust model. We illustrate how to incorporate existing trust models in Section 7, in which we demonstrate how the framework allows for the incorporation of three contemporary trust models.

4 Preliminaries

In this paper, we define a method for reasoning about trust models. However for this we will need to refer to some of the other cognitive entities in the system. Accordingly, we start by introducing the multi-context representation of a BDI-agent. The specification of an agent using a multi-context system (MCS) has several advantages for both the logical specification and the modeling of the agent architecture [23]. From a software engineering perspective, an MCS supports modular architectures and encapsulation. From a logical modeling perspective, it allows the construction of agents with different and well-defined logics, keeping all formulae of the same logic in their corresponding context. This increases the representational power of logical agents considerably and, at the same time, simplifies their conceptualization. The MCS paradigm is therefore a popular formalism for extending the basic BDI logic. For instance, Criado et al. [5] and Joseph et al. [16] use an MCS to allow a BDI agent to reason about norms and Pinyol and Sabater [25] use an MCS to incorporate trust into a BDI agent. We follow a similar approach to this last work, but, as explained in Section 2, their approach firstly only deals with Repage and secondly does not allow for the adaptation of the model: their integration focuses on making decisions to interact, based on trust. Finally, Sabater et al. [32] show how the specification of BDI-agents using an MCS corresponds directly with the concept of modules in software engineering, allowing for rapid and easy design of an executable agent architecture.

4.1 Multi-context Systems

Multi-context Systems (MCS) provide a framework to allow several distinct theoretical components to be specified together, with a mechanism to relate these components [10]. An MCS consists of a family of contexts, which are logically connected by a set of bridge rules. Each context contains a formal language and a theory of that language. We say a sentence is in a context if it is a logical consequence of the theory in that context. Bridge rules serve to relate theories between contexts and can be seen as inference rules, but rather than inferring conclusions within a context, the premises and conclusion are in different contexts. They have the form:

$$\frac{C_1 : X; C_2 : Y}{C_3 : Z}$$

where X, Y and Z are *schemas* for sentences in their respective contexts. The meaning of a bridge rule is that if a sentence complying with schema X holds in context C_1 and a sentence with schema Y holds in context C_2 then the corresponding sentence with schema Z holds in context C_3. This is true in the logical sense, but the bridge rules have a second use, which is representing the operational procedures in the system. The schema in the conclusion might be the outcome of some operation. In such cases we will give an abstract description of the function that performs this operation.

Let I be the set of context names, then an MCS is formalized as: $\langle \{C_i\}_{i \in I}, \Delta_{br} \rangle$ with contexts C_i and Δ_{br} a set of bridge rules. In Section 4.3 we show how to represent a BDI-agent as an MCS, but first we introduce the logics we require in our integration of a trust model into a BDI system.

4.2 Logics

In an MCS, each context is specified using a logic. In this section we present the various different logics that we use in the contexts.

4.2.1 First-order logic (FOL)

While we do not deviate from the standard definition of first-order predicate logic, we need to introduce the notation, which is used throughout this paper. The syntax of FOL is defined using a set of constants $\mathcal{C}$, a set of function symbols $\mathcal{F}$, and a set of predicate symbols $\mathcal{P}$, each function and predicate symbol with a fixed arity greater than one. Terms and formulae are given in the usual manner, using quantifiers $\forall, \exists$ and connectives $\neg, \wedge, \vee, \rightarrow$.

4.2.2 First-order dynamic logic (FODL)

We use first-order dynamic logic (FODL), as first proposed by Harel [12]. FODL is first-order logic that is extended by adding action modalities to it. We use FOL as defined above and any FOL-wff is a *program-free* FODL-wff. We now define the set RG of first-order regular programs and the set of FODL-wffs by simultaneous induction as follows:

- For any variable x and term e, $x := e$ is in RG.
- For any program-free FODL-wff φ, $\varphi?$ is in RG.
- For any α and β in RG, $(\alpha; \beta)$, $(\alpha \cup \beta)$ and α^* are in RG.
- Any FOL-wff is an FODL-wff.
- For any FODL-wffs φ and ψ, α in RG and variable x the following are FODL-wffs:
  - $\neg\varphi$

- $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$
- $\forall x : \varphi$, $\exists x : \varphi$
- $[\alpha]\varphi$

Programs formed by the application of the first and second rule are called *assignments* and *tests* respectively. We will refer to both of these as *basic actions* while any other type of program is a *composite* program. For the formal semantics of FODL we refer to [12]. For this article it is sufficient to have an intuitive understanding: the semantics of *program-free* formulae is the same as for standard FOL. The semantics of $[\alpha]\varphi$ can be summarized as: after performing α, φ holds. $\alpha;\beta$ is sequential composition: $[\alpha;\beta]\varphi$ means that after first performing α and then performing β, φ holds. Similarly $\alpha \cup \beta$ is non-deterministic choice: $[\alpha \cup \beta]\varphi$ means that after either performing α or performing β, φ holds; and α^* is repetition: $[\alpha^*]\varphi$ means that after performing α any finite number of times (including none), φ holds.

4.2.3 Priority Logic (PL)

In our specification of the trust model we will require a logic to formalize a structure of priorities between predicates in our MCS. We call this logic Priority Logic (PL)[1]. It is a subset of first-order logic, following the system defined by Schorlemmer [33] for a logic of binary relations. PL limits the predicates to two predicates of relations, denoted $\succ$ and $=$. As is customary, we use infix notation for these predicates. For any two constants a and b, $a \succ b$ and $a = b$ are PL-atoms. Using this we define PL-wffs as:

- Any PL-atom π is a PL-wff.
- For any PL-atom π, $\neg\pi$ is a PL-wff.
- For any PL-wffs π and ξ, $\pi \wedge \xi$ is a PL-wff.

The semantics are given by standard FOL-semantics, but with the following axiom schemas for the predicates, for any constants a and b:

$\succ$**-irreflexivity:** $\neg(a \succ a)$
$\succ$**-transitivity:** $(a \succ b) \wedge (b \succ c) \rightarrow (a \succ c)$
$=$**-reflexivity:** $a = a$
$=$**-symmetry:** $(a = b) \rightarrow (b = a)$
$=$**-transitivity:** $(a = b) \wedge (b = c) \rightarrow (a = c)$

4.3 Multi-context Representation of a BDI-Agent

With all the logics described, it is time to present our specification of a BDI-agent using an MCS. We will not yet define the contexts used for reasoning about the trust model: in this section we present the contexts and bridge rules for a BDI-agent, whereas Sections 5 and 6 extend this framework to include reasoning about trust. A BDI-agent is defined as an MCS: $A = \langle\{BC, DC, IC, PC, XC\}, \Delta_{br}\rangle$. We will first describe the contexts, before defining the bridge rules. The first three contexts correspond with the classical notions of beliefs, desires and intentions as specified by Rao & Georgeff [28]. The first, the Belief Context (BC) contains the belief base of the agent. We use an FODL language, as described in the previous section, to represent beliefs. Let $\mathcal{P}_{Domain}$, $\mathcal{F}_{Domain}$ and $\mathcal{C}_{Domain}$ be the predicates, functions and constants required to describe the agent and its domain, then $\mathcal{L}_{Bel}$ is the FODL language generated from them. If $\varphi \in \mathcal{L}_{Bel}$ is in the belief context, this means the agent believes φ holds. In the Desire Context (DC) we use an FOL language $\mathcal{L}_{Des}$ to represent the agent's desire base, generated from the same set of predicates, functions and constants as the belief base. The agent's desires are thus represented in the program-free segment of the logic for the belief base. If $\psi \in \mathcal{L}_{Des}$ is in the desire

[1] Note that this logic is not related to Wang et al.'s logic [38] with the same name.

context, then the agent desires ψ. The Intention Context (IC) holds the intention base of the agent and uses the FODL language $\mathcal{L}_{Int}$, a subset of the language that the belief context uses: let $\alpha \in RG$ and ψ an FOL-wff, then $[\alpha]\psi \in \mathcal{L}_{Int}$. In other words, an FODL language consisting only of program-free formulae preceded by a program. Intuitively, the meaning is that if $[\alpha]\psi$ is in the intention context, then the agent has the intention to achieve $\psi \in \mathcal{L}_{Des}$ by acting on plan $\alpha \in RG$. We will use goal and intention interchangeably and write symbol γ as shorthand for such a goal.

In addition to these mental contexts, we follow Casali's lead [3] and define two functional contexts: the Planner Context (PC) and the Interaction Context (XC). The first is in charge of finding plans to achieve an agent's desires, while the second is an agent's way of interacting with the world: it controls an agent's sensors, performs basic actions and sends and receives messages. To connect these contexts to the mental contexts of beliefs, desires and intentions, we use an additional notational device: quoting.

Definition 4.1 (Quotation operator)**.** *The quote-operator '.' transforms programs, PL-wffs, FODL-wffs and sets of FODL-wffs into first-order terms.*

The Planner Context uses a first-order language restricted to Horn clauses, where a theory of planning uses some special predicates for representing plans:

- $basic\_action(\text{`}\alpha\text{'}, \text{`}\Phi_{Pre}\text{'}, \text{`}\Phi_{Post}\text{'})$ where α is a basic action in RG and Φ_{Pre} and Φ_{Post} are program-free subsets of $\mathcal{L}_{Bel}$. This allows for the definition of basic capabilities of the agent, together with their pre- and post-conditions: for α to be executable, Φ_{Pre} must hold (or in other words, be in the agent's belief base). Φ_{Post} is guaranteed to hold on successful completion.

- $plan(\text{`}\alpha\text{'}, \text{`}\Phi_{Pre}\text{'}, \text{`}\Phi_{Post}\text{'})$ where α is any program in RG and Φ_{Pre} and Φ_{Post} are program-free subsets of $\mathcal{L}_{Bel}$. The meaning of this is the same as above, but allows for composite plans.

- $bestplan(\text{`}\psi\text{'}, \text{`}\alpha\text{'}, \text{`}\Phi_{Pre}\text{'}, \text{`}\Phi_{Post}\text{'})$ which corresponds to the agent's notion of the best instance of a plan to achieve desire ψ. At the very least we require $\psi \in \Phi_{Post}$.

In addition, the planner context contains two special predicates to choose plans based on the agent's beliefs and desires:

- $belief(\text{`}\Phi\text{'})$, with $\Phi \subseteq \mathcal{L}_{Bel}$.
- $desire(\text{`}\psi\text{'})$, with $\psi \in \mathcal{L}_{Des}$.

This context is more operational than the other contexts described thus far: its internal logic is unimportant and its function in the reasoning is to select a specific plan for the agent to execute in order to achieve a goal ψ, given the current beliefs about the world. The agent has some method of selecting the *best* plan to achieve a desire ψ, and this is the plan returned. How formulae of the form $belief(\text{`}\Phi\text{'})$ and $desire(\text{`}\psi\text{'})$ are introduced into the PC is described in the next section about bridge rules. This is the only context presented in this section that requires modification to allow for reasoning about trust, because, being oriented towards single agents, it does not accommodate actions that require other agents' participation. We will revisit this context in Section 5.3 to add this possibility.

The Interaction Context is also a functional context, but in a different sense from the planner context, as it encapsulates the sensors and actuators of an agent. It contains the following special predicates:

- $do(\text{`}\alpha\text{'})$ where $\alpha \in RG$. This has the meaning that α is performed by the agent, although at this abstract level we do not deal with the possibility of failure. Note that if α is a composite action the Interaction Context has some way of decomposing this into a sequence of basic actions which can be executed.

- $sense(\text{`}\varphi\text{'})$ with φ an FOL-wff. This has the meaning that φ is observed in the agent's environment.

4.3.1 Bridge rules

One of the main advantages of using an MCS to represent an agent is that it separates the deduction mechanism inside the context from the deduction between the contexts. Reasoning internal to the contexts is defined in terms of the deduction rules of the logic used within a context, while bridge rules allow for inference between the contexts. The first corresponds to reasoning about beliefs, or desires, individually, while the second corresponds to the BDI notion of a deliberation cycle: inference between the contexts. In the definition of the bridge rules below, and throughout the remainder of this paper, we use symbols $\Phi, \Phi_{Pre}, \Phi_{Post} \subseteq \mathcal{L}_{Bel},\ \psi \in \mathcal{L}_{Des}$ and $\alpha \in RG$. We start with the two bridge rules to add the beliefs and desires into the planning context, so that the beliefs can be used to find plans to achieve the desires:

$$\frac{BC : \Phi}{PC : belief(\text{`}\Phi\text{'})} \qquad \frac{DC : \psi}{PC : desire(\text{`}\psi\text{'})} \tag{1}$$

With this information the planning context deduces formulae with predicate $bestplan$ for any desire ψ the agent wishes to fulfill and any belief base Φ such that $\Phi_{Pre} \subseteq \Phi$

$$\frac{PC : bestplan(\text{`}\psi\text{'}, \text{`}\alpha\text{'}, \text{`}\Phi_{Pre}\text{'}, \text{`}\Phi_{Post}\text{'}) \qquad DC : \psi}{IC : [\alpha]\psi} \tag{2}$$

Similarly an agent has a bridge rule to allow for the execution of an intention:

$$\frac{DC : \psi \qquad IC : [\alpha]\psi}{XC : do(\text{`}\alpha\text{'})} \tag{3}$$

Upon execution, the effect of the agent's actions are added back into the belief base:

$$\frac{PC : basic\_action(\text{`}\alpha\text{'}, \text{`}\Phi_{Pre}\text{'}, \text{`}\Phi_{Post}\text{'}) \qquad XC : do(\text{`}\alpha\text{'})}{BC : \Phi_{Post}} \tag{4}$$

Note that none of these bridge rules take temporal considerations into account, or the possibility of failure: Φ_{Post} is instantly added to the belief base. They also require some form of belief revision to happen to keep the agent's beliefs consistent and up to date. Here we just give the minimal specification to allow for reasoning about trust, but one could think of extending the model to deal with such issues. The last bridge rule we define now is a simple rule for receiving sensory input:

$$\frac{XC : sense(\text{`}\Phi\text{'})}{BC : \Phi} \tag{5}$$

For the correct specification of a BDI-agent further bridge rules are necessary. Most notably those to allow for a chosen level of realism. Rao & Georgeff [29] specify three different levels of realism: strong realism (if an agent desires a formula to hold, it will believe the formula to be an option), realism (if an agent believes a formula, it will desire it) and weak realism (if an agent desires a formula to hold, it will not believe the negation of that formula to be inevitable). These each correspond to a different set of bridge rules between the belief base, desire base and intention base [23]. However none of this affects our specification of trust and we leave the choice of the level of realism open. The MCS as specified thus far is summarized in Figure 1.

5 Specifying Trust Models

The MCS described in the previous section lays the groundwork for an agent that can reason about trust. As discussed in Section 3, we assume a trust model is an algorithm to calculate a trust evaluation based on a number of inputs. These inputs are formulae in one of the contexts of the agent and we distinguish different types of input:

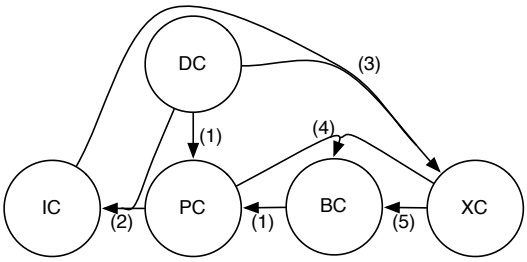

Figure 1: The MCS specification of a BDI-agent, without the contexts required for reasoning about trust. Nodes are contexts and (hyper)edges are bridge rules, with the labels corresponding to those in the text.

1. the target agent to be evaluated
2. beliefs about direct experiences
3. communicated trust evaluations
4. the desire the agent wishes to fulfill, for which it needs the target agent to perform some action

We will assume that the agent has a specific unary predicate $agent \in \mathcal{P}_{Domain}$, which is used to identify the agents in the system. We will use the shorthand $Agents = \{X \in \mathcal{C}_{Domain} | BC : agent(X)\}$, the set of identifiers of agents in the system. The first input of the trust model, the target agent to be evaluated, is thus an agent $a \in Agents$. The second and third inputs, the beliefs about direct experience and communicated trust evaluations, are also formulae in the belief context. While we did not model communication in our MCS, we can consider it as part of the interaction context. Sending a message is simply a basic action of the agent and do performs that basic action, sending the specified message. Similarly $sense$ also receives communications from other agents. We therefore have two subsets of the belief base Φ_{DE} and Φ_{Comm} representing the agent's beliefs about direct experiences and communications respectively. Furthermore, we assume that such beliefs form a distinguished subset in the language of beliefs. We take $\mathcal{P}_{DE} \subset \mathcal{P}_{Domain}$, the set of predicates about direct experiences and $\mathcal{L}_{DE}$, the set of FODL-wffs generated in the same way we described in Section 4.2.2. It is easy to see $\mathcal{L}_{DE} \subset \mathcal{L}_{Bel}$. The same holds for $\mathcal{P}_{Comm}$ and $\mathcal{L}_{Comm}$. The last input corresponds to the reason why the agent is evaluating the trustworthiness of the target in the first place: trustworthiness is a tool used to help the agent in choosing a partner to achieve some goal. Depending on the goal the agent wishes to achieve, the way the trust evaluation is computed changes. As such the goal γ that the agent is attempting to achieve is the fourth input for the trust model.

Some trust models [13] also consider norms as input for the trust model and in [26] a method of incorporating this into an MCS is given. However, a discussion about norms is outside the scope of this article. We will assume the agent's background knowledge is affected by norms and that this may affect how the agent specifies its trust model, however we will disregard norms in the use of the trust model: possible norm violations could already be encoded in the agent's direct experiences and received communications.

The output of the trust calculation is a trust evaluation: a predicate in a domain-specific language for describing trust. This predicate is an element of $\mathcal{P}_{Domain}$, and the set of all possible trust evaluations $\mathcal{L}_{Trust}$ is a program-free subset of $\mathcal{L}_{Bel}$.

This abstract description of a trust model in terms of its in- and output satisfies the condition given in Property 1 of Section 3: a trust model is a Turing computable function. Throughout this paper we call this function ***trust_calculation*** and, regardless of the actual computational trust model, we see that the domain of this function is the set of the agent's beliefs, goals and agents under evaluation; its range is the

set of possible trust evaluations. For example, if an agent wants to evaluate a salesman s for the goal of having a bicycle by performing the basic action $buy\_bicycle$, then it can evaluate s using its beliefs about interactions $\Phi_{DE} \subseteq \mathcal{L}_{DE}$ and communications $\Phi_{Comm} \subseteq \mathcal{L}_{DE}$ using its trust model. This may result in a trust evaluation of agent s with regards to the goal $[buy\_bicycle]have\_bicycle(me)$ with, for example, value $trustworthy$. We can represent this in a functional form as the function ***trust_calculation*** with the inputs Φ_{DE}, Φ_{Comm}, $[buy\_bicycle]have\_bicycle(me)$ and the agent s as described above:

$$\textbf{\textit{trust\_calculation}}(\Phi_{DE}, \Phi_{Comm}, [buy\_bicycle]have\_bicycle(me), s) = \\ trust(s, \text{`}[buy\_bicycle]have\_bicycle(me)\text{'}, trustworthy)$$

In Property 2 we recognized that, for reasoning about the trust model, this model must have something that can be adapted to the agent's needs. In other words, ***trust_calculation*** is a parametric function. We define the set $Params_{\textbf{\textit{trust\_calculation}}}$ as the set of parameters of ***trust_calculation*** and a function $labels : Params_{\textbf{\textit{trust\_calculation}}} \rightarrow 2^{\mathcal{C}_{PL}}$ that associates subsets of constants $\mathcal{C}_{PL}$ in a priority logic (PL) with the parameters of ***trust_calculation***. These constants are the factors that are of importance to the agent in calculating the trust evaluation, as discussed in Section 3.2. This will be formalized in Section 5.2, but first we give an example of a ***trust_calculation*** function.

5.1 An Illustrative Trust Model

We illustrate the parametrization of a trust model with the following small model to evaluate agents in an auction environment. The model calculates a trust evaluation of a trustee agent t with respect to a goal γ by using a weighted average to aggregate information from three different sources: the agent's own direct experiences as a buyer and as a seller with the trustee, specified in $Sales_t$ and $Purchases_t$, respectively, and the communicated trust evaluations from other agents in the system, specified in Φ_{Comm_t}. The ***trust_calculation*** function of this example is therefore as follows:

$$\textbf{\textit{trust\_calculation}}(Sales_t \cup Purchases_t, \Phi_{Comm_t}, \gamma, t) = \\ trust(t, \text{`}\gamma\text{'}, \frac{w_{buyer} \cdot DE_{Buyer}(Sales_t) + w_{seller} \cdot DE_{Seller}(Purchases_t) + w_{reputation} \cdot Comm\_Trust(\Phi_{Comm_t})}{w_{buyer} + w_{seller} + w_{reputation}})$$

w_{buyer}, w_{seller} and $w_{reputation}$ are the three weights and the functions DE_{Buyer}, DE_{Seller} and $Comm\_Trust$ calculate intermediate values for direct experiences with buyers, sellers and communicated evaluations, defined as follows:

$$Comm\_Trust(\Phi_{Comm_t}) = \sum_{C \in \Phi_{Comm_t}} \frac{value(C)}{|\Phi_{Comm_t}|}$$

$$DE_{Buyer}(Sales_t) = \sum_{S \in Sales_t} \frac{w_{profit} \cdot eval\_profit(S) + w_{time} \cdot eval\_paytime(S)}{|Sales_t| \cdot (w_{profit} + w_{time})}$$

$$DE_{Seller}(Purchases_t) = \sum_{P \in Purchases_t} \frac{w_{cost} \cdot eval\_cost(P) + w_{delivery} \cdot eval\_delivery(P)}{|Purchases_t| \cdot (w_{cost} + w_{delivery})}$$

$Comm\_Trust$ aggregates the communicated trust evaluations in a straightforward manner: it simply takes the communicated values as they are and averages these over all the received communicated evaluations. DE_{Buyer} and DE_{Seller} both use a straight up average over all interactions in which a sale, or a purchase, was made. Each interaction is evaluated by taking a weighted average of the evaluation of two aspects of the interaction. For sales interactions this is the profit made and whether the payment was on time. For purchases this is the cost incurred and whether the delivery was on time. The weights w_{profit} and w_{time} determine the importance of these two factors for DE_{Buyer}, while w_{cost} and $w_{delivery}$ do the same for DE_{Seller}. Each individual sales interaction is evaluated using the following functions:

$$eval\_profit(Sale) = max(-1, min(1, \frac{profit(Sale) - expected\_profit(Sale)}{profit\_threshold}))$$

$$eval\_paytime(Sale) = \begin{cases} -1 & \text{if } date\_payed(Sale) > payment\_deadline(Sale) \\ 1 & \text{otherwise} \end{cases}$$

$eval\_profit$ calculates the normalized value of profit. The $profit\_threshold$ is a constant: if the difference between the actual profit and expected profit is greater than the threshold, then the output of the function is capped at 1, or -1, depending on whether the difference is positive or negative. $eval\_paytime$ is a binary function: it is 1 if the price of the item was paid on time and -1 if it was not. Two very similar functions $eval\_cost$ and $eval\_delivery$ perform the same task for evaluating purchase interactions:

$$eval\_cost(Purchase) = max(-1, min(1, \frac{expected\_price(Purchase) - price(Purchase)}{cost\_threshold}))$$

$$eval\_delivery(Purchase) = \begin{cases} -1 & \text{if } delivery\_date(Purchase) > expected\_delivery(Purchase) \\ 1 & \text{otherwise} \end{cases}$$

This trust model serves as a first example of a model that can be incorporated into the agent's reasoning system. The key points of this demonstration are twofold: the first is to stress the intuition described so far that a trust model can be seen as a function taking inputs from the agent's mental contexts and calculating a trust evaluation. Obviously the actual calculation of this example is quite simple, however we demonstrate this in a similar manner for actual computational trust models in Section 7. The second point is to demonstrate that the designer of the system has a choice in what to consider as parameters. In the above system it would be intuitive to take the weights as parameters. However, such a choice does not fit our model: the value of a single weight in a weighted average is meaningless, because it is the ratio between weights that defines their relative importance within a weighted average. We thus take these ratios as our parameters. The actual value of the weights can follow trivially from these.

Note, that fixing the values for all possible ratios between all weights may lead to inconsistencies that would violate Property 2 of Section 3 (that any combination of values for the parameters yields a valid trust model). In our example, this problem is easily solved by choosing a mutually independent set of ratios for which the remaining ratios are dependent on. We thus choose the set of parameters $Params_{\textbf{\textit{trust\_calculation}}} = \{\frac{w_{profit}}{w_{time}}, \frac{w_{cost}}{w_{delivery}}, \frac{w_{buyer}}{w_{seller}}, \frac{w_{buyer}}{w_{reputation}}\}$ for the ***trust_calculation*** function. It is easy to see that Property 2 of trust models is fulfilled. While $profit\_threshold$ and $cost\_threshold$ could technically also be parameters of the calculation, they serve no purpose in adapting the trust model to the agent's behavior. Rather they need to be chosen correctly for the trust model to be of use in any domain.

5.1.1 Adapting the trust model

The parameters above can take any number of values, which result in different behaviors of the trust model. As argued in Sections 3 and 5, the behavior of the trust model should be adapted to the current beliefs and goal of the agent. The values of the parameters should depend on certain factors – the labels of the parameters – that the agent can prioritize over. For instance, if in the example the agent regards the delivery time of an item as more important than its cost, then the corresponding weights $w_{delivery}$ and w_{cost} in the trust calculation should reflect that.

We define this more generally, with the set of constants $\mathcal{C}_{PL}$, which can be used in a priority logic (PL) as defined in Section 4.2.3. In the above example we choose: $\mathcal{C}_{PL} = \{delivery\_time, cost, payment\_time, profit, outcome\_buying, outcome\_selling, reputation\}$, which corresponds exactly with the variables in the equations of the trust model. These are the factors that the agent uses to describe interactions in which it buys and sells items in an auction as well as the factors describing the intermediate stages of the trust calculation. Consequently changes in the relative importance between these factors should cause the trust model to change. In a real scenario far more factors can influence the

parameters, but we aim to keep the example simple.

The different parameters of the trust model are not all influenced by the same factors. As specified in the introduction of Section 5, the $labels$ function defines which factors influence which parameter. In our example we have:

- $labels(\frac{w_{profit}}{w_{time}}) = \{payment\_time, profit\}$
- $labels(\frac{w_{cost}}{w_{delivery}}) = \{delivery\_time, cost\}$
- $labels(\frac{w_{buyer}}{w_{seller}}) = \{outcome\_buying, outcome\_selling\}$
- $labels(\frac{w_{buyer}}{w_{reputation}}) = \{outcome\_buying, reputation\}$

In the continuation of this paper we describe a method that allows an agent to reason about the labels affecting its trust model. Viewing the trust model as a function allows us to abstract away from the actual computation and give this specification in more general terms than would be the case if we had to consider each method of computation separately.

5.2 A Priority System

Our language must allow for the specification of the importance of the different factors that are taken into account in a trust calculation. Another thing to note is that these factors may be both the initial inputs, such as communicated information or direct experiences, as well as internal predicates, such as the concepts of image and reputation in the case of Repage [31]. However, some comparisons do not make sense. For example, in the case of Repage it does not make sense to specify that direct experiences are more important than image, because nowhere in the algorithmic process do the two concepts occur together. In fact, image is considered as the output of an aggregation of, among other things, direct experiences. We therefore do not need to specify the importance ordering over all possible factors, because in the algorithmic design of the trust model some comparisons are pointless. However, we do need a way for identifying those factors that require ordering. For this we turn to the parameters of the trust model. We only need to define the importance between any factors that appear together in the labels related to a parameter. In our example, for instance, there is no need to define the importance between the factors $payment\_time$ and $reputation$ as the relative importance between these two factors do not influence the same parameter.

The parameters and their labels give us a natural way to specify what is important in the calculation of trust. *How* each of these labels influences the trust evaluation is dependent on the calculation itself. Take for instance, our example of a weighted average: the higher the ratio $\frac{w_{profit}}{w_{time}}$, the more weight is given to the profit made in a sale, in comparison to the punctuality of the payment. This is a natural translation from the relative importance of profit as opposed to punctuality. We do not elaborate on how exactly this translation is implemented. For instance, as long as the parameter $\frac{w_{profit}}{w_{time}}$ is given a value greater than 1, it complies with the idea that profit is more important than delivery time, independent of the actual value it has. However, that does not mean the actual value is unimportant: the values for this parameter influence the trust model in very different ways. We will assume the agent has a way of translating the relative importance into actual values for the parameters in a reasonable manner. One way to choose the values is through the relative importance of the priority rules, which we will discuss in Section 6, but first we need to finish formalizing this idea of relative importance between the various factors: we do this using our priority logic (see Section 4.2.3), which allows us to specify a priority system for each of the parameters, on the labels which influence its value.

Definition 5.1 (Priority System)**.**
The priority system context PSC contains a set of priorities for each parameter of the trust model. We

recall that a set of priorities is a theory in a PL-language $\mathcal{L}_{PL}$. Let ***trust_calculation*** *be a trust model with parameters $Params_{\textbf{trust\_calculation}}$, then for each parameter $p \in Params_{\textbf{trust\_calculation}}$ we define the PL-language $\mathcal{L}_{PL_p}$ with predicates $\succ$ and $=$ and constants $\mathcal{C}_{PL} = labels(p)$. A theory in $\mathcal{L}_{PL_p}$ is denoted as Π_p. The PSC therefore contains the indexed family $\{\Pi_p\}_{p \in Params_{\textbf{trust\_calculation}}}$.*

For instance, in our auction example, we could have the following PSC:

$$\{cost \succ \ delivery\_time\}_{\frac{w_{cost}}{w_{delivery}}}$$
$$\{profit \succ \ payment\_delay\}_{\frac{w_{profit}}{w_{time}}}$$
$$\{outcome\_buying = \ outcome\_selling\}_{\frac{w_{buyer}}{w_{seller}}}$$
$$\{outcome\_buying \succ \ reputation\}_{\frac{w_{buyer}}{w_{reputation}}}$$

These priorities would mean that any actual instantiation of a computational trust model must give more importance to cost than delivery time, profit to payment delays, and outcomes from either type of direct experience to reputation, when evaluating an agent.

Now we recall that trust is multifaceted: the priority system is dependent on what goal the trust evaluation serves. For instance the priorities may differ, depending on whether the agent intends to buy or sell an item. As such the priorities must be dependent on the goal. We call any goal, whose achievement depends on other agents, a *socially-dependent goal*.

5.3 Socially-dependent Goals

We recall from Section 4 that a goal in an agent is an FODL sentence that combines a desired outcome with a plan. Not all goals can be achieved by the agent working in isolation. We call a goal which requires interaction with another agent a socially-dependent goal. However, different plans may require different interactions to fulfill the same desire, so the need to interact must come from the specific manner in which to achieve the goal. Some desires may be fulfilled in strict isolation, but have alternative methods for fulfillment if interaction is considered. Other desires may inherently require interaction. Either way, it is the plan in which the interaction is defined. Such a plan is formed in the Planner Context and is a sequence of actions. We should therefore specify in the actions, whether there is a need for interaction. Furthermore we should define the type of action, or set of actions, we expect the other agent to perform (such as buying or selling an item). For this we will use an abstraction, so that rather than specifying the actual actions, we specify which *role* the agent should fulfill. We will assume such roles are defined in the MAS itself [21, 13] and the agent knows what can be expected from an agent performing any given role. The agent therefore has knowledge about a set of roles which can be performed in the system, using the special unary predicate $role \in \mathcal{P}_{Domain}$. We define the set $Roles$ analogously to $Agents$: $Roles = \{r | BC : role(r)\}$. These roles are used in the definition of a social action:

Definition 5.2 (Social action)**.** *$social\_action(`\alpha', `\Phi_{Pre}', `\Phi_{Post}', r)$ is defined analogously to the definition of $basic\_action$ in 4.3, where $\alpha \in RG$ is a basic action and $\Phi_{Pre}, \Phi_{Post} \subseteq \mathcal{L}_{Bel}$ are the pre- and post-condition, respectively. The distinction between a $basic\_action$ and a $social\_action$ is that the latter requires that some other agent, performing role $r \in Roles$, participates in the action. With this extension of the syntax of the planning context's internal logic, we also need to extend the definition of the planning predicate: $plan(`\alpha', `\Phi_{Pre}', `\Phi_{Post}', R)$, where $R \subseteq Roles$ is the set of roles required by social actions in the program α.*

Note that this is a simplified version of a more general definition, in which an action might require the participation of multiple agents performing multiple roles. This entire framework can be extended in a trivial manner to such multi-agent social actions, but to keep the notation clean we limit the actions to two

agents and thus a single other agent performing a single role. A socially-dependent goal is defined as any goal $[\alpha]\psi$ where α is a plan in which at least one social action is involved. For convenience we explicitly write the roles involved in the goal as $[\alpha]\psi\langle R\rangle$, as shorthand for $PC : plan(\alpha, Pre, Post, R) \wedge IC : [\alpha]\psi$. In other words R is the set of roles required by the social actions in the plan. We also write $[\alpha]\psi\langle r\rangle$ to denote any role $r \in R$ with the above condition. Whether an agent performs a certain role adequately is directly linked to its trustworthiness.

From an organizational perspective of the MAS we consider all agents inherently capable of performing any role. Whether they perform this role adequately is a matter for the individual agents to decide. It is also in this aspect that our multifaceted view of trustworthiness comes into play, as trust is based not just on agents' willingness to perform an action, but also their capability in performing this action [4]. When an agent wishes to accomplish some socially-dependent goal, it must attempt to find the agents best able to fulfill the required roles, using the corresponding priority system to instantiate the trust model to perform this evaluation.

6 Goal-based Instantiation of Trust Models

In the previous section we showed how to abstract away from a computational model and specify the factors that influence the trust computation. We can now tie this together with the BDI-reasoning of an agent. Particularly, we specify how the beliefs and goals affect the trust model. The factors that influence the trust model are defined in the priority system, but which PL-theory is used depends on the agent's reasoning system. The socially-dependent goal the agent wants to accomplish, the set of roles required to achieve this goal, and the beliefs an agent has about its environment may all influence the importance of the different factors and thus the trust model itself. A priority system, as described in Definition 5.1, is thus not a static set of rules, but is defined dynamically for each goal an agent intends to achieve: given a set of beliefs Φ, for each goal γ for which role $r \in R$ is required, the agent has a PL-theory $PS_{\gamma,r,\Phi}$ describing the priorities at that moment. We need to specify how priorities come into existence and how they are used in instantiating a trust model for a specific goal. So far we have only tangentially involved the reasoning system of the agent; now we will show how the trust model and priorities are involved in the reasoning. We use the MCS described in Section 4 and specify the bridge rules required.

6.1 Reasoning about the Priority System

We work from the intuition that an agent's beliefs, and the goal it is attempting to achieve, *justify* the priorities. We illustrate this by taking another look at the example of Section 5.1: if an agent believes it is beneficial to be frugal, it could choose to prioritize cost over delivery time and profit over payment delay. However, there might be a specific goal, such as if an agent needs to buy an item urgently, that could change these priorities: if the agent wants the item delivered quickly it could prioritize delivery time over cost.

To allow an agent to adapt its priorities – and thus its trust model – in such a manner we specify rules encoding causal relationships between cognitive elements and the priority system, formalized using a first-order logic restricted to Horn clauses. These rules are deduced in the priority rule context (PRC); a functional context similar to the planner context. We have three special priority rule predicates: $belief\_rule, role\_rule$ and $goal\_rule$. These predicates specify that a set of beliefs, a role or a goal support a certain ordering of priorities. Additionally, these predicates have a third argument, specifying the preference value for that rule. This is a numerical indicator of how important that rule is, which is used in the $resolve$ function below. Let $\Phi \subset \mathcal{L}_{Bel}$, $r \in Roles$ and $\gamma \in \mathcal{L}_{Int}$. Additionally let $v \in \mathbb{N}$ and $\pi \in \mathcal{L}_{PL_p}$ for some $p \in Params_{\textbf{\textit{trust\_calculation}}}$, then the following are priority rule predicates:

- $belief\_rule(\text{`}\Phi\text{'}, \text{`}\pi\text{'}, v)$. This allows for the definition of *priority rules* stating that the set of beliefs Φ supports the priority π with preference v.

- $role\_rule(r, \text{`}\pi\text{'}, v)$. The use is similar to the previous rule, but with a role, rather than beliefs, supporting a priority.
- $goal\_rule(\text{`}\gamma\text{'}, \text{`}\pi\text{'}, v)$. A priority rule stating that goal γ supports priority π with preference v.

Additionally, we have the predicates $belief, role$ and $goal$ to represent the agent's mental attitudes in the PRC, which are needed in the deductions. To see how this works in practice, we once again consider the example and formalize the scenario from the start of this section: the agent has the belief that it is frugal. This belief supports prioritizing cost over delivery time. The formalization of this is with the priority rule $belief\_rule(\text{`}frugal(me)\text{'}, \text{`}cost \succ delivery\_time\text{'}, 1)$.

How an agent obtains such priority rules is dependent on the implementation. Similar to the planning context, such rules can be predefined by the programmer, or the agent can be equipped with some reasoning system to deduce them. We use Υ to denote the set of all priority rules an agent has. The rules are *used* with a specific set of beliefs, a goal and a role, for which the agent needs to evaluate other agents' trust. As such we need bridge rules to connect the priority rule context to the belief and intention contexts:

$$\frac{BC : \Phi}{PRC : belief(\text{`}\Phi\text{'})} \qquad \frac{BC : role(\rho)}{PRC : role(\rho)} \qquad \frac{IC : [\alpha]\psi}{PRC : goal(\text{`}[\alpha]\psi\text{'})} \tag{6}$$

Let $p \in Params_{\textbf{\textit{trust\_calculation}}}$ be a parameter of the trust calculation, with its corresponding set of labels. Given a set of beliefs Φ, a role r and a goal γ we can now use the PRC to find a set of priorities $\Pi_{p,r,\gamma,\Phi}$ over the set $labels(p)$:

$$\begin{aligned}\Pi_{p,r,\gamma,\Phi} \;=\; & \bigcup_{\Phi' \subseteq \Phi} \{\pi | belief\_rule(\text{`}\Phi'\text{'}, \text{`}\pi\text{'}, v) \in \Upsilon \wedge terms(\pi) \subseteq labels(p)\} \\ & \cup \{\pi | role\_rule(r, \text{`}\pi\text{'}, v) \in \Upsilon \wedge terms(\pi) \subseteq labels(p)\} \\ & \cup \{\pi | goal\_rule(\text{`}\gamma\text{'}, \text{`}\pi\text{'}, v) \in \Upsilon \wedge terms(\pi) \subseteq labels(p)\}\end{aligned}$$

Note that while $\Pi_{p,r,\gamma,\Phi}$ is a wff in PL, it may be inconsistent. We illustrate this by continuing the formalization of the example. The agent in the scenario has the goal to buy an item, for instance a book, urgently. This can be formalized in the rule $goal\_rule(\text{`}[buy\_book]have\_book(tomorrow)\text{'}, (delivery\_time \succ cost), 2)$. The set of priority rules that is generated with the singleton set of beliefs $\{frugal(me)\}$ and goal $[buy\_book]have\_book(tomorrow)$ contains both $delivery\_time \succ cost$ as well as $cost \succ delivery\_time$ and is thus not a consistent PL-theory.

To use sets of priorities generated in this manner in the Priority System, the agent must perform some form of conflict resolution, which guarantees a consistent PL-theory. This is done using $resolve$: a function that, when given a set of priorities, a set of beliefs, a goal, a role and a trust model, returns a consistent PL-theory. It is immediately obvious that there are many different ways of obtaining a consistent PL-theory from a set of priorities and there are intuitively better, and worse, ways of doing this. For instance, $\emptyset$ is a consistent PL-theory, but it is not a very useful one! To maximize the utility of the obtained PL-theory we use the preference factors for each of the rules in Υ. The output of the $resolve$ function must fulfill the following two conditions: (1) it is a consistent PL-theory and (2) the total preference value of the rules used is maximal. This allows the agent designer a lot of freedom in defining how to resolve the priorities. For instance, one way would be to guarantee that $resolve$ returns a maximal subset of $\Pi_{p,r,\gamma,\Phi}$ that is a consistent PL-theory. Another possibility ties in with the idea that the trust model should be goal-oriented. In such a case the priorities supported by the goal should supersede the priorities supported by the role, which in their turn should supersede those supported by the beliefs. Both of these methods can be expressed by choosing adequate preference factors for the rules: if we just want the maximal subset, all preferences can be 1, but if the implementation calls for some distinction between the rules, this can be implemented using the preferences of the rules.

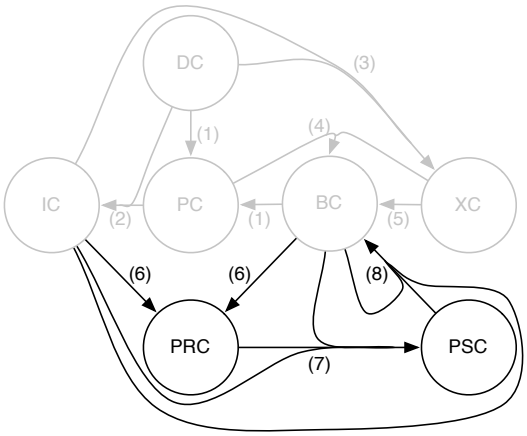

Figure 2: The MCS specification of a BDI-agent that can reason about trust. This is an extension of Figure 1, whose bridge rules are colored gray. The bridge rules added for reasoning about trust are black, with labels corresponding to those in the text.

We have now defined all the individual parts of the extended BDI model and can define the first bridge rule related to reasoning about trust, namely the generation of a priority system:

$$\frac{BC : \Phi \qquad IC : \gamma = [\alpha]\psi\langle r\rangle \qquad PRC : \Upsilon}{PSC : PS_{r,\gamma,\Phi}} \tag{7}$$

with $PS_{r,\gamma,\Phi} = \{resolve(\Pi_{p,r,\gamma,\Phi}, \Phi, \gamma, r, \textbf{\textit{trust\_calculation}})\}_{p \in Params_{\textbf{\textit{trust\_calculation}}}}$ being the outcome of an operational procedure, as alluded to in Section 4.1.

It is up to the implementation of the agent to define $resolve$ as well as the rules in Υ (or a method for generating them).

6.2 Instantiating Trust Models

A socially-dependent goal γ is not immediately executable, unlike a goal with a plan which does not contain social actions. To achieve a social action it is not sufficient to define which role an agent must fulfill, but the agent must actually choose another agent to fulfill that role. For this the trust evaluation of other agents in the system must be calculated, so that a decision can be made whom to interact with. As stipulated earlier, we require there to be a method to obtain an instantiation of a trust model complying with a priority system. Accordingly, the agent can use such an instantiation to calculate its trust evaluations. A system implementing our model has to provide a definition of the $instantiate$ function that, when given a priority system $PS_{r,\gamma,\Phi}$ and a trust function ***trust_calculation***, outputs another trust function $\textbf{\textit{trust\_calculation}}_{r,\gamma,\Phi}$, such that $instantiate(PS_{r,\gamma,\Phi}, \textbf{\textit{trust\_calculation}})$ complies with the priority system $PS_{r,\gamma,\Phi}$. A trust model complies with a priority system if the value of each of its parameters complies with the priorities over its labels, as explained in Section 5.2. Both $instantiate$ and how the compliance of the resulting trust function is checked depend heavily on the actual trust model used and we will demonstrate this in an example in Section 7.1.

Now this $\textbf{\textit{trust\_calculation}}_{r,\gamma,\Phi}$ can be used to aid the agent in selecting a partner for the required role. However trust may not be the only thing an agent uses to select a partner. In a BDI-based agent architecture all reasons an agent may have to select a partner are usually inferred from the belief base. Therefore

trust evaluations should be added to the belief base, allowing them to be incorporated in this reasoning process. Once a partner has been selected, a social action may be executed just as a basic action can be executed by the agent. The details of how this happens is outside the scope of a high-level specification, but we do need to specify how trust evaluations get added to the belief base. ***trust_calculation***$_{r,\gamma,\Phi}$ is a functional representation of an underlying computational trust model. This trust model can calculate the trust evaluation of prospective partners for the achievement of the goal γ. These calculations need to be added in the belief base and, accordingly, we require a bridge rule for this:

$$\frac{IC : \gamma = [\alpha]\psi\langle r\rangle \qquad BC : \Phi \qquad PSC : PS_{r,\gamma,\Phi}}{BC : \varphi_{\gamma,r,a}} \tag{8}$$

where $\varphi_{\gamma,r,a} =$ ***trust_calculation***$_{r,\gamma,\Phi}(\Phi_{DE}, \Phi_{Comm}, \gamma, a)$ with $\Phi_{DE}, \Phi_{Comm} \subset \Phi$ and $a \in Agents$.

Finally this trust evaluation can then be used in the execution of a plan, by selecting the best partner for interaction. The MCS with the two new contexts and the additional bridge rules is represented schematically in Figure 2.

7 Integrating Trust Models

With the extended BDI-framework in place we can now show how this allows an agent to reason about its trust model. We will demonstrate how to incorporate three different trust models into the framework: BRS [15], ForTrust [20] and ReGReT [30]. We show how incorporating a trust model into the framework allows the agent to proactively change its trust evaluation, in addition to allowing an easy and intuitive way of allowing any model to deal with the multifaceted aspect of trust by use of the agent's goals and the roles other agents may perform. We only demonstrate the entire reasoning model for BRS, while presenting a detailed discussion of the algorithmic representation of the other models and their parameters.

7.1 BRS

The Beta Reputation System (BRS) is a statistical method for calculating reputation: the aggregation of other agents' trust evaluations. The approach described by Jøsang & Ismail [15] is a centralized approach and can thus be seen as a model which does not take individual's own direct experiences into account separately. However, as they explicitly mention, this same method can be used in a decentralized approach and newer extensions of this model, such as TRAVOS [36], are distributed. The basis of TRAVOS is the same statistical model as Jøsang & Ismail describe, so we focus on BRS. An agent's own direct experiences are aggregated just the same as any other agent's communicated experiences with no special preference. BRS is based on the beta-family of probability distributions of binary events. This family is expressed by the following probability density function, where p is a probability conditioned by the shape parameters α and β and Γ is the Gamma function[2]:

$$f(p|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\cdot\Gamma(\beta)} \cdot p^{\alpha-1} \cdot (1-p)^{\beta-1}, \text{ where } 0 \leq p \leq 1, \alpha > 0, \beta > 0$$

The expected value, given such a probability density function is:

$$E(p) = \frac{\alpha}{\alpha+\beta}$$

The expected outcome is thus in the range $[0, 1]$. BRS uses this expectation as the reputation, but converts it to a value in $[-1, 1]$, which they state is more intuitive to human users. The α and β are

[2] $\Gamma(x) = \int_0^\infty t^{x-1} \cdot e^{-t}\mathrm{d}t$

determined by the number of "good" and "bad" evaluations of interactions with a certain trustee. Jøsang & Ismail add further refinements by discounting opinions from agents who are uncertain and discounting experiences over time. Additionally they provide a method for using non-binary evaluations of an interaction by treating a single interaction as a number of interactions with a mix of "good" and "bad". A numerical evaluation of a single interaction can be represented as a pair (r, s) where r is the value of "good" and s the value of "bad" in that single interaction. It is necessary to keep $r+s$ constant, but in this way we can add r good and s bad evaluations for each interaction, representing non-binary evaluations. By using a weight w it is further possible to give important interactions a higher value for r and s and thus more importance in the determination of α and β, although this is not explored further by Jøsang and Ismail [15]. Algorithm 1 describes the entire process.

Input: $Evals = \{(v, a, t)\}$, a set of evaluations. Each evaluation with value $v \in [-1, 1]$, evaluator agent a and time t
Input: $j \in Agents$, the agent to be evaluated
Input: w, the weight parameter
Input: λ, a parameter for decay over time
$s := 0$
$r := 0$
foreach $(v, a, t) \in Evals$ **do**
  $r := r + \lambda^{now-t} \cdot w \cdot (1 + v)/2$
  $s := s + \lambda^{now-t} \cdot w \cdot (1 - v)/2$
reputation := $\frac{r-s}{r+s+2}$
Output: $Reputation(j, \text{reputation}, now)$

Algorithm 1: Centralized BRS

7.1.1 Parametrization of centralized BRS

When looking at the algorithm we see that there are two explicit parameters, w and λ. λ can be interpreted as defining the balance between new and old evidence: if $\lambda = 1$ the time since the evidence was observed is irrelevant; with $\lambda < 1$ historical evidence is given less importance and the unusual situation of $\lambda > 1$ would mean the longer ago the evidence was observed, the more importance it is given. However, w in the formula given by Jøsang and Ismail only influences the convergence rate. We feel the weight w can be put to better use: as they themselves describe, it can be used to define the *importance* of an interaction to the final calculation. This ties in directly to another issue with the algorithm: it starts off with a set of evaluations without describing how these are calculated from an agent's beliefs. Thus the algorithm above only describes the *aggregation* method used for calculating trust based on individual evaluations, but not the evaluation method itself, which forms an important part of the evaluation. This is mainly because the article describes a *centralized* method in which it uses communicated information: the computational entity that performs the aggregation is not the same one that performs the evaluation of individual interactions. We thus propose a decentralized version of BRS defined in Algorithm 2, which is a procedure for calculating the outcome of the ***trust_calculation*** function for an agent using BRS as its trust and reputation model. This uses three secondary functions: **eval**, **weight** and **time_stamp** to calculate the value of a direct experience, the weight assigned to this value by the agent, and the time the direct experience took place, respectively. Because the agent's knowledge of its direct experiences and received communications are collected in its belief base, we make the link to the belief context BC explicit in the algorithm.

This algorithm does not strictly follow the description of BRS [15]. To make sense in a decentralized setting a distinction must be made between the own and others' experiences. This distinction need not be

Input: $j \in Agents$, the agent to be evaluated
Input: $BC : I_j$, a set of direct experiences with agent j
Input: $BC : R_j = \{received(reputation, i, j, v, t)\}$, a set of messages with sender i, communicating that agent j has reputation value v at time t
Input: λ, a parameter for decay over time
$s := 0$
$r := 0$
foreach $i \in I_j$ **do**
  $v :=$ **eval**(i)
  $w :=$ **weight**(i)
  $t :=$ **time_stamp**(i)
  $r := r + \lambda^{now-t} \cdot w \cdot v$
  $s := s + \lambda^{now-t} \cdot w \cdot (1 - v)$
foreach $received(reputation, i, j, v, t) \in R_j$ **do**
  $Rep_i := (1 + rv)/2$, where $BC : Reputation(i, rv, now - 1)$
  $r := r + \lambda^{now-t} \cdot Rep_i \cdot v$
  $s := s + \lambda^{now-t} \cdot Rep_i \cdot (1 - v)$
reputation $:= \frac{r-s}{r+s+2}$
Output: $Reputation(j, \text{reputation}, now)$

Algorithm 2: Decentralized BRS

made in a centralized system and therefore we propose to discount incoming communications based on the reputation of the sender. This discounting is not done in the original article, but we feel it represents an accurate extension of BRS to a decentralized model. The own experiences are evaluated using the functions **eval** and **weight**. These can easily be parametrized to be dependent on the goal the agent is attempting to achieve.

As mentioned earlier, TRAVOS [36] is an example of a newer trust model that extends BRS and it makes a similar distinction between own direct experiences and incoming communications as in Algorithm 2, however it extends the BRS model significantly, mainly in the way it chooses what information to take into account when calculating a trust evaluation. BRS uses all information available, although weights can influence this. TRAVOS, however, uses further calculations of the confidence in different sources to possibly disregard information.

7.1.2 Parametrization of decentralized BRS

The only explicit parameter is the same one as in the centralized model, the decay factor for time. However, there is also an implicit one: while the communicated evaluations always have a weight of at most 1, the weight of own experiences may be arbitrarily high. This leads to an implicit weighing of the own experiences as compared to the communicated evaluations with ratio $w_{direct} = max_{i \in I_j}(\textbf{weight}(i))$. This implict weight can be made explicit by adding the constant $w_{reputation}$, that is multiplied with all the communicated evaluations (in addition to the reputation of the sender). The parameter $\frac{w_{direct}}{w_{reputation}}$ describes the relative influence of direct experiences and communicated evaluations, thereby allowing the agent to modify this parameter to change this balance. The **eval** function may also be parametrized, similar to the other models and our description in Section 3.3.

7.1.3 An e-commerce agent using decentralized BRS

We will now show how this system allows an agent to proactively change its trust model, using Algorithm 2 as the procedure for calculating trust. The implementation depends on the domain in which the agent is operating, so to keep our example compact, we use the same domain as in the example of Section 5.1: an agent in an e-commerce environment. The environment has two roles an agent may play, buyer or seller, corresponding to the actions an agent can perform, buying an item or selling an item. To evaluate agents' direct interactions we use evaluation functions similar to those described in Section 5.1, but returning the outcome in a format that BRS can handle. BRS performs the aggregation, so we just need to provide a definition for the functions **eval** and **weight**. BRS does not give any description of how these should be defined and it is up to the designer. We choose the following definitions, for the different types of interactions, because they are simple calculations that have parameters which we can adjust:

$$\textbf{eval}(DE_{Sale}) = round(w_{profit} \cdot eval\_profit(DE_{Sale}) + w_{time} \cdot eval\_paytime(DE_{Sale}))$$
$$\textbf{eval}(DE_{Purchase}) = round(w_{cost} \cdot eval\_cost(DE_{Purchase}) + w_{delivery} \cdot eval\_delivery(DE_{Purchase}))$$
$$\textbf{weight}(DE_{Sale}) = w_{sale}$$
$$\textbf{weight}(DE_{Purchase}) = w_{purchase}$$

which use the functions:

$$eval\_profit(Sale) = max(-1, min(1, \frac{profit(Sale) - expected\_profit(Sale)}{profit\_threshold}))$$
$$eval\_paytime(Sale) = \begin{cases} -1 & \text{if } date\_payed(Sale) > payment\_deadline(Sale) \\ 1 & \text{otherwise} \end{cases}$$
$$eval\_cost(Purchase) = max(-1, min(1, \frac{expected\_price(Purchase) - price(Purchase)}{cost\_threshold}))$$
$$eval\_delivery(Purchase) = \begin{cases} -1 & \text{if } delivery\_date(Purchase) > expected\_delivery(Purchase) \\ 1 & \text{otherwise} \end{cases}$$

Having now specified all the calculations of the ***trust_calculation*** function, we choose the following set of parameters: $Params_{\textbf{\textit{trust\_calculation}}} = \{\lambda, \frac{w_{profit}}{w_{time}}, \frac{w_{cost}}{w_{delivery}}, \frac{w_{sale}}{w_{purchase}}, \frac{w_{sale}}{w_{reputation}}\}$, giving a full parametrization of BRS. The last four parameters are described in Section 5.1, while λ is a parameter from Algorithm 2. We demonstrate the agent's reasoning using a simple set of labels. These labels are constants that allow us to capture the meaning of the parameter. The numerical value of a parameter is thus symbolically represented as an ordering of factors that influence the parameters:

- $labels(\lambda) = \{old, new\}$
- $labels(\frac{w_{profit}}{w_{time}}) = \{payment\_time, price\}$
- $labels(\frac{w_{cost}}{w_{delivery}}) = \{delivery\_time, cost\}$
- $labels(\frac{w_{sale}}{w_{purchase}}) = \{sale\_experience, purchase\_experience\}$
- $labels(\frac{w_{sale}}{w_{reputation}}) = \{sale\_experience, reputation\}$

The ordering of these labels are defined in a priority system, which is deduced in the PRC. The following are a sample of rules an agent may use to reason in this example.

The rules for deducing the PL for parameter λ are:

- $belief\_rule(\text{`}dynamism(environment, high)\text{'}, \text{`}new \succ old\text{'}, 1)$
- $belief\_rule(\text{`}dynamism(environment, none)\text{'}, \text{`}new = old\text{'}, 1)$

These state that if the environment is very dynamic, old information should be given less importance than new, whereas if the environment is not dynamic, all information should be treated equally.

For $\frac{w_{profit}}{w_{time}}$ the rules are:

- $belief\_rule(\text{`}frugal(me)\text{'}, \text{`}price \succ payment\_time\text{'}, 1)$
- $belief\_rule(\text{`}[pay(rent, landlord)]have(home, next\_month) \wedge \neg have(rent, now)\text{'}, \text{`}payment\_time \succ price\text{'}, 5)$
- $goal\_rule(\text{`}[sell(item)]have(money, future)\text{'}, \text{`}price \succ payment\_time\text{'}, 1)$

With the meaning that if the agent is frugal, price is given more importance than the deadline for payment. A similar rule applies if the agent does not have a goal to sell an item urgently. However, if the agent needs money urgently, for instance to pay the rent, then it should give a high importance to the time the payment is made, rather than the profit made.

For $\frac{w_{cost}}{w_{delivery}}$:

- $belief\_rule(\text{`}frugal(me)\text{'}, \text{`}cost \succ delivery\_time\text{'}, 1)$
- $goal\_rule(\text{`}[buy(item)]have(item, tomorrow)\text{'}, \text{`}delivery\_time \succ cost\text{'}, 2)$

Similar to the rules for $\frac{w_{profit}}{w_{time}}$, a frugal agent should prioritize the cost over timeliness of the delivery. However, if the item is needed urgently, for instance the next day, then the delivery time is more important than the cost.

For $\frac{w_{sale}}{w_{purchase}}$:

- $role\_rule(seller, \text{`}purchase\_experience \succ sale\_experience\text{'}, 2)$
- $role\_rule(buyer, \text{`}sale\_experience \succ purchase\_experience\text{'}, 2)$

These rules define the importance in the aggregation process for direct experiences of different types. If the agent is searching for a seller, then it should give more importance to direct experiences in which it was buying items than those in which it was selling. Vice versa if the agent is searching for a buyer.

Finally for $\frac{w_{sale}}{w_{reputation}}$:

- $belief\_rule(\top, \text{`}sale\_experience \succ reputation\text{'}, 1)$
- $belief\_rule(\text{`}\forall x \in Agents : good\_reputation\_source(x)\text{'}, \text{`}sale\_experience = reputation\text{'}, 2)$

The first rule states that the default is for sale experiences to be more important than reputation. However, if all agents in the system are good reputation sources then the two types of information should be given equal importance.

These rules are triggered by the first parameter (the antecedent) being true in its respective context. The bridge rules (6) cause these true sentences to be added in the PRC, where the internal reasoning checks which priority rules hold true at any one time. For bridge rule (7) to be of any effect, we need to define a $resolve$ function. An example of a $resolve$ function that resolves possible conflicts in the priority system could be to perform a best first search for each parameter, which removes rules from the set of applicable rules recursively, based on their preference value (removing the lowest values first) until the set of rules results in a consistent PL-theory, thus guaranteeing the highest value set that forms a consistent PL-theory. The priority system to be used is the family of the PL-theories for each individual

parameter. In our example case we can use a simpler algorithm for $resolve$, because each parameter is only influenced by two factors. We thus have that, for any two factors a, b a consistent set of priorities states that either $a \succeq b$, $b \succeq a$ or $a = b$. We sum the preference values for the priority rules supporting any of the three consistent cases, and the one with the highest total value wins. If all three cases are tied, then $a = b$. In the case of a tie between the first two cases, we need to break the tie in some manner. For instance, we could choose at random, or use the number of rules supporting each case. Our choice is to use the priority supported by the single rule with the highest preference as a tiebreaker, with as rationale that the weights accurately reflect the importance for having the priority, and one important rule should be chosen over multiple less important ones. If it is still tied, we choose one of the tied cases at random.

To obtain a trust model complying with these parameters, we also need to define the $instantiate$ function. We give an example of a straightforward manner for obtaining such a function, although this, once again, depends on parameters being influenced by only two factors. For the first parameter, λ, the value is determined simply by discerning the two cases: if $new = old$ then $\lambda = 1$, otherwise $\lambda = 0.9$ (which we use as a default decay rate).

For the other parameters, we use the preference values of the priority rules that are triggered by the agent's requirements. We demonstrate the function for the parameter $\frac{w_{profit}}{w_{time}}$. Let us assume the agent requires trust evaluations for a role r, goal γ and beliefs Φ, and that we have $\pi = resolve(\Pi_{\frac{w_{profit}}{w_{time}}}, \Phi, \gamma, r, \textbf{\textit{trust\_calculation}})$. We define $instantiate$ in the following manner: if $\pi \equiv (price = payment\_time)$ then $\frac{w_{profit}}{w_{time}} = 1$, otherwise we depend on the preference values of the priority rules. We define S^{profit} as the sum of the preference values of the priority rules in Υ, whose conclusion is $price \succ payment\_time$ and that are triggered by role r, goal γ, or a subset of the beliefs Φ. We define S^{time} analogously, but for the priority rules with $payment\_time \succ price$ in the conclusion. We now calculate the ratio $\frac{w_{profit}}{w_{time}} = \frac{1+S^{profit}}{1+S^{time}}$, where the 1 is added in both the numerator and denominator to avoid division by 0. There is a particular case in which this calculation does not reflect the priority π, which corresponds to when we need to use a tiebreaker in the $resolve$ function: if $S^{price} = S^{profit}$, but $\pi \not\equiv (price = payment\_time)$, then we want $\frac{w_{profit}}{w_{time}}$ near 1, but still reflecting the priority π. In this case we choose an $\varepsilon > 0$ and use $\frac{w_{profit}}{w_{time}} = \frac{1+\varepsilon}{1}$ if $price \succ payment\_time$, or $\frac{w_{profit}}{w_{time}} = \frac{1}{1+\varepsilon}$ otherwise.

A similar function exists for the other weight ratios. Note that the trust model does not use the weight ratios, but needs to be instantiated with actual weights. We choose a simple algorithm for instantiating these weights, using the ratios. The algorithm starts by assigning the value 1 to each weight. Then, for each parameter, it adjusts one of the weights so that the ratio in the parameter is correct (in other words, if $\frac{w_{profit}}{w_{time}}$ has value $1/2$ it could adjust w_{profit} to 0.5 or w_{time} to 2). The algorithm always starts by adjusting a weight that has value 1, before adjusting other weights. This loop continues until all parameters are satisfied.

Now we see that this allows the agent to instantiate different trust models with different weights, depending on the role the other agent plays in the interaction, if there is urgency in selling off stock, or the agent believes all other agents to be truthful in their reputation assessments. The agent designer could create more priority rules (or implement a system in which the agent learns priority rules) to cover more cases, similar to agent's plans are required to allow an agent more flexibility in fulfilling its desires.

We will not discuss the integration of ForTrust or ReGReT in such a detailed example, but rather show how those algorithms can be parametrized. The further steps to full integration into the extended BDI model are left to the engaged reader.

7.2 ForTrust

ForTrust is a logical framework for describing properties of trust: it provides a definition of trust in the trustee's action [20] and states that an agent trusts a target agent to perform an action α, which will fulfill a desire φ, if the following conditions hold:

Power: the target agent can ensure φ, by doing α
Capability: the target agent is able to perform action α
Intention: the target agent intends to do α

Trust is then defined in a multimodal logic, with trust as a modality equivalent to an agent having a desire (or, in their logic, an achievement goal) φ and beliefs corresponding to the above conditions of power, capability and intention of the target, desire and action. Their formalization of trust thus incorporates those aspects of trust that we have mentioned earlier: an agent trusts another agent with regards to a specific goal. The role an agent plays is directly dependent on the social action that it is required to perform. As such this formalization fits exactly with our own one. Lorini and Demolombe [20] also present a *graded* version of the multimodal logic, which allows trust to have a strength, rather than be binary as in classical logic. Either way, this logical description of trust is very abstract and it does not describe the computational processes involved. For instance, the very basics of computational trust: when given a direct experience in which the target agent performed α and the result was a world in which φ' held, how likely is it that next time the agent performs α, the desire φ is fulfilled? This is just one of the issues that the formalization stays away from, but any implementation must necessarily solve.

Hübner et al. present an implementation of this graded version of ForTrust, using Jason [14]. This implementation resolves many of the abstract issues by making very specific choices for how to calculate the outcome of parts of the formalization in the ART Testbed [9]. The ART Testbed is a testbed for trust models in which the aim is to obtain the best possible appraisal of various pieces of art. Thus the trust evaluation depends on the trustee actually performing the "appraise" action, thereby fulfilling the conditions of capability and intention and if the belief "good_eval" is true after appraising, then the condition of power is also fulfilled. Because the agent has a graded belief base, both the belief that the trustee will perform the "appraise" action and the belief that "good_eval" will hold are numerical and the final trust evaluation of a trustee is the minimum of these two grades. This definition is very specific and we give a more general version of the same system in Algorithm 3, which is not limited to application in only the ART Testbed.

We assume the existence of the predicates $request(a, \alpha, contract)$ and $performed(contract, a, \psi, time)$ in $\mathcal{L}_{Bel}$ to model the requests to perform action α from an agent a and the result ψ of a's performance of α, respectively. The function **eval** then compares the outcome of a previous interaction with the requirement φ and returns a numerical value. It is easy to fill in a formula here so the same result is obtained as in Hübner et al.'s implementation for the ART Testbed [14]. Another application-specific implementation of ForTrust is presented by Krupa et al. [18], which also adds the possibility of evaluating the trust in an agent to not perform a malicious action. While this requires a different type of logical reasoning, from a trust perspective it may be treated in a similar manner. An agent simply trusts another agent regarding the ficticious action α': the action of not performing α. By considering the absence of an action as performing a different action and reasoning about that instead, Algorithm 3 is also a generalization of Krupa et al's implementation of ForTrust.

A final remark is that while the ART Testbed provides the possibility of asking for reputation information from other agents, Hübner et al.'s implementation does not use this. We have therefore not included it either, but, just as the integration of *reputation* is considered future work by Hübner et al., this algorithm can be extended to include reputation as a source of information for estimating both the power and capability of other agents.

7.2.1 Parametrization of ForTrust

Looking at Algorithm 3 we see there are three parameters explicitly specified. Of these, γ is the easiest to interpret: it has the exact same effect as λ in BRS (see Section 7.1). The parameters δ and ϵ can be considered independently or together. Independently they describe the cut-off rates for power and capability, respectively. As such, a high ϵ means we do not wish to consider agents who do not perform

Input: $BC : \mathcal{B}$, the set of the agent's beliefs
Input: $IC : [\alpha]\varphi$, the goal to be achieved
Input: $a \in Agents$, the agent to be evaluated
Input: ϵ, a parameter defining the cut-off rate for capability of performing α
Input: δ, a parameter defining the cut-off rate for the power of achieving φ by performing α
Input: γ, a parameter defining the decay factor for evidence over time
Input: c_0, a default value for the $capability$
Input: p_0, a default value for the $power$
performed := 0
contracts := 0
outcome := 0
denominator := 0
foreach $request(a, \alpha, contract) \in \mathcal{B}$ **do**
    contracts := contracts + 1
    if $performed(contract, a, \psi, t) \in \mathcal{B}$ **then**
        performed := performed + 1
        modifier := γ^{now-t}
        outcome := outcome + modifier · **eval**(φ, ψ)
        denominator := denominator + modifier
if contracts $= 0$ **then**
    capability := c_0
else
    capability := $\frac{\text{performed}}{\text{contracts}}$
if capability $\leq \epsilon$ **then**
    capability := 0
if performed $= 0$ **then**
    power := p_0
else
    power := $\frac{\text{outcome}}{\text{denominator}}$
if power $\leq \delta$ **then**
    power := 0
x := min(capability, power)
Output: $trust(a, \text{`}[\alpha]\varphi\text{'}, x)$

Algorithm 3: ForTrust

α when asked, independent of the degree of success at achieving φ. We see ϵ thus directly specifies the importance of being capable, as we would expect from the algorithm. Similarly, δ specifies the importance of actually achieving φ if the agent performs α. However, the interplay between the two variables is complicated, because if one of these two cut-off rates is unrealistically high, then it does not make any difference what the other value is at, because the trust will be 0, based on this unrealistic expectation. In other words, it depends on the probability distributions of power and capability, what influence δ and ϵ have. When both are above the cut-off rate, whether capability or power is the deciding factor also depends on their probability distributions: because we take the minimum, the smallest of the two is the deciding factor, given they are both larger than the cut-off rate. We thus see that capability is the deciding factor in the following two situations:

- $capability \leq \epsilon$ and $power > \delta$
- $capability > \epsilon$, $power > \delta$ and $capability < power$

The situations that power is the deciding factor are analogous. Note that we disregard the situations where both power and capability are smaller than δ and ϵ, respectively, because in such situations both are 0. Because we need to choose the values for δ and ϵ before calculating the agent's capability and power, the best we can do is use an estimation: we can influence the probability that capability (or power) is the deciding factor. The probability that capability is the deciding factor is thus the sum of the probabilities for either case above. This works out to the following formula:

$$\begin{aligned} Pr(cap\_decides) \;=\; & Pr_{cap}(c \leq \epsilon) \;\cdot\; Pr_{power}(p > \delta) \\ & +\; Pr_{cap}(c > \epsilon) \;\cdot\; Pr_{power}(p > \delta) \;\cdot\; Pr_{cap}(c < p \mid p > \delta) \\ =\; & F_{cap}(\epsilon) \;\cdot\; (1 \;-\; F_{power}(\delta)) \\ & +\; (1 \;-\; F_{cap}(\epsilon)) \;\cdot\; (1 \;-\; F_{power}(\delta)) \;\cdot\; \int_{\delta}^{\infty} (F_{cap}(x) - F_{cap}(\epsilon)) \cdot Pr_{power}(p = x)dx \end{aligned}$$

Where $Pr_{cap}(c > \epsilon)$ calculates the probability that c is greater than ϵ given the probability distribution of the capability of the target. Pr_{power} does the same for power, F_{cap} is the cumulative probability function for capability, and F_{power} the cumulative probability for power.

A similar equation can be found for deciding when power defines the trust value, which, because we are not discounting those situations where both $p \leq \delta$ and $c \leq \epsilon$, will not simply be the inverse probability of the above formula. All of this is needed in order to say something about how the values for δ and ϵ influence the relation between $Pr(cap\_decides)$ and $Pr(power\_decides)$. However, as we can see, the equations are rather complex and, additionally, the probability distributions for power and capability are generally unknown, and thus the influence cannot be calculated exactly. As a rule of thumb we can assume that power and capability are distributed equally. In this case the probability that the capability is the deciding factor is greater than the probability that power is the deciding factor when ϵ is greater than δ. For this, we need to show that, if $\epsilon > \delta$, $Pr(cap\_decides) - Pr(power\_decides) \geq 0$. The proof is straightforward and hinges on two realizations: the first is that the two probability distributions for power and capability are the same, so Pr_{cap} and Pr_{power} represent the same probabilities. The second is that if $\epsilon > \delta$ the same holds for cumulative probabilities: $Pr(x \leq \epsilon) \geq Pr(x \leq \delta)$. With these two realizations, we can simplify the algebraic expression as follows:

$$Pr(cap\_decides) - Pr(power\_decides) \geq Pr(x \leq \epsilon) + Pr(x > \epsilon) \cdot Pr(x > \delta) \geq 0$$

□

The reverse is also true and $Pr(power\_decides) \geq Pr(cap\_decides)$ if $\delta > \epsilon$. We simplify this further and say that, under the assumption that power and capability are equally distributed, if $\delta > \epsilon$ then power is more important than capability and vice versa if $\epsilon > \delta$.

Even if the assumption of equal distributions does not hold, it is clear that the relations between power and capability are influenced by δ and ϵ, and thus we see that all three parameters γ, δ and ϵ can be

instantiated with different values, depending on what importance the agent wants to give to the different factors. As in BRS and, as we shall see, also in ReGReT, the **eval** function may also be parametric, in which case there are even more options available for the agent to adapt its trust model to the situation.

7.3 ReGReT

ReGReT [30] attempts to define a comprehensive trust and reputation model, which takes many different types of information into account. The first two are *direct trust* and *information from other agents*, or reputation. However, it is the first model to consider the multifaceted aspect of trust by linking a trust evaluation to a behavior (or role). An agent is not simply evaluated, but rather a trust evaluation is an evaluation of an agent performing a specific behavior. To achieve this, ReGReT adds *ontological information* to the calculation. Finally, ReGReT considers the structure of the social network as a source of information about the relations between agents. While none of these individual issues were new, at the time ReGReT was presented it was the first system to incorporate all these different aspects into a single comprehensive trust model. As such it is one of the most influential trust and reputation models in existence.

7.3.1 Roles and the ontology

The ontological dimension is considered for both the calculation of direct trust and reputation. These values are calculated specifically for a single role. These roles are related through a role taxonomy. The calculation of direct trust and reputation is only done for so-called "atomic roles", which take a single aspect of an interaction into account, such as, for instance, the price of an item in an auction. These coincide with the leaves of the role taxonomy. For any interior role, the trust in an agent fulfilling that role is the weighted mean of the trust in that agent for each of the child nodes. For instance, in an electronic auction, a seller is evaluated based on the cost of an item and the delivery time. Thus, the trust in an agent based only on cost is calculated and similarly for delivery time. These are then aggregated using a weighted average to obtain the trust in that agent as a seller. The direct trust and reputation calculations below are thus the calculations for leaves of the role taxonomy.

7.3.2 Direct trust

ReGReT gives a clear description on how an agent can evaluate its own direct experiences with the trustee. It does this in terms of an *outcome*, which consists of two things: a contract between two agents and the resulting actions. It is represented as a tuple $o = \langle i, j, I, X^c, X^f, t\rangle$, where i is the evaluating agent, j is the target, t is the time when the contract was formed and I is a set of terms in an ontology that the contract is about. X^c is a vector with the agreed values of the contract for each issue in I. X^f is a similar vector, but with the actual values, after the contract is deemed "fulfilled". An agent's outcomes are stored in a part of the belief base, called the outcome database (ODB). The direct trust an agent has in the trustee is calculated directly from this ODB.

To perform this calculation, the outcomes need to be evaluated. For each atomic role there is a function $g_r : \mathbb{R}^n \times \mathbb{R}^n \to [-1, 1]$, where n is the length of the vectors X^c and X^f. This function is used to evaluate an outcome and returns an impression in the range $[-1, 1]$. The impressions from all outcomes are aggregated and this is the direct trust of i in j concerning role r

$$DT_{i\to j}(r) = \sum_{o\in ODB} f(now, time(o)) \cdot g_r(contract(o), fulfillment(o))$$

with $time(o) = t$, $contract(o) = X^c$ and $fulfillment(o) = X^f$ if we let $o = \langle i, j, I, X^c, X^f, t\rangle$. Moreover, $f(now, t) = \frac{f'(now,t)}{\sum_{j\in ODB} f'(now,time(j))}$, where $f' : \mathbb{R}^2 \to [0, 1]$ is a function to calculate the

decay factor for outcomes over time. Examples are $f'(x, y) = 0.5^{x-y}$ or $f'(x, y) = y/x$. We see the aggregation method is a weighted average, with the weight dependent on the time an interaction took place.

This gives the *value* of the direct trust, but ReGReT also uses the *reliability* of the calculation. For direct trust this is defined by two factors: the *number of outcomes factor* and the *outcome reputation deviation*. These encode the uncertainty from possibly having too few interactions to reliably predict the other's behavior and, respectively, the uncertainty from the variability in the outcomes. The reliability of direct trust, $DTRL_{i\to j}(r)$ is simply the multiplication of the reliability calculated from either factor.

7.3.3 Reputation

In addition to direct trust, information from other agents is taken into account. This is calculated in a number of manners, resulting in three different types of reputation: the *witness reputation*, a value giving the reputation according to information received from other agents, the *neighbourhood reputation*, calculated by considering the neighbours of the trustee in a social network and the *system reputation*: a default reputation based on the role played by the trustee.

Witness reputation is calculated in two steps: the first of these is to decide which witnesses' opinions to consider. ReGReT uses the topology of the social network to find the witnesses. The details of this analysis are not important to the further explanation of the model and we refer the reader to Sabater-Mir's work [30] for these. The output of the social network analysis is a set of witnesses who are asked for their opinion. To decide how reliable a witness is, ReGReT uses two systems. The first of these is simply the individual reputation of the witness. If the reliability of this reputation is too low (according to a threshold), then another metric is used, once again based on the structure of the social network, which they call *socialTrust*. ReGReT has a set of conditional rules. The antecedent specifies properties that can hold in the social network and the conclusion is a statement about the reliability of an agent's opinion. If the witness' position in the social network coincides with the condition in the antecedent, the conclusion defines the reliability of the witness' opinion.

Neighbourhood reputation is calculated in a similar way to the socialTrust metric. System reputation is used in case neither witness reputation nor neighbourhood reputation can be used and is a type of default reputation, which is specified for any agent with certain properties, such as the role it plays in a system or other information generally available. If no information is available at all, a default value is used.

As such, ReGReT has a list of reputation metrics, each considered less reliable than the next. Individual reputation is considered before witness reputation, which in its turn is considered before neighbourhood reputation or system reputation. This is achieved by considering the final metric reputation $R_{i\to j}(r) = \sum_{x\in W,N,S,D} \xi_x \cdot R_{i\overset{x}{\to} j}(r)$, where $R_{i\overset{x}{\to} j}(r)$ is the reputation type x, with W, N, S and D being shorthand for Witness, Neighbourhood, System and Default. The ξ_x are weights for this system which depend on the reliability of each metric, as defined by the functions $RL_{i\overset{W}{\to} j}$, $RL_{i\overset{N}{\to} j}$ and $RL_{i\overset{S}{\to} j}$. For the definition of these reliability functions we refer to Sabater-Mir's work [30]. The weights for the types of reputation are then defined as follows:

- $\xi_W = RL_{i\overset{W}{\to} j}(r)$
- $\xi_N = RL_{i\overset{N}{\to} j}(r) \cdot (1 - \xi_W)$
- $\xi_S = RL_{i\overset{S}{\to} j}(r) \cdot (1 - \xi_W - \xi_N)$
- $\xi_D = 1 - \xi_W - \xi_N - \xi_S$

It is easy to see that if the reliability of the witness reputation is high (near 1), then the weight for the aggregation of the other reputation types is low.

7.3.4 Combining direct trust and reputation

The final calculation step is to combine direct trust with reputation. This is done using the following formula:

$$Trust_{i\to j}(r) = DTRL_{i\to j}(r) \cdot DT_{i\to j}(r) + (1 - DTRL_{i\to j}(r)) \cdot R_{i\to j}(r)$$

Trust is a weighted sum of direct trust and reputation, with the reliability of the direct trust defining the weights. This calculation can be performed for any of the atomic roles defined in the role taxonomy. For an internal node, representing a non-atomic role, the trust in its children must be calculated first. The trust in an agent performing a non-atomic role is then the weighted mean as described in Section 7.3.1.

We refer to Sabater-Mir's work [30] for a full description of the algorithm.

7.3.5 Parametrization of ReGReT

ReGReT is the most comprehensive trust model we consider and it has many different parts about which an agent could reason. The first and most obvious of these, are the weights used to calculate trust for non-atomic roles. ReGReT considers roles in a similar manner to the way we have incorporated them into our system. The roles thus have a double function. Firstly the role taxonomy defines the structure for aggregating atomic roles into non-atomic roles. However our reasoning system allows for rules to be set up defining the importance of these child roles in the aggregation and thus influence the weight of the aggregation, which in ReGReT is defined statically. An additional improvement is that for a specific goal, or set of beliefs, the importance of the child roles might be changed and thus the weights can be defined dynamically, dependent on the situation of the agent.

The second place the agent may incorporate reasoning is in the weight functions of direct trust. Both the time-dependent weight and the role-dependent evaluation functions are undefined in the model and left for the implementation. The time-dependent weight may be parametrized similarly to the decay factor in either BRS or ForTrust. However, the role-dependent evaluation functions are more interesting: the definition in ReGReT is in terms of a single issue of an interaction and thus a one-dimensional comparison. We have taken the liberty of extending this to an arbitrary function g_r for any atomic role r that calculates the evaluation of an outcome. If, as in the original description, g_r is a one-dimensional comparison it is obvious that this single issue is the only important factor in the calculation of trust for role r. However, by converting this into a multi-dimensional comparison it should be possible, if the function g_r is parametric, to specify dynamically which issues of an outcome are important for role r dynamically.

Finally, in the social network analysis used for witness reputation and neighbourhood reputation, ReGReT defines a set of fuzzy conditional rules. Sabater-Mir explicitly states that these rules are hand-coded, but a better approach would be to automate the process. While our reasoning system does not allow for full automation, it does allow for a mechanism to adapt these rules.

8 Conclusion and Future Work

In this paper we propose a method for integrating trust models into a cognitive agent by opening the "black box of trust". By making the parameters of a trust model explicit an agent can proactively adapt their values and thus the trust model. The values of these parameters are expressions of the relative importance of different factors on the trust calculation. We introduce an explicit representation of these factors and a priority logic for representing their relative importance to each other. Priority rules link the agent's cognitive aspect, such as goals and beliefs, and the social aspect (role) with particular orderings of these factors. These, in turn, determine the value of the parameters. In this manner the trust calculation can be adapted to the cognitive and social dimensions of the agent system.

This paper gives a complete formalization of this method by incorporating the trust model into a multi-context system representation of the BDI framework. Additionally we illustrate how this general method can be applied to particular trust models (BRS, ForTrust and ReGReT). This illustration serves two purposes: the first is to demonstrate our method and the formalization we provide. The second is to provide a guide to perform this incorporation for other trust models.

We intentionally left out the details of the implementation of the $resolve$ and $instantiate$ functions, and left the design of the trust priority rules deliberately vague. The details of their implementation depends on the specific agent architecture and trust model the agent designer uses, as well as the domain in which the agent should function. Filling in such details is an important step, but in this paper we describe the *first* step: an abstract, declarative framework, describing a new way to integrate an agent's trust model into its reasoning system. In this work we have focused on using the trust model's parameters in order to adapt the model, however it can be imagined that adapting the computational process itself might be desirable. There is no reason why the priority system could not be connected to such computational processes instead of parameters, however we leave this, far more complex, adaptation as future work. In the long run, we intend to use the specification of the priorities not just for reasoning about trust, but rather to allow agents to argue about trust. The idea of arguing about the validity of trust evaluations was presented by Pinyol et al. [26], who use BDI+Repage to generate the arguments, which are used to achieve more reliable communication about trust evaluations. The argument serves to make the trust evaluation more convincing, by linking it to an agent's knowledge about the environment. While this is an exemplary tool for the communication of trust, a similar argument needs to be constructed for every communicated trust evaluation. If, instead, the agents could *change* their trust model, then agents could use argumentation to reach an agreement of what should form the support for a trust evaluation and adapt their trust models to coincide with this. In this way, future communicated evaluations could be accepted simply by virtue of having agreed on what a trust evaluation means. However, as explained above, the BDI+Repage model does not provide a method for such adaptation. The extended BDI framework we present in this paper does, and this use of argumentation to form agreements about trust models is an important venue of future work as a possible method for aligning trust [17].

Acknowledgements

This work is supported by the Generalitat de Catalunya grant 2009-SGR-1434 and the Spanish Ministry of Education in the Agreement Technologies project: CONSOLIDER-INGENIO 2010 (CSD2007-00022) and the CBIT project: TIN2010-16306. Additionally the authors would like to thank the anonymous reviewers for their feedback.

References

[1] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley, 2007.

[2] C. Burnett, T. J. Norman, and K. Sycara. Trust decision-making in multi-agent systems. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 115–120, Barcelona, Spain, 2011. AAAI Press.

[3] A. Casali. *On Intentional and Social Agents with Graded Attitudes*, volume 42 of *Monografies de l'Institut d'Investigació en Intel·ligencia Artificial*. Consell Superior d'Investigacions Científiques, 2008.

[4] C. Castelfranchi and R. Falcone. *Trust Theory: A Socio-cognitive and Computational Model*. Wiley, 2010.

[5] N. Criado, E. Argente, and V. Botti. Normative deliberation in graded bdi agents. In J. Dix and C. Witteveen, editors, *Proceedings of MATES'10*, volume 6251 of *LNAI*, pages 52–63. Springer, 2010.

[6] M. Dastani. 2apl: A practical agent programming language. *Journal on Autonomous Agents and Multi-Agent Systems*, 16:214–248, 2008.

[7] M. Dastani, A. Herzig, J. Hulstijn, and L. van der Torre. Inferring trust. In J. a. A. Leite and P. Torroni, editors, *Proceedings of Fifth Workshop on Computational Logic in Multi-agent Systems (CLIMA'04)*, volume 3487 of *LNCS*, pages 144–160, 2004.

[8] R. Falcone, G. Pezzulo, and C. Castelfranchi. *A Fuzzy Approach to a Belief-based Trust Computation*, volume 3577 of *LNAI*, pages 43–58. Springer, 2005.

[9] K. K. Fullam, T. B. Klos, G. Muller, J. Sabater-Mir, K. S. Barber, and L. Vercouter. The Agent Reputation and Trust (ART) testbed. In *Proc. of the 4th International Conference on Trust Management*, volume 3986 of *Lecture Notes in Computer Science*, pages 439–442. Springer, 2006.

[10] F. Giunchiglia and L. Serafini. Multilanguage hierarchical logics, or: how we can do without modal logics. *Artificial Intelligence*, 65:29–70, 1994.

[11] J. Golbeck. Combining provenance with trust in social networks for semantic web content filtering. In L. Moreau and I. Foster, editors, *Provenance and Annotation of Data (IPAW 2006)*, volume 4145 of *LNCS*, pages 101–108. Springer, 2006.

[12] D. Harel. *First-Order Dynamic Logic*, volume 68 of *Lecture Notes in Computer Science*. Springer, 1979.

[13] R. Hermoso, H. Billhardt, and S. Ossowski. Dynamic evolution of role taxonomies through multidimensional clustering in multiagent organizations. In *Proc. of EUMAS 2009*, 2009.

[14] J. F. Hübner, E. Lorini, A. Herzig, and L. Vercouter. From cognitive trust theories to computational trust. In *Proc. of the Twelfth Workshop "Trust in Agent Societies" at AAMAS '09*, pages 55–67, Budapest, Hungary, 2009.

[15] A. Jøsang and R. Ismail. The beta reputation system. In *Proceedings of the Fifteenth Bled Electronic Commerce Conference e-Reality: Constructing the e-Economy*, Bled, Slovenia, 2002.

[16] S. Joseph, C. Sierra, M. Schorlemmer, and P. Dellunde. Deductive coherence and norm adoption. *Logic Journal of the IGPL*, 18(1):118–156, 2010.

[17] A. Koster. Why does trust need aligning? In *Proc. of the Thirteenth Workshop "Trust in Agent Societies" at AAMAS '10,*, pages 125–136, Toronto, Canada, 2010. IFAAMAS.

[18] Y. Krupa, L. Vercouter, J. F. Hübner, and A. Herzig. Trust based evaluation of wikipedia's contributors. In *Engineering Societies in the Agents World X*, volume 5881 of *Lecture Notes in Computer Science*, pages 148–161. Springer, 2009.

[19] C.-J. Liau. Belief, information acquisition, and trust in multi-agent systems – a modal logic formulation. *Artificial Intelligence*, 149(1):31–60, 2003.

[20] E. Lorini and R. Demolombe. *From Binary Trust to Graded Trust in Information Sources: a Logical Perspective*, volume 5396 of *LNAI*, pages 205–225. Springer, 2008.

[21] J. J. Odell, P. Van Dyke, and M. Fleischer. The role of roles in designing effective agent organizations. In A. Garcia, C. Lucena, F. Zambonelli, A. Omicini, and J. Castro, editors, *Software Engineering for Large-Scale Multi-Agent Systems*, volume 2603 of *Lecture Notes in Computer Science*, pages 27–38. Springer, 2003.

[22] P. Omidyar. Ebay. http://www.ebay.com, retrieved September 26, 2008, 1995.

[23] S. Parsons, C. Sierra, and N. R. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.

[24] S. Parsons, Y. Tang, E. Sklar, P. McBurney, and K. Cai. Argumentation-basded reasoning in agents with varying degrees of trust. In *Proceedings of AAMAS'11*, pages 879–886, Taipei, Taiwan, 2011. IFAAMAS.

[25] I. Pinyol and J. Sabater-Mir. Pragmatic-strategic reputation-based decisions in bdi agents. In *Eighth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'09)*, pages 1001–1008, Budapest, Hungary, 2009.

[26] I. Pinyol and J. Sabater-Mir. An argumentation-based protocol for social evaluations exchange. In *Proceedings of The 19th European Conference on Artificial Intelligence (ECAI 2010)*, Lisbon, Portugal, 2010.

[27] I. Pinyol and J. Sabater-Mir. Computational trust and reputation models for open multi-agent systems: a review. *Artificial Intelligence Review*, In Press.

[28] A. S. Rao and M. P. Georgeff. Modeling rational agents within a bdi-architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484, San Mateo, CA, USA, 1991. Morgan Kaufmann.

[29] A. S. Rao and M. P. Georgeff. Bdi agents: From theory to practice. In *Proceedings of the First International Congress on Multi-Agent Systems (ICMAS'95)*, pages 312–319, San Francisco, USA, 1995. AAAI.

[30] J. Sabater-Mir. *Trust and Reputation for Agent Societies*, volume 20 of *Monografies de l'Institut d'Investigació en Intel·ligencia Artificial*. Consell Superior d'Investigacions Científiques, 2003.

[31] J. Sabater-Mir, M. Paolucci, and R. Conte. Repage: REPutation and imAGE among limited autonomous partners. *JASSS - Journal of Artificial Societies and Social Simulation*, 9(2), 2006.

[32] J. Sabater-Mir, C. Sierra, S. Parsons, and N. R. Jennings. Engineering executable agents using multi-context systems. *Journal of Logic and Computation*, 12(3):413–442, 2002.

[33] M. Schorlemmer. Term rewriting in a logic of special relations. In *AMAST'98*, volume 1546 of *Lecture Notes in Computer Science*, pages 178–198, 1998.

[34] M. Şensoy and P. Yolum. Ontology-based service representation and selection. *IEEE Transactions on Knowledge and Data Engineering*, 19(8):1102–1115, 2007.

[35] C. Sierra and J. Debenham. An information-based model for trust. In *Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 497–504, Utrecht, The Netherlands, 2005. ACM.

[36] W. T. L. Teacy, J. Patel, N. R. Jennings, and M. Luck. Travos: Trust and reputation in the context of inaccurate information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, 12(2):183–198, 2006.

[37] G. Vogiatzis, I. MacGillivray, and M. Chli. A probabilistic model for trust and reputation. In *Ninth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'10)*, pages 225–232, Toronto, Canada, 2010. IFAAMAS.

[38] X. Wang, J.-h. You, and L. Y. Yuan. Nonmonotonic reasoning by monotonic inferences with priority constraints. In *Nonmonotonic Extensions of Logic Programming*, volume 1216 of *LNAI*, pages 91–109, 1997.

[39] B. Yu and M. P. Singh. An evidential model of distributed reputation management. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 294–301, New York, NY, USA, 2002. ACM.

[40] G. Zacharia. Collaborative reputation mechanisms for online communities. Master's thesis, Massachusetts Institute of Technology, 2000.