Behaviour Regulation in Multi-agent Systems

AISB 2008 Proceedings Volume 4

AISB '08

GLoRiClass

AISB ABERDEEN - 2008

UNIVERSITY OF ABERDEEN

AISB 2008 Convention
Communication, Interaction and Social Intelligence

1st-4th April 2008

University of Aberdeen

Volume 4 :
Proceedings of the
AISB 2008 Symposium on Behaviour Regulation in Multi-agent Systems

Published by
**The Society for the Study of
Artificial Intelligence and
Simulation of Behaviour**

http://www.aisb.org.uk/convention/aisb08/

ISBN 1 902956 64 8

Contents

The AISB'08 Convention: Communication, Interaction and Social Intelligence

As the field of Artificial Intelligence matures, AI systems begin to take their place in human society as our helpers. Thus it becomes essential for AI systems to have sophisticated social abilities, to communicate and interact. Some systems support us in our activities, while others take on tasks on our behalf. For those systems directly supporting human activities, advances in human-computer interaction become crucial. The bottleneck in such systems is often not the ability to find and process information; the bottleneck is often the inability to have natural (human) communication between computer and user. Clearly such AI research can benefit greatly from interaction with other disciplines such as linguistics and psychology. For those systems to which we delegate tasks: they become our electronic counterparts, or agents, and they need to communicate with the delegates of other humans (or organisations) to complete their tasks. Thus research on the social abilities of agents becomes central, and to this end multi-agent systems have had to borrow concepts from human societies. This interdisciplinary work borrows results from areas such as sociology and legal systems. An exciting recent development is the use of AI techniques to support and shed new light on interactions in human social networks, thus supporting effective collaboration in human societies. The research then has come full circle: techniques which were inspired by human abilities, with the original aim of enhancing AI, are now being applied to enhance those human abilities themselves. All of this underscores the importance of communication, interaction and social intelligence in current Artificial Intelligence and Cognitive Science research.

In addition to providing a home for state-of-the-art research in specialist areas, the convention also aimed to provide a fertile ground for new collaborations to be forged between complementary areas. Furthermore the 2008 Convention encouraged contributions that were not directly related to the theme, notable examples being the symposia on "Swarm Intelligence" and "Computing and Philosophy".

The invited speakers were chosen to fit with the major themes being represented in the symposia, and also to give a cross-disciplinary flavour to the event; thus speakers with Cognitive Science interests were chosen, rather than those with purely Computer Science interests. Prof. Jon Oberlander represented the themes of affective language, and multimodal communication; Prof. Rosaria Conte represented the themes of social interaction in agent systems, including behaviour regulation and emergence; Prof. Justine Cassell represented the themes of multimodal communication and embodied agents; Prof. Luciano Floridi represented the philosophical themes, in particular the impact of society. In addition there were many renowned international speakers invited to the individual symposia and workshops. Finally the public lecture was chosen to fit the broad theme of the convention – addressing the challenges of developing AI systems that could take their place in human society (Prof. Aaron Sloman) and the possible implications for humanity (Prof. Luciano Floridi).

The organisers would like to thank the University of Aberdeen for supporting the event. Special thanks are also due to the volunteers from Aberdeen University who did substantial additional local organising: Graeme Ritchie, Judith Masthoff, Joey Lam, and the student volunteers. Our sincerest thanks also go out to the symposium chairs and committees, without whose hard work and careful cooperation there could have been no Convention. Finally, and by no means least, we would like to thank the authors of the contributed papers – we sincerely hope they get value from the event.

Frank Guerin & Wamberto Vasconcelos

The AISB'08 Symposium on Behaviour Regulation in Multi-Agent Systems

Multi-agent systems are a powerful problem solving paradigm, attacking a problem via the interactions of many discrete components. The interactions between agents provide the approach's strength, but they also pose a great challenge: detecting and preventing undesirable behaviour (and interactions between agents) is difficult. While the formal specification of allowable agent behaviour is possible, unintended effects and purposefully malicious agents mean that other approaches to regulating the behaviour of agents are needed.

Techniques such as trust and reputation mechanisms and offline and online mechanism design (examples of which are social laws and machine interpretable contracts respectively) are able to regulate agent behaviour while allowing the agent to remain autonomous. While powerful, these techniques — and others — are still being refined, and many issues — including detecting and handling conflicting requirements, representation of permissible activities, and ways of monitoring and enforcing behaviour — remain. This symposium concerns itself with the theories, methodologies and computational issues related to regulating agent behaviour in multi-agent systems.

Out of the 17 papers submitted to the workshop, 10 papers were accepted for inclusion, an acceptance rate of 59%. Each paper was reviewed by at least three members of the programme committee. Paper presentations are limited to 20 minutes, with 10 minutes for questions. The workshop program also includes an invited talk by Javier Vázquez-Salceda (Universitat Politécnica de Catalunya), and a round table at the end of the day.

As can be seen by the breadth and scope of the submitted papers, behaviour regulation is a vibrant area of research. We hope that this will be the first of many successful symposia on the topic.

Of course, none of this would have been possible without the efforts of the BRMAS Programme Committee members in reviewing submitted papers, the authors in having submitted their work, and the participants for attending the symposium. Most importantly, perhaps, the organisers of the AISB convention as a whole must be recognised for dealing with all the other organisational aspects of the symposium and allowing us to focus on the technical issues.

Nir Oren and Michael Luck
BRMAS Organizers, King's College London, U.K.

Modelling MAS with Finite Analytic Stochastic Processes

Luke Dickens, Krysia Broda and **Alessandra Russo**[1]

Abstract. The Multi-Agent paradigm is becoming increasingly popular as a way of capturing complex control processes with stochastic properties. Many existing modelling tools are not flexible enough for these purposes, possibly because many of the modelling frameworks available inherit their structure from single agent frameworks. This paper proposes a new family of modelling frameworks called FASP, which is based on *state encapsulation* and powerful enough to capture multi-agent domains. It identifies how the FASP is more flexible, and describes systems more naturally than other approaches, demonstrating this with a number of robot football (soccer) formulations. This is important because more natural descriptions give more control when designing the tasks, against which a group of agents' collective behaviour is evaluated and regulated.

1 Introduction

Modelling stochastic processes is a time consuming and complicated task that involves many issues of concern. Among these, some that may appear high on most modellers' list are highlighted below. What type of language to use for building the model is often among the first decisions to make. Models are often finite state but they represent real-life continuous domains, so approximations are needed. A common problem is deciding what approximations to make: it is often difficult to determine how drastically a model will suffer in terms of results from making choice A rather than choice B; moreover if the language is restrictive or arcane it can hamper further. Furthermore, the modeller may wish to relate small problems to more complex tasks, especially if they want to partially solve or learn solutions on the simpler systems, and then import them to larger and more complicated ones[2]. A related issue is one of tuning — a system may have some parameters which are hard to anticipate; instead experimentation and incremental changes might be needed, so a language which supports natural representations with tunable features is desirable. Additionally, it is preferable to have as many analytic and learning tools available as is realistically possible. This means that the modelling framework should either support these directly or allow models to be transformed into frameworks which support them. Finally, single agent control paradigms may not be sufficient, nor may single real valued measures for evaluation; the modern experimenter may want more flexibility; potentially using multiple non-cooperative, isolated, agents each with a number of orthogonal constraints.

This paper highlights some of the issues above, and aims to address them by proposing a family of modelling frameworks known as Finite Analytic Stochastic Processes (FASP). We will demonstrate: how the multi-agent scenarios alluded to above can be described within this family; the naturally descriptive nature of the representations thus produced; and which real world applications might benefit. We highlight some of the features of this family by examining an often visited problem domain in the literature, that of robot soccer[3]. To do this, we introduce a relatively simple problem domain first developed by Littman [16]. We will show how this can be rewritten in the FASP framework in a number of ways, with each one giving more flexibility to re-tune in order to examine certain implicit choices of Littman. Then we will examine some other more recent robot soccer models from the related literature, which extend and add more interest to Littman's original formulation. We will show that these can also be represented by the FASP family, and will show how our more flexible framework can be exploited here. Ultimately, we generate a tunable turn-based non-cooperative solution, which can be scaled up to finer granularity of pitch size and a greater number of players with simultaneous team and independent objectives.

We finish with a discussion of the added value provided by the FASP languages, who might benefit from them, and some of the new challenges this presents to the reinforcement learning community (amongst others). The guiding principle here is that richer domains and evaluation – provided by better modelling tools, allows us to regulate agent and group behaviour in more subtle ways. It is expected that advanced learning techniques will be required to develop strategies concordant with these more sophisticated demands. This paper is a stepping stone on that longer journey.

2 Preliminaries

This section introduces the concept of Stochastic Map and Function, and uses them to construct a modelling framework for stochastic processes. Initially, we formalise a way of probabilistically mapping from one finite set to another.

Definition 1 (Stochastic Map) *A stochastic map m, from finite* independent *set X to finite* dependent *set Y, maps all elements in X probabilistically to elements in Y, i.e. $m : X \to \mathrm{PD}(Y)$, where $\mathrm{PD}(Y)$ is the set of all probability distributions over Y. The set of all such maps is $\Sigma(X \to Y)$; notationally the undecided probabilistic outcome of m given x is $m(x)$ and the following shorthand is defined $\Pr(m(x) = y) = m(y|x)$. Two such maps, $m_1, m_2 \in \Sigma(X \to Y)$, identical for each such conditional probability, i.e. $(\forall x, y)\ (m_1(y|x) = m_2(y|x))$, are said to be equal, i.e. $m_1 = m_2$.*

The stochastic map is a generalisation of the functional map, see [8]. Another method for generating outcomes probabilistically is the stochastic function, and relies on the concept of probability density function (PDF). For readers unfamiliar with the PDF, a good definition can be found in [19].

[1] Imperial College London, South Kensington Campus, UK, email: luke.dickens@imperial.ac.uk

[2] This step-wise training is sometimes called shaping, and is an active area of research, see [7, 14, 22]

[3] Or as we like to call it outside the US and Australia, robot football

Definition 2 (Stochastic Function) *A stochastic function f from finite* independent *set X to the set of real numbers $\mathbb{R}$, maps all elements in X probabilistically to the real numbers, i.e. $f : X \to \mathrm{PDF}(\mathbb{R})$. The set of all such functions is $\Phi(X \to \mathbb{R})$; notationally $f(x) = F_x$, where $F_x(y)$ is the $\mathrm{PDF}(\mathbb{R}))$ associated with x and belonging to f; $f(x)$ is also used to represent the undecided probabilistic outcome of F_x — it should be clear from the context which meaning is intended; the probability that this outcome lies between two bounds $\alpha_1, \alpha_2 \in \mathbb{R}$, $\alpha_1 < \alpha_2$, is denoted by $[f(x)]_{\alpha_1}^{\alpha_2}$. Two functions, $f, g \in \Phi(X \to \mathbb{R})$, identical for every PDF mapping, i.e. $(\forall x, \alpha_1, \alpha_2)([f(x)]_{\alpha_1}^{\alpha_2} = [g(x)]_{\alpha_1}^{\alpha_2})$, are said to be equal, i.e. $f = g$; if the two functions are inversely equal for each PDF mapping, i.e. $(\forall x, \alpha_1, \alpha_2)([f(x)]_{\alpha_1}^{\alpha_2} = [g(x)]_{-\alpha_2}^{-\alpha_1})$, then the stochastic functions are said to be inversely equal, i.e. $f = -g$.*

Armed with these two stochastic generators, we can formalise the Finite Analytic Stochastic Process (FASP), an agent oriented modelling framework.

Definition 3 (FASP) *A FASP is defined by the tuple $(S, A, O, t, \omega, F, i, \Pi)$, where the parameters are as follows; S is the finite state space; A is the finite action space; O is the finite observation space; $t \in \Sigma(S \times A \to S)$ is the transition function that defines the actions' effects on the system; $\omega \in \Sigma(S \to O)$ is the observation function that generates observations; $F = \{f^1, f^2, \ldots, f^N\}$ is the set of measure functions, where for each i, $f^i \in \Phi(S \to \mathbb{R})$ generates a real valued measure signal, f_n^i, at each time-step, n; $i \in \Sigma(\varnothing \to S)$ is the initialisation function that defines the initial system state; and Π is the set of all control policies available.*

Broadly similar to MDP style constructions, it can be used to build models representing agent interactions with some environmental system, in discrete time-steps. The FASP allows an observation function, which probabilistically generates an observation in each state, and multiple measure signals — analogous to the MDP's reward signal, which generate a measure signal at each state. One important feature of the FASP is that it generates observations and measures from state information alone. There is no in-built interpretation of the measure functions, their meaning and any preferred constraints on their output are left to be imposed by the modeller.

Care needs to be taken when interpreting the measure signals, and setting the desired output. These can be considered separate reward functions for separate agents - see section Section 3, or conflicting constraints on the same agent. A FASP does not have a natively implied solution (or even set of solutions): there can be multiple policies that satisfy an experimenters constraints; or there may be none — this is a by-product of the FASP's flexibility.

Fortunately, small problems (those that are analytically soluble) can be solved in two steps, first by finding the policy dependent state occupancy probabilities, and then solving the expected output from the measure functions. This means that measure function interpretation can be imposed late and/or different possibilities can be explored without having to resolve the state probabilities again, [8].

If we confine ourselves to a purely reactive policy space, i.e. where action choices are based solely on the most recent observation, hence $\Pi = \Sigma(O \to A)$, and a policy, $\pi \in \Pi$ is fixed, then the dynamics of this system resolves into a set of state to state transition probabilities, called the Full State Transition Function.

Definition 4 (FASP Full State Transition Function) *The FASP $M = (S, A, O, t, \omega, F, i, \Pi)$, with $\Pi = \Sigma(O \to A)$,has full state transition (FST) function $\tau_M : \Pi \to \Sigma(S \to S)$, where for $\pi \in \Pi$, $\tau_M(\pi) = \tau_M^\pi$ (written τ^π when M is clear), and, $\forall s, s' \in S$,*

$$\tau_M^\pi(s'|s) = \sum_{o \in O} \sum_{a \in A} \omega(o|s)\pi(a|o)t(s'|s, a)$$

It is the FST function that concisely defines the policy dependent stochastic dynamics of the system, each fixed policy identifying a Markov Chain, and together giving a policy labelled family of Markov Chains. A more detailed examination of FASPs (including the Multi-Agent derivatives appearing later in this paper) can be found in [8].

3 Multi-Agent Settings

This section shows how the FASP framework can be naturally extended to multi-agent settings. We distinguish two domains; models with simultaneous joint observations and subsequent actions by synchronised agents, and asynchronous agents forced to take turns, observing and acting on the environment. To model the first situation, we define the action space as being a tuple of action spaces — each part specific to one agent, similarly the observation space is a tuple of agent observation spaces. At every time-step, the system generates a joint observation, delivers the relevant parts to each agent, then combines their subsequent action choices into a joint action which then acts on the system. Any process described as a FASP constrained in such a way is referred to as a Synchronous Multi-Agent (SMA)FASP.

Definition 5 (Synchronous Multi-Agent FASP) *A Synchronous multi-agent FASP (SMAFASP), is a FASP, with the set of enumerated agents, G, and the added constraints that; the action space A, is a Cartesian product of action subspaces, A^g, for each $g \in G$, i.e. $A = \times_{g \in G} A^g$; the observation space O, is a Cartesian product of observation subspaces, O^g, for each $g \in G$, i.e. $O = \times_{g \in G} O^g$; and the policy space, Π, can be rewritten as a Cartesian product of sub-policy spaces, Π^g, one for each agent g, i.e. $\Pi = \times_{g \in G} \Pi^g$, where Π^g generates agent specific actions from A^g, using previous such actions from A^g and observations from O^g.*

To see how a full policy space might be partitioned into agent specific sub-policy spaces, consider a FASP with purely reactive policies; any constrained policy $\pi \in \Pi\,(= \Sigma(O \to A))$, can be rewritten as a vector of sub-policies $\pi = (\pi^1, \pi^2, \ldots, \pi^{|G|})$, where for each $g \in G$, $\Pi^g = \Sigma(O^g \to A^g)$. Given some vector observation $o_i \in O$, $\vec{o}_i = (o_i^1, o_i^2, \ldots, o_i^{|G|})$, and vector action $\vec{a}_j \in A$, $a_j = (a_j^1, a_j^2, \ldots, a_j^{|G|})$, the following is true,

$$\pi(\vec{a}_j|\vec{o}_i) = \prod_{g \in G} \pi^g(a_j^g|o_i^g)$$

In all other respects $\vec{o}_i$ and $\vec{a}_j$ behave as an observation and an action in a FASP. The FST function depends on the joint policy, otherwise it is exactly as for the FASP. Note that the above example is simply for illustrative purposes, definition 5 does not restrict itself to purely reactive policies. Many POMDP style multi-agent examples in the literature could be formulated as Synchronous Multi-Agent FASPs (and hence as FASPs), such as those found in [6, 12, 13, 20], although the formulation is more flexible, especially compared to frameworks that only allow a single reward shared amongst agents, as used in [20].

In real world scenarios with multiple rational decision makers, the likelihood that each decision maker chooses actions in step with every other, or even as often as every other, is small. Therefore it is natural to consider an extension to the FASP framework to allow each agent to act independently, yielding an Asynchronous Multi-Agent Finite Stochastic Process (AMAFASP). Here agents' actions affect the system as in the FASP, but any pair of action choices by two different agents are strictly non-simultaneous, each action is either before or after every other. This process description relies on the FASPs state encapsulated nature, i.e. all state information is accessible to all agents, via the state description.

Definition 6 (AMAFASP) *An AMAFASP is the tuple* $(S, G, A, O, t, \omega, F, i, u, \Pi)$*, where the parameters are as follows;* S *is the state space;* $\{G\}g_1, g_2, \ldots, g_{|G|}$ *is the set of agents;* $A = \bigcup_{g\in G} A^g$ *is the action set, a disjoint union of agent specific action spaces;* $O = \bigcup_{g\in G} O^g$ *is the observation set, a disjoint union of agent specific observation spaces;* $t \in \Sigma(S \times A \to S)$ *is the transition function and is the union function* $t = \bigcup_{g\in G} t^g$*, where for each agent* g*, the agent specific transition function* $t^g \in \Sigma((S \times A^g) \to S)$ *defines the effect of each agent's actions on the system state;* $\omega \in \Sigma(S \to O)$ *is the observation function and is the union function* $\omega = \bigcup_{g\in G} \omega^g$*, where* $\omega^g \in \Sigma(S \to O^g)$ *is used to generate an observation for agent* g*;* $\{F\}f^1, f^2, \ldots, f^N$ *is the set of measure functions;* $i \in \Sigma(\varnothing \to S)$ *is the initialisation function as in the FASP;* $u \in \Sigma(S \to G)$ *is the turn taking function; and* $\Pi = \times_{g\in G} \Pi^g$ *is the combined policy space as in the SMAFASP.*

A formal definition of the union map and union function used here, can be found in [8], for simplicity these objects can be thought of as a collection of independent stochastic maps or functions.

As with the FASP and SMAFASP, if we consider only reactive policies, i.e. $\Pi^g = \Sigma(O^g \to A^g)$ for each g, it is possible to determine the probability of any state-to-state transition as a function of the joint policy, and hence write full state transition function.

Definition 7 (AMAFASP Full State Transition) *An AMAFASP,* M*, with* $\Pi^g = \Sigma(O^g \to A^g)$ *for each* g*, has associated with it a full state transition function* τ_M*, given by,*

$$\tau^{\pi}_M(s'|s) = \sum_{g\in G} \sum_{o^g \in O^g} \sum_{a^g \in A^g} u(g|s)\, \omega^g(o^g|s)\, \pi^g(a^g|o^g)\, t^g(s'|s, a^g)$$

To our knowledge, there are no similar frameworks which model asynchronous agent actions within stochastic state dependent environments. This may be because without state encapsulation it would not be at all as straightforward. A discussion of the potential benefits of turn-taking appears in Appendix A. From this point on, the umbrella term FASP covers FASPs, SMAFASPs and AMAFASPs.

The multi-agent FASPs defined above allow for the full range of cooperation or competition between agents, dependent on our interpretation of the measure signals. This includes general-sum games, as well as allowing hard and soft requirements to be combined separately for each agent, or applied to groups. Our paper focuses on problems where each agent, g, is associated with a single measure signal, f^g_n at each time-step n. Without further loss of generality, these measure signals are treated as rewards, $r^g_n = f^g_n$, and each agent g is assumed to prefer higher values for r^g_n over lower ones[4].

[4] It would be simple to consider cost signals rather than rewards by inverting the signs

For readability we present the general-sum case first, and incrementally simplify.

The general-sum scenario considers agents that are following unrelated agendas, and hence rewards are independently generated.

Definition 8 (General-Sum Scenario) *A FASP is general-sum, if for each agent* g *there is a measure function* $f^g \in \Phi(S \to \mathbb{R})$*, and at each time-step* n *with the system in state* s_n*,* g*'s reward signal* $r^g_n = f^g_n$*, where each* f^g_n *is an outcome of* $f^g(s_n)$*, for all* g*. An agent's measure function is sometimes also called its reward function, in this scenario.*

Another popular scenario, especially when modelling competitive games, is the zero-sum case, where the net reward across all agents at each time-step is zero. Here, we generate a reward for all agents and then subtract the average.

Definition 9 (Zero-Sum Scenario) *A FASP is zero-sum, if for each agent* g *there is a measure function* $f^g \in \Phi(S \to \mathbb{R})$*, and at each time-step* n *with the system in state* s_n*,* g*'s reward signal* $r^g_n = f^g_n - \bar{f}_n$*, where each* f^g_n *is an outcome of* $f^g(s_n)$ *and* $\bar{f}_n = \sum_h f^h_n \big/ |G|$*.*

With deterministic rewards, the zero-sum case can be achieved with one less measure than there are agents, the final agent's measure is determined by the constraint that rewards sum to 1 (see [4]). For probabilistic measures with more than two agents, there are subtle effects on the distribution of rewards, so to avoid this we generate rewards independently.

If agents are grouped together, and within these groups always rewarded identically, then the groups are referred to as teams, and the scenario is called a team scenario.

Definition 10 (Team Scenario) *A FASP is in a team scenario, if the set of agents* G *is partitioned into some set of sets* $\{G_j\}$*, so* $G = \bigcup_j G_j$*, and for each* j*, there is a team measure function* f^j*. At some time-step* n *in state* s_n*, each* j*'s team reward* $r^j_n = f^j_n$*, where* f^j_n *is an outcome of* $f^j(s_n)$*, and for all* $g \in G_j$*,* $r^g_n = r^j_n$*.*

The team scenario above is general-sum, but can be adapted to a zero-sum team scenario in the obvious way. Other scenario's are possible, but we restrict ourselves to these. It might be worth noting that the team scenario with one team (modelling fully cooperative agents) and the two-team zero-sum scenario, are those most often examined in the associated multi-agent literature, most likely because they can be achieved with a single measure function and are thus relatively similar to the single agent POMDP, see [1, 9, 10, 11, 15, 16, 18, 23, 24, 25].

4 Examples

This section introduces the soccer example originally proposed in [16], and revisited in [2, 3, 5, 20, 24]. We illustrate both the transparency of the modelling mechanisms and ultimately the descriptive power this gives us in the context of problems which attempt to recreate some properties of a real system. The example is first formulated below as it appears in Littman's paper [16], and then recreated in two different ways.

Formulation 1 (Littman's adversarial MDP soccer) *An early model of MDP style multi-agent learning and referred to as soccer, the game is played on a* 4×5 *board of squares, with two agents (one of whom is holding the ball) and is zero-sum, see Fig. 1. Each*

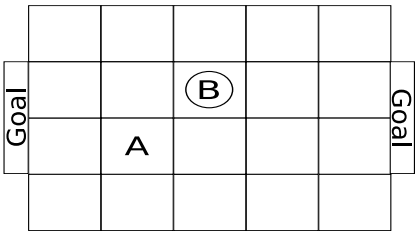
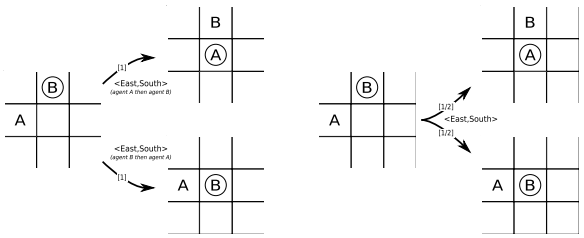


Figure 1. The MDP adversarial soccer example from [16].

*agent is located at some grid reference, and chooses to move in one of the four cardinal points of the compass (*N, S, E *and* W*) or the* H*(old) action at every time-step. Two agents cannot occupy the same square. The state space, S_{L1}, is of size $20 \times 19 = 380$, and the joint action space, A_{L1} is a cartesian product of the two agents action spaces, $A^A_{\mathrm{L1}} \times A^B_{\mathrm{L1}}$, and is of size $5 \times 5 = 25$. A game starts with agents in a random position in their own halves. The outcome for some joint action is generated by determining the outcome of each agent's action separately, in a random order, and is deterministic otherwise. An agent moves when unobstructed and does not when obstructed. If the agent with the ball tries to move into a square occupied by the other agent, then the ball changes hands. If the agent with the ball moves into the goal, then the game is restarted and the scoring agent gains a reward of $+1$ (the opposing agent getting -1).*

Therefore, other than the random turn ordering the game mechanics are deterministic.



(a) (b)

Figure 2. If agent A chooses to go East and agent B chooses to go South when diagonally adjacent as shown, the outcome will be non-deterministic depending on which agent's move is calculated first. In the original Littman version this was achieved by resolving agent specific actions in a random order, fig. (a). In the SMAFASP version this is translated into a flat transition function with probabilities on arcs (square brackets), fig. (b).

The Littman soccer game can be recreated as a SMAFASP, with very little work. We add a couple more states for the reward function, add a trivial observation function, and flatten the turn-taking into the transition function.

Formulation 2 (The SMAFASP soccer formulation) *The SMAFASP formulation of the soccer game, is formed as follows: the state space $S_{\mathrm{L2}} = S_{\mathrm{L1}} + s^\star_A + s^\star_B$, where $s^\star_g$ is the state immediately after g scores a goal; the action space $A_{\mathrm{L2}} = A_{\mathrm{L1}}$; the observation space $O_{\mathrm{L2}} = S_{\mathrm{L2}}$; the observation function is the identity mapping; and the measure/reward function for agent A, $f^A \in \Phi(S_{\mathrm{L2}} \to \mathbb{R})$ is as follows,*

$$f^A(s^\star_A) = 1, \quad f^A(s^\star_B) = -1, \quad \text{and } f^A(s) = 0 \text{ for all other } s \in S.$$

Agent B's reward function is simply the inverse, i.e. $f^B(s) = -f^A(s)$, for all s.

To define the transition function we need first to imagine that Littman's set of transitions were written as agent specific transition functions, $t^g\mathrm{L1} \in \Sigma(S_{\mathrm{L1}} \times A_{\mathrm{L1}} \to S_{\mathrm{L1}})$ for each agent g — although this is not explicitly possible within the framework he uses, his description suggests he did indeed do this. The new transition function, $t_{\mathrm{L2}} \in \Sigma(S_{\mathrm{L2}} \times A_{\mathrm{L2}} \to S_{\mathrm{L2}})$, would then be defined, for all $s_n, s_{n+1} \in S_{\mathrm{L1}}$, $a^A_n \in A^A_{\mathrm{L2}}$, $a^B_n \in A^B_{\mathrm{L2}}$, in the following way,

$$t_{\mathrm{L2}}(s_{n+1}|s_n,(a^A_n,a^B_n)) = \frac{1}{2}\cdot\sum_{s\in S_{\mathrm{L1}}}\left(\begin{array}{l} t^A_{\mathrm{L1}}(s|s_n,a^A_n).t^B_{\mathrm{L1}}(s_{n+1}|s,a^B_n) \\ \quad +t^B_{\mathrm{L1}}(s|s_n,a^B_n).t^A_{\mathrm{L1}}(s_{n+1}|s,a^A_n)\end{array}\right).$$

The transition probabilities involving the two new states, $s^\star_A$ and $s^\star_B$, would be handled in the expected way.

The turn-taking is absorbed so that the random order of actions within a turn is implicit within the probabilities of the transition function, see Fig. 2(b), rather than as before being a product of the *implicit* ordering of agent actions, as in Fig. 2(a).

It is possible to reconstruct Littman's game in a more flexible way. To see how, it is instructive to first examine what Littman's motives may have been in constructing this problem, which may require some supposition on our part. Littman's example is of particular interest to the multi-agent community, in that there is no independently optimal policy for either agent; instead each policy's value is dependent on the opponent's policy — therefore each agent is seeking a policy referred to as the *best response* to the other agent's policy. Each agent is further limited by Littman's random order mechanism, see Fig. 2(a), which means that while one agent is each turn choosing an action based on current state information, in effect the second agent to act is basing its action choice on state information that is one time-step off current; and because this ordering is random, neither agent can really rely on the current state information. Littman doesn't have much control over this turn-taking, and as can be seen from the SMAFASP formulation, the properties of this turn-taking choice can be incorporated into the transition function probabilities (see Fig. 2 (b)). Different properties would lead to different probabilities, and would constitute a slightly different system with possibly different solutions.

However, consider for example, that an agent is close to their own goal defending an attack. Its behaviour depends to some degree on where it expects to see its attacker next: the defender may wish to wait at one of these positions to *ambush* the other agent. The range of these positions is dependent on how many turns the attacker might take between these observations, which is in turn dependent on the turn-taking built into the system. For ease of reading, we introduce an intermediate AMAFASP formulation. The individual agent action spaces are as in the SMAFASP, as is the observation space, but the new state information is enriched with the positional states of the previous time-step, which in turn can be used to generate the observations for agents.

Formulation 3 (The first AMAFASP soccer formulation) *This AMAFASP formulation of the soccer game, M_{L3}, is formed as follows: the new state space $S_{\mathrm{L3}} = S_{\mathrm{L2}} \times S_{\mathrm{L2}}$, so a new state at some time-step n, is given by the tuple (s_n, s_{n-1}), where $s_{n-1}, s_n \in S_{\mathrm{L2}}$ and records the current and most recent positional states; there are two action space, one for each agent, $A^A_{\mathrm{L3}} = A^A_{\mathrm{L2}}$ and $A^A_{\mathrm{L3}} = A^A_{\mathrm{L2}}$; and two identical agent specific observation*

spaces, $O^A_{\mathrm{L3}} = O^B_{\mathrm{L3}} = O_{\mathrm{L2}}$; the new agent specific transition functions, $t^g_{\mathrm{L3}} \in \Sigma(S_{\mathrm{L3}} \times A^g_{\mathrm{L3}} \rightarrow S_{\mathrm{L3}})$, are defined, for all $s_{n-1}, s_n, s'_n, s_{n+1} \in S_{\mathrm{L2}}$, $a^g_n \in A^g_{\mathrm{L3}}$, in the following way:

$$t^g_{\mathrm{L3}}((s_{n+1}, s'_n)|(s_n, s_{n-1}), a^g_n) = \begin{cases} t^g_{\mathrm{L1}}(s_{n+1}|s_n, a^g_n) & \textit{iff } s'_n = s_n, \\ 0 & \textit{otherwise}. \end{cases}$$

where t^g_{L1} represents agent g's deterministic action effects in Littman's example, as in Formulation 2. The goal states, $s^\star_A$ and $s^\star_B$, are dealt with as expected.

Recalling that $O_{\mathrm{L3}} = S_{\mathrm{L2}}$, the observation function, $\omega^g_{\mathrm{L3}} \in \Sigma(S_{\mathrm{L3}} \rightarrow O_{\mathrm{L3}})$, is generated, for all $(s_{n-1}, s_n) \in S_{\mathrm{L3}}$, $o_n \in O^g_{\mathrm{L3}}$, and $g \in \{A, B\}$, in the following way,

$$\omega^g_{\mathrm{L3}}(o_n|(s_n, s_{n-1})) = \begin{cases} 1 & \textit{iff } s_{n-1} = o_n, \\ 0 & \textit{otherwise}. \end{cases}$$

The reward function is straightforward and left to the reader.

Finally, we construct the turn-taking function $u_{\mathrm{L3}} \in \Sigma(S_{\mathrm{L3}} \rightarrow \{A, B\})$, which simply generates either agent in an unbiased way at each time-step. The turn taking function is defined, for all $(s_n, s_{n-1}) \in S_{\mathrm{L3}}$, as

$$u_{\mathrm{L3}}(A|(s_n, s_{n-1})) = u_{\mathrm{L3}}(B|(s_n, s_{n-1})) = 1/2.$$

This does not fully replicate the Littman example, but satisfies the formulation in spirit in that agents are acting on potentially stale positional information, as well as dealing with an unpredictable opponent. In one sense, it better models hardware robots playing football, since *all* agents observe slightly out of date positional information, rather than a mix of some and not others. Both this and the Littman example do, however, share the distinction between turn ordering and game dynamics typified by Fig. 2 (a), what is more, this is now explicitly modelled by the turn-taking function.

To fully recreate the mix of stale and fresh observations seen in Littman's example along with the constrained turn-taking, we need for the state to include turn relevant information. This can be done with a tri-bit of information included with the other state information, to differentiate between; the start of a Littman time-step, when either agent could act next; when agent A has just acted in this time-step – and it must be B next; and vice versa when A must act next; we shall label these situations with l_0, l_B and l_A respectively. This has the knock on effect that in l_0 labelled states the observation function is as Formulation 2; in l_A and l_B labelled states the stale observation is used – as in Formulation 3. Otherwise Formulation 4 is very much like formulation Formulation 3.

Formulation 4 (The second AMAFASP soccer formulation) *This AMAFASP formulation of the soccer game, M_{L4}, is formed as follows: there is a set of turn labels, $L = \{l_0, l_A, l_B\}$; the state space is a three way Cartesian product, $S_{\mathrm{L4}} = S_{\mathrm{L2}} \times S_{\mathrm{L2}} \times L$, where the parts can be thought of as current-positional-state, previous-positional-state and turn-label respectively; the action spaces and observation spaces are as before, i.e. $A^g_{\mathrm{L4}} = A^g_{\mathrm{L3}}$, $O^g_{\mathrm{L4}} = O^g_{\mathrm{L3}}$, for each agent g; the transition and reward functions are straightforward and are omitted for brevity; the observation and turn taking functions are defined, for all $(s_n, s_{n-1}, l_n) \in S_{\mathrm{L4}}$, $o_n \in O^g_{\mathrm{L4}}$ and all agents g, in the following way,*

$$\omega^g_{\mathrm{L4}}(o_n|(s_n, s_{n-1}, l_n)) = \begin{cases} 1 & \textit{if } s_n = o_n \textit{ and } l_n = l_0, \\ 1 & \textit{if } s_{n-1} = o_n \textit{ and } l_n = u_g, \\ 0 & \textit{otherwise}. \end{cases}$$

and

$$u_{\mathrm{L4}}(g_n|(s_n, s_{n-1}, l_n)) = \begin{cases} \frac{1}{2} & \textit{if } g_n = g \textit{ and } l_n = l_0, \\ 1 & \textit{if } g_n = g \textit{ and } l_n = u_g, \\ 0 & \textit{otherwise} \end{cases}$$

The above formulation recreates Littman's example precisely, and instead of the opaque turn-taking mechanism hidden in the textual description of the problem, it is transparently and explicitly modelled as part of the turn-taking function.

So the Littman example can be recreated as a SMAFASP or AMAFASP, but more interestingly both AMAFASP formulations, 3 and 4, can be tuned or extended to yield new, equally valid, formulations. What is more, the intuitive construction means that these choices can be interpreted more easily.

Consider Formulation 3; the turn-taking function u can be defined to give different turn-taking probabilities at different states. For instance, if an agent is next to its own goal, we could increase its probability of acting (over the other agent being chosen) to reflect a defender behaving more fiercely when a loss is anticipated. Alternatively, if an agent's position has not changed since the last round, but the other's has then the first agent could be more likely to act (possible as two steps of positional data are stored); giving an advantage to the H(old) action, but otherwise encouraging a loose alternating agent mechanism.

While Formulation 4 recreates the Littman example, it again can be adjusted to allow different choices to the turn taking mechanism; in particular it is now possible to enforce strictly alternating agents. This would be done by flipping from state label l_A to l_B or vice versa, at each step transition, and otherwise keeping things very much as before. It is important to note that many specific models built in this way, can be recreated by implicit encoding of probabilities within existing frameworks, but it is difficult to see how the experimenter would interpret the group of models as being members of a family of related systems.

4.1 Flexible Behaviour Regulation

If we increase the number of players in our game, we can consider increasing the number of measure functions for a finer degree of control over desired behaviour. With just 2 agents competing in the Littman problem, it is difficult to see how to interpret any extra signals, and adding agents will increase the state space and hence the policy size radically. So, before we address this aspect of the FASP formalisms, it is useful to examine a more recent derivative soccer game, namely Peshkin et al.'s partially observable identical payoff stochastic game (POIPSG) version [20], which is more amenable to scaling up.

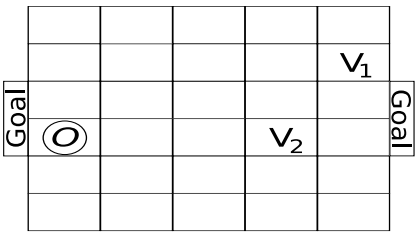


Figure 3. The POIPSG cooperative soccer example from [20].

Peshkin et al.'s example is illustrated in Fig. 3. There are two teammates, V_1 and V_2, and an opponent O, each agent has partial observability and can only see if the 4 horizontally and vertically adjacent squares are occupied, or not. Also, players V_1 and V_2 have an extra *pass* action when in possession of the ball. Otherwise the game is very much like Littman's with joint actions being resolved for individual agents in some random order at each time-step. Contrary to expectations, the opponent is not modelled as a learning agent and does not receive a reward; instead the two teammates share a reward and learn to optimise team behaviour versus a static opponent policy; for more details see [20].

As with its progenitor, the Peshkin example could be simply reworked as a SMAFASP in much the same way as in Formulation 2. Recreating it as an AMAFASP is also reasonably straightforward; as before, the trick of including the previous step's positional state in an AMAFASP state representation, allows us to generate *stale* observations – which are now also partial. As this is relatively similar to the Littman adaptions, the details are omitted. The focus here instead, is to show that, in the context of Peshkin's example, a zero- or general-sum adaption with more agents, could have some utility. Firstly, agent O could receive the opposite reward as the shared reward of V_1 and V_2, this could be done without introducing another measure function, merely reinterpreting Peshkin's reward function; now, the opponent could learn to optimise against the cooperative team. More interestingly, the players V_1 and V_2 could be encouraged to learn different roles by rewarding V_1 (say) more when the team scores a goal and penalising V_2 more when the team concedes one, all this requires is a measure function for each agent and a zero-sum correction. Further, we can add a second opponent (giving say O_1 and O_2), either rewarding them equally or encouraging different roles as with V_1 and V_2. In this way we could explore the value of different reward structures by competing the teams. If more agents were added, even up to 11 players a side, and a much larger grid, the AMAFASP framework supports a much richer landscape of rewards and penalties, which can encourage individual roles within the team, while still differentiating between good and bad team collaborations.

5 Discussion

In this paper, we have examined a new family of frameworks — the FASP, geared towards modelling stochastic processes, for one or many agents. We focus, on the multi-agent aspects of this family, and show how a variety of different game scenario's can be explored within this context. Further, we have highlighted the difference between synchronous and asynchronous actions within the multi-agent paradigm, and shown how these choices are explicit within the FASP frameworks. As a package, this delivers what we consider to be a much broader tool-set for regulating behaviour within cooperative, non-cooperative and competitive problems. This is in sharp contrast to the more traditional pre-programmed expert systems approaches, that attempt to prescribe agent intentions and interactions.

The overarching motivation for our approach is to provide the modeller with *transparent mechanisms*, roughly this means that all the pertinent mechanisms are defined explicitly within the model. There are two such mechanisms that are visited repeatedly in this paper, these are multiple measures and turn-taking, but they are not the only such mechanisms. In fact, the FASP borrows from the MDP, and POMDP, frameworks a few (arguably) transparent mechanisms. The transition function is a good example, and the observation and reward functions as they are defined in the POMDP have a degree of transparency; we argue that strict *state encapsulation* improves upon this though. The general agent-environment relationship is the same as the POMDP, meaning existing analytic and learning tools can still be applied where appropriate[5]. Moreover, techniques for shaping and incremental learning developed for related single agent frameworks [14, 22] and multi-agent frameworks [7] can also be applied without radical changes.

Other ideas for good transparent mechanisms exist in the literature, and the FASP family would benefit by incorporating the best of them. For instance, modelling extraneous actions/events can reduce the required size of a model, by allowing us to approximate portions of an overall system, without trying to explicitly recreate the entire system. Imagine a FASP with a external event function $X \in \Sigma(S \rightarrow S)$, which specifies how an open system might change from time-step to time-step, a similar extraneous event function can be found in [21]. This transition might occur between the observation and action outcomes, incorporating *staleness* into the observations as with our examples in Section 4. More interestingly, this function could be part of an AMAFASP, and be treated analogously to the action of a *null agent*; this would allow the turn-taking function to manage how rarely/often an extraneous events occurred. Another candidate for transparent mechanism, can be found in [4], where they simulate a broken actuator by separating intended action from actual action with what amounts to a stochastic map between the two. We would encourage modellers to incorporate such mechanisms as needed.

There are other clear directions for future work, the tools outlined in this paper enable a variety of multi-agent problems, with choice of measures for evaluation, but it leaves out how these measures might be used to solve or learn desirable solutions. The terms general- and zero-sum are not used accidentally, the similarities with traditional game theory are obvious, but the differences are more subtle. The extensive form game in the game theoretic sense, as described in [17], enforces that a game has a beginning and an end, only rewards at the end of a game run, players (agents) do not forget any steps in their game; and seeks to address the behaviour of intelligent players who have full access to the game's properties. While this can be defended as appropriate for human players, our paradigm allows for an ongoing game of synthetic agents with potentially highly restricted memory capabilities and zero prior knowledge of the game to be played; and rewards are calculated at every step. In fact, if we were to combine the AMAFASP with extraneous actions, it would constitute a generalisation of the extensive form game, as described in [17], but we omit the proof here.

It is sufficient to say that the FASP family demands a different solution approach than the extensive form game, and certainly a different approach than the easily understood reward-maximisation/cost-minimisation required by single-agent and cooperative MDP style problems. Some preliminary research has been done with respect to such systems, [4, 11, 26], we envisage this as being an active area of research in the next few years, and hope that the FASP tool-set facilitates that study.

REFERENCES

[1] Daniel S. Bernstein, Shlomo Zilberstein, and Neil Immerman, 'The complexity of decentralized control of markov decision processes', in *Proceedings of the 16th Annual Conference on Uncertainty in Artificial*

[5] Tools for evaluating the expected long term reward in a POMDPs, can be used without change within a FASP except that multiple measure signals could be evaluated simultaneously. Maximisation procedures are still possible too, but care needs to be taken with what is maximised and where multiple agents are maximising different measures independently [4, 26].

Intelligence (UAI-00), pp. 32–37, San Francisco, CA, (2000). Morgan Kaufmann.
[2] Reinaldo A. C. Bianchi, Carlos H. C. Ribeiro, and Anna H. Reali Costa, 'Heuristic selection of actions in multiagent reinforcement learning', in *IJCAI*, ed., Manuela M. Veloso, pp. 690–695, (2007).
[3] Michael Bowling, Rune Jensen, and Manuela Veloso, 'A formalization of equilibria for multiagent planning', in *Proceedings of the AAAI-2002 Workshop on Multiagent Planning*, (August 2002).
[4] Michael Bowling and Manuela Veloso, 'Existence of Multiagent Equilibria with Limited Agents', *Journal of Artificial Intelligence Research 22*, (2004). Submitted in October.
[5] Michael H. Bowling, Rune M. Jensen, and Manuela M. Veloso, 'Multiagent planning in the presence of multiple goals', in *Planning in Intelligent Systems: Aspects, Motivations and Methods*, John Wiley and Sons, Inc., (2005).
[6] Michael H. Bowling and Manuela M. Veloso, 'Simultaneous adversarial multi-robot learning', in *IJCAI*, eds., Georg Gottlob and Toby Walsh, pp. 699–704. Morgan Kaufmann, (2003).
[7] Olivier Buffet, Alain Dutech, and François Charpillet, 'Shaping multiagent systems with gradient reinforcement learning', *Autonomous Agents and Multi-Agent Systems*, **15**(2), 197–220, (2007).
[8] Luke Dickens, Krysia Broda, and Alessandra Russo, 'Transparent Modelling of Finite Stochastic Processes for Multiple Agents', Technical Report 2008/2, Imperial College London, (January 2008).
[9] Alain Dutech, Olivier Buffet, and Francois Charpillet, 'Multi-agent systems by incremental gradient reinforcement learning', in *IJCAI*, pp. 833–838, (2001).
[10] Jerzy Filar and Koos Vrieze, *Competitive Markov decision processes*, Springer-Verlag New York, Inc., New York, NY, USA, 1996.
[11] P. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multi-agent settings, 2004.
[12] Amy Greenwald and Keith Hall, 'Correlated q-learning', in *AAAI Spring Symposium Workshop on Collaborative Learning Agents*, (2002).
[13] Junling Hu and Michael P. Wellman, 'Multiagent reinforcement learning: theoretical framework and an algorithm', in *Proc. 15th International Conf. on Machine Learning*, pp. 242–250. Morgan Kaufmann, San Francisco, CA, (1998).
[14] Adam Daniel Laud, *Theory and Application of Reward Shaping in Reinforcement Learning*, Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2004. Advisor: Gerald DeJong.
[15] Martin Lauer and Martin Riedmiller, 'An algorithm for distributed reinforcement learning in cooperative multi-agent systems', in *Proc. 17th International Conf. on Machine Learning*, pp. 535–542. Morgan Kaufmann, San Francisco, CA, (2000).
[16] Michael L. Littman, 'Markov games as a framework for multi-agent reinforcement learning', in *Proceedings of the 11th International Conference on Machine Learning (ML-94)*, pp. 157–163, New Brunswick, NJ, (1994). Morgan Kaufmann.
[17] Roger B. Myerson, *Game Theory: Analysis of Conflict*, Harvard University Press, September 1997.
[18] Fans Oliehoek and Arnoud Visser, 'A Hierarchical Model for Decentralized Fighting of Large Scale Urban Fires', in *Proceedings of the Fifth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, eds., P. Stone and G. Weiss, Hakodate, Japan, (May 2006).
[19] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, McGraw Hill, 3rd edn., 1991.
[20] Leonid Peshkin, Kee-Eung Kim, Nicolas Meuleau, and Leslie P. Kaelbling, 'Learning to cooperate via policy search', in *Sixteenth Conference on Uncertainty in Artificial Intelligence*, pp. 307–314, San Francisco, CA, (2000). Morgan Kaufmann.
[21] Bharaneedharan Rathnasabapathy and Piotr Gmytrasiewicz. Formalizing multi-agent pomdp's in the context of network routing.
[22] Mark B. Ring, 'Child: A first step towards continual learning', *Machine Learning*, **28**, 77–104, (May 1997).
[23] Yoav Shoham, Rob Powers, and Trond Grenager, 'If multi-agent learning is the answer, what is the question?', *Artif. Intell.*, **171**(7), 365–377, (2007).
[24] William T. B. Uther and Manuela M. Veloso, 'Adversarial reinforcement learning.', Technical report, Computer Science Department, Carnegie Mellon University, (April 1997).
[25] Erfu Yang and Dongbing Gu, 'Multiagent reinforcement learning for multi-robot systems: A survey', Technical report, University of Essex, (2003).
[26] Martin Zinkevich, Amy Greenwald, and Michael Littman, 'Cyclic Equilibria in Markov Games', in *Advances in Neural Information Processing Systems 18*, 1641–1648, MIT Press, Cambridge, MA, (2005).

A The Benefits of Turn-Taking

Modelling with turn-taking is not only more natural in some cases, in certain cases it allows for a more concise representation of a system. Consider a system with 2 agents, A and B, and 9 positional states (in a 3×3 grid), where agents cannot occupy the same location (so there are $9 \times 8 = 72$ states), and movement actions are in the 4 cardinal directions for each agent.

Let us first consider how this might be formulated without explicit turn taking. Imagine, this is our prior-state,

$s_1 =$ 

Imagine also that agent A chooses action E(ast), agent B chooses action S(outh) — making joint action $<\mathrm{E},\mathrm{S}>$, and that these actions will be resolved in a random order. This results in one of three post-action states, with the following probabilities;

$$\Pr\left(s' = \right.$$ 

$$\left| s = \right.$$ 

$$\left., \ a =<\mathrm{E},\mathrm{S}>\right) = \alpha_1,$$

$$\Pr\left(s' = \right.$$ 

$$\left| s = \right.$$ 

$$\left., \ a =<\mathrm{E},\mathrm{S}>\right) = \beta_1,$$

and

$$\Pr\left(s' = \right.$$ 

$$\left| s = \right.$$ 

$$\left., \ a =<\mathrm{E},\mathrm{S}>\right) = \gamma_1,$$

where $\alpha_1 + \beta_1 + \gamma_1 = 1$.

Let's assume that they are transparently modelled by agents A and B, whose actions fail with the following probabilities; $\Pr(A\text{'s action fails}) = p$ and $\Pr(B\text{'s action fails}) = q$, acting in some order, where $\Pr(A \text{ is first}) = r$, and such that agents cannot move into an already occupied square. We can use these values to determine the *flattened* probabilities as $\alpha_1 = (1-p)\,(r + q(1-r))$, $\beta_1 = pq$, and $\gamma_1 = (1-q)\,(pr + (1-r))$. α_1, β_1 and γ_1 are not independent, they represent only 2 independent variables, but are fed into by p, q and r, which are independent. It may seem that α_1, β_1 and γ_1 model the situation more concisely, since any one combination of α_1, β_1 and γ_1 could correspond to many different choices for p, q and r. However, this isn't the whole story. Consider instead the prior-state s_2, where

$s_2 =$ 

,

and the combined action $<\mathrm{S},\mathrm{E}>$. Treated naively the possible outcomes might all be specified with separate probabilities (say α_2, β_2 and γ_2) as before. However, the modeller can, with transparent mechanisms, choose to exploit underlying symmetries of the system. If, as is quite natural, the turn ordering probabilities are independent of position, then there need be no extra parameters specified for this situation, the original p, q and r are already sufficient to define the transition probabilities here too.

Automated Mechanism Design Using Process Algebra

Emmanuel M. Tadjouddine[1]

Abstract. This paper shows how process algebra can be used to automatically generate verifiable mechanisms for multi-agent systems wherein agents need to trust the system. We make the link between games and process models and then present an iterative algorithm allowing us to generate mechanisms as computer programs implementing given systems' requirements, which are expressed as constraints and desirable properties such as incentive compatibility. This methodology can be used to deploy for example agent mediated e-commerce systems.

1 Introduction

As game theory is used to model and analyze multi-agent systems composed of selfish but rational agents, there is a need to tailor game rules according to given parameters of the underlying system. This is known as *Automated Mechanism Design* (AMD) [3, 23, 24]. For example, in agent mediated e-commerce, software agents may be engaged in financial transactions in which auction rules may be chosen to suit the features of the participants. It is not obvious to provide a mechanism that induces certain behaviour of the agents in such a scenario. For example, how do we prevent agents from colluding bearing in mind it is even difficult to detect collusion between the agents in the system. We therefore restrict ourselves to models of rationality and behaviour given by equilibrium concepts that are well studied in game theory mechanism design.

There are at least two ways of designing mechanisms automatically. First, we can rely on well-known hand-coded mechanisms to find new ones that satisfy the given system objectives and constraints by solving an optimization problem [3, 15, 23, 24]. Second, we can generate automatically verifiable mechanisms as computer programs in a logical framework.

This preliminary work adopts the latter approach and makes the following contributions:

- it relies on van Benthem's work presented in [22] and makes the connections between mechanism design and process algebra [9] by describing game mechanisms as process models in the modal μ-calculus [8]. This connection permits us to choose a process algebra language, e.g., Promela [7] and to automate the generation of verifiable game mechanisms given some desirable properties of the system.
- it presents an iterative algorithm that generates Promela programs representing mechanisms whose properties can be verified by the SPIN model checker [7] and illustrates the approach using a cake cutting protocol and a single item auction mechanism.

We stress on the importance of producing verifiable mechanisms by AMD since in a setting where agents are self-motivated, claimed mechanism properties must be checked by the participants in order to avoid cheating and to ensure trust in the system.

The remainder of this paper is organized as follows. Section 2 presents a brief introduction to mechanism design. Section 3 describes game mechanisms in a logical game calculus, provides a method in order to decide in two mechanisms are the same, and justifies the use of Promela as a process algebra language for describing games. Section 4 presents an iterative algorithm to generate Promela programs representing game mechanisms from a set of requirements i.e., the mechanism's objectives and constraints. Section 5 discusses the related work and Section 6 concludes.

2 Mechanism Design

Mechanism design, see for example [12, 11] aims to find a decision procedure that determines the outcome for a game according to some desired objective. An objective may be *incentive compatibility* (no agent can benefit from lying provided all other agents are truthful) or *strategy-proofness* (no agent can benefit from lying regardless of what its opponents do). In this section, we present two classes of mechanism we intend to study. The first concerns games with *complete information* (players' utility functions are common knowledge) and for the case of dynamic games we assume *perfect information* (each player knows the history of the game thus far), the second is related to auction design using the class of direct revelation mechanisms, see for example [11] for details.

Definition 1 *A mechanism design problem for n player games of complete and perfect information is defined by*

- *a finite set O of outcomes,*
- *an utility function $\mathbf{u}$ associating each allowed outcome $o \in O$ to a n-vector of real numbers $\mathbf{u}(o)$,*
- *a desirable property of the mechanism, e.g., total utility maximization or incentive compatibility.*

The aim is to find a function that selects outcomes for which the desired property is satisfied.

An example of such a mechanism is the cake cutting protocol wherein we need to share a cake between, say two agents. The protocol consists in asking one agent to cut the cake and let the other pick up its share first. This forces the first acting agent to cut the cake in equal pieces.

Definition 2 *A direct revelation mechanism design problem for n agents is described by*

- *a finite set O of outcomes,*
- *n types or valuation functions $v_i(o \in O), i = 1 \ldots n$, private to the agents,*

[1] Department of Computing Science, King's College, University of Aberdeen, Aberdeen AB24 3UE, Scotland, Email: etadjoud@csd.abdn.ac.uk

- n *quasi-linear utility functions* $u_i(o) = v_i(o) - p_i$ *wherein* p_i *depends on the course of play for agent* i*.*

The objective is to find (i) a function f*, which, given a vector* $\mathbf{v}$ *of declared valuations, returns an outcome* $f(\mathbf{v}) \in O$ *and (ii) a payment function* $\mathbf{p}(\mathbf{v}) = (p_1(\mathbf{v}), \ldots, p_n(\mathbf{v}))$ *such that reporting its true valuation is a dominant strategy.*

An example of such a mechanism is the well-known Vickrey auction and more generally the class of VCG mechanisms, see [11]. For the purpose of generating game mechanisms for a given logical property, we need a closer look at the connections between games, logic, and computer programs.

3 Games: Actions, Outcomes, Utilities and Equilibria

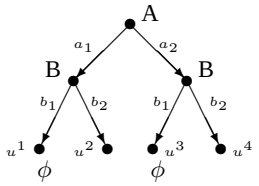
We first confined ourselves to games of complete and perfect information. Such games can be represented in *extensive form* as follows.

Definition 3 *The extensive form representation of a game with perfect and complete information is a tuple* $(T, P, (A_i)_{i\in P},$ *turn*$_{i\in P}$, *move*$_{i\in P}$, *end*, $(u_i)_{i\in P})$ *wherein*

- T *is a tree composed of an initial node, intermediate or decision nodes, and final nodes with no successors and edges labelled by players' actions,*
- P *is the set of players in the game,*
- $(turn_i)_{i\in P}$ *is a function which marks player* i*'s turn,*
- $(move_i)_{i\in P}$ *represents the set of actions or outgoing transitions for each player* i *at each decision node,*
- *end is a mark for final nodes,*
- $(u_i)_{i\in P}$ *are the utilities of the players at the final nodes of the tree* T*.*

Consider a simple game with two players A and B having the set of actions $\{a_1, a_2\}$ and $\{b_1, b_2\}$ respectively as described by Figure 1. Clearly, player B has a strategy forcing the outcome associated with

Figure 1. A two player game



the utility vectors u^1 or u^3. This can be viewed as a partial transition relation for players' turns by using conditionals of the form:

if A plays this, then B should play that.

Thus, the two players interact using strategies involving basic moves and tests for conditions. More generally, a strategy is a plan of actions; it is viewed as a computer program describing alternative sequence of actions that are taken at decision nodes for each player. A sequence of actions can be chosen so as to attain a certain outcome of the game. Each outcome is associated with a value (utility) and each player strives for maximal utility by means of competition or cooperation. In the case of non-cooperative games, the foundational result due to Nash [10] proves the existence of a mixed strategy *Nash equilibrium*, which describes a strategy profile by which no player has an incentive to deviate from it provided all its opponents stick to it. In general, the players have preferences, beliefs, or expectations about the game play and different equilibrium concepts can be found in the literature, see for example [2]. In short, a game is viewed as being composed of basic moves, relational operations such as choice, composition, and conditionals, and utilities involving real number arithmetic.

3.1 A Logical Game Calculus

The above description of a game can be carried out using modal $\mu-$calculus, a modal logic with fixed points introduced in [8]. Starting with the basic moves, the *turn* function and the *end* mark, we can add modal operators such as composition, choice, iteration, or test. Observe that the turn function describes the way players take turns in the game and can be concurrent allowing us for example to describe simultaneous moves in the game. For a given action a and a property ϕ, the modal operator $[a]\phi$ means the execution of a will necessarily make ϕ hold and $< a > \phi$ means the execution of a will possibly make ϕ hold. Obviously, we can have a set of actions in lieu of a single one in the above modal formulae.

Consider for example the game of Figure 1 wherein ϕ is a logical formula involving the players' utilities at the designated end points. Whatever the action of player A, player B has a clear strategy to make property ϕ holds. In modal logic, this is expressed by the formula, wherein $\cup$ represents the choice operator.

$$[a_1 \cup a_2] < b_1 \cup b_2 > \phi$$

We can also define a *winning strategy* for a player i using a recursive predicate $\mathtt{win}_i$. At the terminal nodes $\mathtt{win}_A$ indicates player A wins the game. At any other node, it indicates A has an action which will eventually make him the winner. This gives a recursive definition of the winning strategy. With recursion comes the question of termination. However, the modal μ-calculus provides us with a fixed point iteration as follows. For a given propositional variable p and a formula Φ, the expression $\mu p.\Phi$ represents the least fixed point of the function that maps the set of states where p is true to the set of states where Φ is true. The existence of this least fixed point is ensured by the Knaster-Tarski fixed point theorem [21] provided p occurs positively in Φ.

The winning strategy can then be expressed as follows:

$$\mu\mathtt{win}_A.(\text{end} \wedge \mathtt{win}_A) \vee (\text{turn}_A \wedge < a_1 \cup a_2 > \mathtt{win}_A) \vee (\text{turn}_B \wedge [b_1 \cup b_2]\mathtt{win}_A)$$

Notice that in any finite game, there is a player that has a winning strategy. If for example the strategy profile (a_1, b_2) is a Nash equilibrium, then we have the following formula:

$$\text{end} \wedge (\text{turn}_B \wedge [b_2](u_A^2 \geq u_A^4)) \wedge (\text{turn}_A \wedge [a_1](u_B^2 \geq u_B^1)),$$

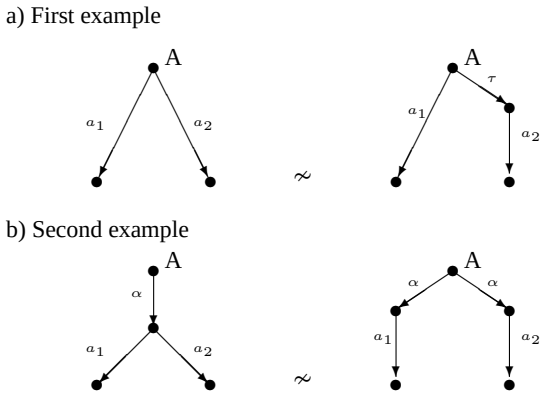
meaning that player A [B] gets a higher utility by playing a_1 [b_2] provided B [A] sticks to b_2 [a_1]. This logic permits us to reason on the interactions between players, their strategies as well as the game outcomes. Following [1], we can add atomic propositions at all end nodes similar to ϕ encoding preferences or expectations about the course of the game. This is important in mechanism design wherein we are given a desirable property about the course of the game or an objective function to optimize (e.g., social welfare maximisation) and strive to design the game rules so as to achieve those outcomes. At this level of description, a natural question is when are two mechanisms the same?

3.2 Game Mechanism Equivalence

In automated mechanism design, we may be interested in finding out if a newly created game mechanism is the same as a popular one. This is possible if we can compare two given game mechanisms. Because a game is made of processes (composed of basic moves, communication commands such as send or receive a message, relational operations such as choice, composition, iterations or conditionals), we need to compare processes. There is already a large body of literature on equivalences of two processes. Generally speaking, two processes are viewed to be equivalent if an external observer interacting with them cannot distinguish them. This is called *bisimulation* [22].

Let us consider the well-known examples in Figure 2 representing two simple one-player games. If we just care about inputs/outputs the

Figure 2. Two examples of one-player games not observationally equivalent



two games are the same. Because in mechanism design, the designer strives to give incentives to the players to behave in a desirable way, we need to have a closer look at the internals of the processes leading to the outcomes. We can consider the execution trace. The two games in Figure 2 a) do not have the same execution trace since in the right hand game, the player makes a move τ to reach a position where it can only performs action a_2. They cannot be equivalent. In Figure 2 b), the two games have the same execution trace $\{\alpha.a_1, \alpha.a_2\}$. They are the same under trace equivalence. However, the two games display different behaviour. In the left hand game of Figure 2 b), after the move α, the player is in a position to perform action a_1 or a_2 whereas in the right hand side, the move α takes the player to a position where it can only performs a_1 or a_2 exclusively. The notion of trace equivalence can be refined so as to account for these internal differences giving rise to the notion of bisimulation. Clearly these two examples display one-player games that are not bisimular. Following [22], we have adopted the stronger notion of bisimulation to define game mechanism equivalence. This can be justified by the fact that we are interested in studying behaviour in a given system. To this end, we need to pay a closer look to the internals of the system in lieu of viewing it as a black box whereby what is important is to ensure the same inputs lead to the same outputs in order to compare two mechanisms.

Considering a game as a bunch of concurrent and communicating processes in a competition or cooperation mode, the question is to find the game rules given a desirable property of the system. For that purpose, a game is generated as a computer program by an iterative procedure in a process modelling language or process algebra so that, at each iteration, the required property can be model checked.

3.3 Process Algebra

Process Algebras (PAs) are mathematical models for concurrent and communicating processes. There are various PAs among them the CCS [9] or Promela [7]. These languages are commonly composed of simple constructs, compositional and operational semantics, behavioural reasoning and observational equivalence. Although, a more generic framework for automatically generating verifiable mechanisms requires dealing with the beliefs of agents (typically in Bayesian games) and therefore the use of a *stochastic* PA [6], the algebra chosen in this paper is Promela [7]. We justify this choice by the fact that Promela is expressive enough for the presentation of our approach and the existence of a well established model checker, SPIN [7] to verify certain properties of the obtained mechanism.

Promela programs are composed of independent and parallel processes communicating through named channels. A process can send or receive a message e trough a channel c by performing actions $[c!e]$ or $[c?e]$ respectively. A process's body is a sequence of operations that can be declarations of typed constants or variables or statements (assignments, conditionals, loops, etc.) made of expressions that manipulate basic algebra terms, see [7] for more details.

The Promela process algebra has a well-defined semantics indicating how processes are executed from the inputs to the outputs in the form of transition system model. This semantics using *structural operational semantics* rules is detailed in [25]. Structural operational semantics rules present a formal way of defining semantics for PAs and other kind of operational semantics. Our aim is to generate well-defined semantics Promela programs representing game mechanisms for a given property of the system.

4 Automatic Generation of Games

This section presents an algorithm to generate Promela programs representing game mechanisms from a set of requirements i.e., the mechanism's objectives and constraints.

4.1 Mechanism Requirements

Mechanism requirements are given as a set of *declarative statements*. A declarative statement describes an identifier (variable or constant) along with its definition, its type, its IO (input or output) status, and its calculation condition.

For simplicity reasons, we use a *choose*$_S$ command, similar to that of [14], which can be modelled in Promela as a non-deterministic choice over the values in the set S. Table 1 gives an example of such requirements. One can see for example that the variable `fp` is an input that is used in the test conditions and a_1, a_2 represent actions taken by the two players. An action is simply a random choice over the set of integers from 0 to 10. Obviously, these requirements mimic constructs of a programming language. However, there is no need to specify the order in which the definitions must be executed; the control logic is left out of the requirements.

Given the requirements of Table 1, we aim to generate the mechanism so that the strategy profile $(a_1{=}5, a_2{=}5)$ is a Nash equilibrium by determining the parameters c_1, c_2 used in the calculation of the utilities u_1, u_2 of both players.

Table 1. Requirements for a design of a game with complete and perfect information

Identifier	Definition	Condition	Type	IO
fp			bit	IN
a_1	$\text{choose}_{\{0..10\}}$		byte	
a_2	$\text{choose}_{\{0..10\}}$		byte	
u_1	$10 + c_1 a_1$	$fp = 0$	int	OUT
u_2	$10 + c_2 a_2$	$fp = 1$	int	OUT
u_1	$10 - u_2$	$\neg(fp = 0)$	int	OUT
u_2	$10 - u_1$	$\neg(fp = 1)$	int	OUT

Table 2 represents a set of requirements for the design of an auction protocol wherein two agents value an item for sale at v_1 and v_2 and bid the numbers b_1 and b_2 respectively. The numbers b_1, b_2 are chosen in the integer interval $(0, 10)$. The two agents must pay a price p_1 and p_2 given in some parameterised form to get the item with associated utilities u_1 and u_2. The aim is to find the model parameters c_{11}, c_{12}, c_{21}, and c_{22} so that the strategy profile $(b_1{=}v_1, b_2{=}v_2)$ is a dominant strategy equilibrium for both agents.

Table 2. Requirements for a single item auction

Identifier	Definition	Condition	Type	IO
v_1			byte	IN
v_2			byte	IN
b_1	$\text{choose}_{\{0..10\}}$		byte	
b_2	$\text{choose}_{\{0..10\}}$		byte	
p_1	$c_{11}b_1 + c_{12}b_2$		int	
p_2	$c_{21}b_1 + c_{22}b_2$		int	
u_1	$v_1 - p_1$	$b_1 \geq b_2$	int	OUT
u_2	$v_2 - p_2$	$\neg(b_1 \geq b_2)$	int	OUT
u_1	0	$\neg(b_1 \geq b_2)$	int	OUT
u_2	0	$b_1 \geq b_2$	int	OUT

The first question is how to generate a Promela program given the mechanism requirements? This can be carried out using the following procedure:

1. each declarative statement of the requirements is translated to a Promela instruction.
2. we then construct the data dependency graph between the Promela instructions by analyzing the chain of definitions of the variables and their uses.
3. we find a valid control-flow of the resulting Promela code. It is reasonable to assume that each variable is assigned only once to avoid the case of a variable being overwritten. In this case, the Promela instructions must be reordered so that data dependencies are respected. Namely, we must ensure that no variable is used before being defined. For example, given the requirements in Table 2, the definition of the price p_1 must be executed before that of the utility u_1.

This procedure will enable us to generate a Promela code that has a well-defined semantics and that represents a game mechanism. Figure 3 and Figure 4 show two codes generated from the requirements of Table 1 and Table 2 in which the non Promela `input` command is used for clarity.

The second question is how to choose the parameters specified in the generated code so that the desired properties are satisfied? Notice that the parameter values $c_1{=}c_2{=}-1$ give a solution to the cake cutting protocol design. The values $c_{11}{=}c_{22}{=}0$ and $c_{12}{=}c_{21}{=}1$ give the Vickrey auction mechanism. The values $c_{11}{=}c_{22}{=}c_{12}{=}c_{21}{=}1/2$ give the modified Vickrey auction, which is better in terms of expected revenue for the auctioneer [2]. These parameters can be determined by an iterative procedure.

Figure 3. Mechanism generation for two agents: Cake cutting

```
input(fp);
if
  :: (fp==1 ) ->
     a1 = choose in 0..10;
     u1 = 10+c1*a1;
     u2 = 10-u1;
  :: (fp==2) ->
     a2 = choose in 0..10;
     u2 = 10+c2*a2;
     u1 = 10-u2;
  :: else -> skip;
fi;
```

Figure 4. Mechanism generation for two agents: Single item auction

```
input(v1); input(v2);
b1 = choose in 0..10;
b2 = choose in 0..10;
if
  :: (b1 >= b2) ->
     p1 = c11*b1+c12*b2;
     u1 = v1 - p1;
     u2 = 0;
  :: else ->
     p2 = c21*b1+c22*b2;
     u2 = v2 - p2;
     u1 = 0;
fi;
```

4.2 An Iterative Algorithm

To generate a mechanism from a set of requirements, we generate a computer code that has a well-defined semantics by the following iterative algorithm:

1. Generate a program that respects the definition-use chains in the Promela language as described in Section 4.1,
2. Choose random values for the model parameters,
3. Verify the required property for the obtained mechanism using the language associated model checker,
4. If the property is not satisfied, use a local search method to find new improved model parameters,
5. Repeat Step 2-4 until the property is satisfied or a maximum number of iterations is reached.

By construction, this algorithm always terminates and when it converges, the resulting mechanism satisfies the objective of the system. When the process is stopped because of attaining a maximum number of iterations, the obtained mechanism may be near enough to satisfying the system's objective. The choice of the Promela language is guided by the ability to use the SPIN model checker. This allows us to generate mechanisms that are guaranteed to satisfy the system requirements at convergence. However, using a model checker to verify desirable properties for multi-agent systems can lead to state space explosion for large models. To avoid this, we can use abstraction techniques, see for example [13, 19, 20]. A property-preserving abstraction map can be found so as to transform a detailed concrete domain into a less complex one; rewrite and check the property in the abstract model and deduce its validity in the concrete domain. Finding such an abstraction map is not straightforward, see for example [19].

Although this approach allows us to generate mechanisms guaranteed to have specified desirable properties, there is the scalability problem (how to deal with deal with large-scale models) due to the facts that:

- certain mechanisms may require large number of parameters,
- model checkers may suffer from state-space explosion,
- property preserving abstractions are hard to invent.

Nonetheless, this is a promising approach that deserves further investigation.

5 Related Work

We can only mention some items of the relevant literature. AMD, see for example [3], is a young topic motivated by the need to provide tailored computationally efficient mechanisms given the objectives and constraints of a system. An incremental approach to designing strategy-proof mechanisms is investigated in [4] and a framework for multistage mechanism design is discussed in [17]. In [23, 24], a general AMD framework is discussed. It relies on existing hand-coded mechanisms (e.g., Vickrey auction) to improve the desired objective of the mechanism by an iterative process. In [15], the focus is on improving the pricing rules for an auction mechanism by using evolutionary algorithms. To some extent, these automated mechanisms make some assumptions on the rules of the game and try to fit the objective and constraints by solving an optimization problem. In here, we aim to generate the entire function encoding the game rules and utilities so that we can verify the requirements of the system are satisfied.

Automated mechanisms using process algebra have been developed elsewhere, e.g., music composition [16] or software code generation given its requirements [5]. Our work builds on that of [22] in which extensive games are analyzed as process models using modal logic and bisimulation [22] and extends it to generating mechanisms using an approach similar to that of [5]. Moreover, a research project looking for a logical framework to specify and verify social choice mechanisms is described in [26]. In here, we use a process algebra language to automatically generate verifiable game mechanisms. Techniques for verifying game-theoretic properties of mechanisms are explored in [14, 18] but we adopted the SPIN model checking approach [7] for this study. The use of SPIN in verifying game equilibria is also explored in [19, 20].

6 Concluding Remarks

In this preliminary work, we have used van Benthem's description of games as process models to make the connection between mechanism design and process algebra. This allows us to choose a process algebra language and to automate the production of game mechanisms viewed as computer programs given some desirable properties. In particular, we present a novel algorithm based on this logical framework in order to generate verifiable mechanisms given the objective and constraints of the system. Our logical approach uses an iterative algorithm, which is computationally expensive and therefore requires some restrictions on the space search for the choice of parameters in order to make it feasible. However it can provide us with the entire game mechanism without assumptions on the game rules. On the other hand the numerical optimisation approach [15, 23, 24] can improve the social welfare by solving a fairly large optimisation problem but it assumes the rules of the game.

Future work includes studying the convergence of our algorithm, possibly identifying a class of mechanisms for which the convergence is guaranteed, extending this approach to Bayesian games, and finally applying it to real-life scenarios involving automated mechanism design such as e-commerce systems.

ACKNOWLEDGEMENTS

The author is grateful to Dr Frank Guerin for helpful discussions on game theory and logics. He thanks the reviewers for helpful comments in an early version of this paper. He also acknowledges funding from the UK EPSRC under grant EP/D02949X/1.

REFERENCES

[1] P. Battigalli and G. Bonanno, 'Synchronic information, knowledge and common knowledge in extensive games', *Research in Economics*, **53**, 77–99, (1999).
[2] Ken Binmore, *Fun and Games, A text on Game Theory*, D.C. Heath and Company, 1992.
[3] Vincent Conitzer and Tuomas Sandholm, 'Automated mechanism design for a self-interested designer', in *EC'03*, pp. 232–233. ACM Press, (2003).
[4] Vincent Conitzer and Tuomas Sandholm, 'Incremental mechanism design', in *IJCAI*, pp. 1251–1256, (2007).
[5] Hamido Fujita, Mohamed Mejri, and Béchir Ktari, 'A process algebra to formalize the lyee methodology', *Knowl.-Based Syst*, **17**(5-6), 263–281, (2004).
[6] Peter G. Harrison and B. Strulo, 'Spades - a process algebra for discrete event simulation', *J. Log. Comput.*, **10**(1), 3–42, (2000).
[7] Gerard J. Holzmann, *The SPIN Model checker: Primer and Reference Manual*, Addison, Boston, USA, 2004.
[8] D. Kozen, 'Results on the propositional mu-calculus', *Theoretical Computer Science*, **27**, 333–354, (1983).
[9] R. Milner, *Communication and Concurrency*, Prentice-Hall, 1989.
[10] John Nash, 'Noncooperative games', *Annals of Mathematics*, **54**, 289–295, (1951).
[11] Noam Nisan and Amir Ronen, 'Computationally feasible vcg mechanisms', *JAIR*, **29**, 19–47, (2007).
[12] D. C. Parkes, 'Auction design with costly preference elicitation', *Annals of Mathematics and AI*, **44**, 269–302, (2004).
[13] Corina S. Pasareanu, Matthew B. Dwyer, and Willem Visser, 'Finding feasible abstract counter-examples', *Soft. Tools for Tech. Transfer*, **5**(1), 34–48, (2003).
[14] Marc Pauly, 'Programming and verifying subgame-perfect mechanisms', *J. Log. Comput.*, **15**(3), 295–316, (2005).
[15] Steve Phelps, Peter McBurney, Simon Parsons, and Elizabeth Sklar, 'Applying genetic programming to economic mechanism design: evolving a pricing rule for a continuous double auction', in *AAMAS*, pp. 1096–1097, (2003).
[16] Brian J. Ross, 'A process algebra for stochastic music composition', in *International Computer Music Conference, ICMC 1995*, pp. 448–451, (1995).
[17] Tuomas Sandholm, Vincent Conitzer, and Craig Boutilier, 'Automated design of multistage mechanisms', in *IJCAI*, pp. 1500–1506, (2007).
[18] Emmanuel M. Tadjouddine and Frank Guerin, 'Verifying dominant strategy equilibria in auctions.', in *CEEMAS'07*, volume 4696, pp. 288–297, Leipzig, Germany, (Sep. 2007). LNAI, Springer.
[19] Emmanuel M. Tadjouddine, Frank Guerin, and Wamberto Vasconcelos, 'Abstracting and model-checking strategy-proofness for auction mechanisms', in *DALT*, (2008). accepted.
[20] Emmanuel M. Tadjouddine, Frank Guerin, and Wamberto Vasconcelos, 'Abstractions for model checking game-theoretic properties in auctions', in *AAMAS*, (2008). accepted.
[21] Alfred Tarski, 'A lattice-theoretic fixed point theorem and its applications', *Pacific J Math*, **5**, 285–309, (1955).
[22] Johan van Benthem, 'Extensive games as process models.', *Journal of Logic, Language and Information*, **11**(3), 289–313, (2002).
[23] Yevgeniy Vorobeychik, Christopher Kiekintveld, and Michael P. Wellman, 'Empirical mechanism design: methods, with application to a supply-chain scenario', in *EC'06*, pp. 306–315, New York, NY, USA, (2006). ACM Press.
[24] Yevgeniy Vorobeychik, Daniel M. Reeves, and Michael P. Wellman, 'Automated mechanism design in infinite games of incomplete information: Framework and applications', (2007).
[25] Carsten Weise, 'An incremental formal semantics for Promela', in *Proceedings of the 3rd International SPIN Workshop*, (1997).
[26] Michael Wooldridge, Thomas Ågotnes, Paul E. Dunne, and Wiebe van der Hoek, 'Logic for automated mechanism design - a progress report', in *AAAI*, pp. 9–17. AAAI Press, (2007).

Using Recency and Relevance to Assess Trust and Reputation

Sarah N. Lim Choi Keung and **Nathan Griffiths** [1]

Abstract.

In multi-agent systems, agents must typically interact with others to achieve their goals. Since agents are assumed to be self-interested, it is important to choose reliable interaction partners to maximise the likelihood of success. Sub-standard and failed interactions can result from a poor selection. Effective partner selection requires information about how agents behave in varying situations, and such information can be obtained from others in the form of recommendations as well as through direct experience. In open and dynamic environments, agents face quick and unforeseen changes to the behaviour of others and the population itself. This paper presents a trust and reputation model which allows agents to adapt quickly to changes in their environment. Our approach combines components from several existing models to determine trust using direct experiences and recommendations from others. We build upon previous models by considering the multi-dimensionality of trust, recency of information, and dynamic selection of recommendation providers. Specifically, we take a multi-dimensional approach for evaluating both direct interactions and recommendations. Recommendation sharing includes information about the recency and nature of interactions, which allows an evaluator to assess relevance, and to select recommenders themselves based on trust.

1 Introduction

Trust and reputation have been widely used to attempt to solve some of the issues linked with the uncertainty of interaction. Trust is used to assess the level of risk associated with cooperating with other agents; it is an estimate of how likely another agent is to fulfil its commitments [4, 6, 11]. Trust can be derived from direct interactions between agents and from reputation. Reputation is built from information received from third parties about an agent's behaviour. Based on the reputation information received, agents can make informed decisions about whether or not to interact with others [3].

In a dynamic environment, agents can change behaviour quickly and this must be identified by the agents relying on them, especially if they become less trustworthy in particular aspects of their service. For trust and reputation to be effective in guiding decisions, they must be sensitive to dynamic environments. Therefore, agents should adapt quickly to changes in their environment by selecting appropriate interaction partners and recommenders. In this respect, multi-dimensional trust and reputation allows the original information to be maintained for each service characteristic, such as timeliness and cost, instead of a single aggregated value. Moreover, the sharing of interaction summaries among agents maintains the richness of opinions on a per-characteristic basis and reduces subjectivity. When agents only share calculated trust values, they can be subjectively interpreted in different ways since the evaluator has calculated trust based on its own priorities. Several existing approaches already make use of these aspects, none addresses all of these issues. In this paper we present a model that integrates and extends components from existing approaches to include richer information in decision making and information sharing. The main contributions of our model are: (i) to use the recency of interactions when selecting interaction partners and witnesses, since information can become outdated in dynamic domains, (ii) to ensure that recommendations are accurate and relevant and that they contribute appropriately to the evaluation of reputation, and (iii) to use a richer format of information sharing to reduce subjectivity (including the recency of interactions and the level of witness experience).

[1] University of Warwick, UK, email: {slck, nathan}@dcs.warwick.ac.uk

2 Related Work

Many trust and reputation models have been developed to support agents in soliciting interaction partners. In this section we introduce some of the relevant related work. Marsh's formalism of trust is the basis for many computation approaches, including ours. ReGreT and FIRE are two of the most widely known approaches, while MDT-R and Ntropi introduce features that we build upon in our approach.

2.1 Marsh's Formalism

Marsh's formalism for direct interactions among agents [11], divides trust into *basic trust*, *general trust* and *situational trust*. Basic trust represents an agent's own trusting disposition, derived from its past experiences. An agent's general trust in another depicts how reliable the other is considered, irrespective of the situation. Situational trust is that placed in another agent in a specific situation.

Our model uses these three views of trust when we consider direct trust from direct agent interactions. An agent has an *initial trust* in another agent when it first starts interacting and has had no previous interactions. This is analogous to Marsh's basic trust. Situational trust is used to express an evaluator's trust in a target about a particular task. If the evaluator has interacted with the target but not for the specific task, then general trust is used. General trust is the average trust value calculated from interactions in different situations with the target. Marsh's approach does not take into account reputation and only models trustworthiness from direct experience. This limits the information available for trust evaluation, especially in cases where there are insufficient or no direct interactions. Our model complements direct trust with witness reputation to achieve greater accuracy when predicting agent behaviour. Additionally, we extend Marsh's view by including multi-dimensionality and agent confidence based on the MDT-R model [6] (described below).

2.2 ReGreT

ReGreT is a modular trust and reputation model that combines three dimensions of information to assess reputation: *individual*, *social* and *ontological* dimensions [12, 13]. The individual dimension relates to direct trust resulting from the outcomes of direct interactions between the evaluator and the target. The social dimension complements this by incorporating information on the experiences of other members of the evaluator's group with the target. There are three aspects to the social dimension: the evaluator's experience with its own group, the experience of members of its group with the target, and the view of the evaluator's group regarding the group that the target belongs to. To determine the social dimension of reputation, an evaluator may use three information sources: *witness reputation* calculated using information gathered from other agents; *neighbourhood reputation* based on the social relations between agents; and *system reputation* which is based on knowledge of the target agent's role. Finally, the ontological dimension considers how the various aspects associated with reputation can be combined. For example, the ontological dimension can define how the reputation of being a good seller relates to a reputation for providing a quality product, a reputation for timeliness, and a reputation for appropriate charging.

ReGreT relies heavily on knowledge of the social structure of the system, in terms of the groups to which agents belong, and the roles that they play. It also relies on knowing the ontological structure of reputation in the domain to define how different aspects of reputation relate to each other. The ReGreT model itself does not consider how agents can build knowledge of the social structure of their environment, but assumes that such information is available for a given domain. In open and dynamic domains such information may not be easily available, and may quickly become outdated as agents leave and join. Additionally, the ontological structure of reputation may not be easily available, and furthermore it may change over time as an agent's preferences change about what is important in an interaction. Although the social structure and reputation ontologies are not necessarily fixed in ReGreT, the sophistication of the model makes it hard to deal with any changes. Our approach uses reputation information provided by others in a similar manner to ReGreT, but without requiring knowledge of the social structure of the system or an ontology of reputation aspects, and so we use witness reputation but not neighbourhood or system reputation. In place of knowing the social structure we use the trust of witnesses and an estimation of the accuracy and relevance of their information, and instead of an ontology we use a weighted product model to combine reputation aspects.

2.3 FIRE

FIRE [9, 10] is a modular approach that integrates up to four types of trust and reputation from different information sources, according to availability. *Interaction trust* results from past direct interactions, and adopts the mechanism used in ReGreT's individual dimension of considering the outcomes of direct interactions between the evaluator and the target. *Role-based trust* uses social and role-based relationships between agents to assess trust, for example the power relationships between agents that might influence trust. *Witness reputation* is built from reports of witnesses about the target agent's behaviour. Finally, *certified reputation* is based on rating references from third-parties that are provided to the evaluator by the target agent itself. An extension to FIRE [8] handles possible inaccurate reports from recommending agents by introducing a credibility model.

The modular approach to trust and reputation in FIRE caters for a wide range of situations that can arise in multi-agent systems. In some situations not all components of FIRE can be used, because the required information may not be available. For example, in dynamic open systems it is likely that role-based trust will be of limited use, since roles are likely to be weakly defined and changeable. Similarly, the use of certified reputation is dependent on the existence of a suitable security mechanism, such as a public-key infrastructure [9]. In open and dynamic domains, as considered in this paper, the interaction trust and witness reputation components of FIRE are the most appropriate. As in ReGreT, FIRE enables an evaluator to rate its direct interactions with the target agent according to a number of terms, such as price and delivery date. Trust can then be calculated within these terms, for example an estimate of trust in terms of delivery date can be determined by extracting all available information about delivery dates from the history of interactions. Our approach extends this model, by providing a mechanism in which overall trust is defined as a combination of the various aspects of previous interactions, such that at run-time an agent can combine information about the various aspects according to their current relative importance. In FIRE, witness selection is done by maintaining a list of acquaintances according to their likelihood of providing the required information. FIRE does not consider how this is done, but assumes an application specific method exists [10]. In this paper, we build upon the interaction and witness reputation components of FIRE to use trust as an estimator for the provision of recommendations, removing the need for an application specific mechanism.

2.4 Ntropi

Abdul-Rahman and Hailes [1, 2] propose a trust and reputation model in which trust and the outcome of experiences are represented in levels. For instance, the labels for the trust level scale are '*Very Trustworthy*', '*Trustworthy*', '*Moderate*', '*Untrustworthy*', and '*Very Untrustworthy*' [1]. The model uses direct trust and reputation, as well as recommender trust to assess witness credibility, in computing a final trust degree for a target. Ntropi models two types of trust: situational trust and basic trust.

This model represents trust by classifying it into five levels, or *strata*. The disadvantage is that the trust values are coarse-grained, thereby losing both sensitivity and accuracy. Although comparisons are easier, the update of values is more complex than using continuous values [5]. In our approach, trust is stored as continuous values for increased accuracy, both for an evaluator's usage and for information sharing. We use direct trust and recommender trust in a similar way to Ntropi, however, we take a multi-dimensional view of trust and reputation that preserves much of the original meaning of the information gathered. In our model, the selection of witnesses is based on two factors: the accuracy and the relevance of recommendations. This is influenced by how Ntropi uses trust in the context of recommendation [1]. Our model incorporates these factors differently to Ntropi due to the difference in the representation of trust values.

2.5 MDT-R

MDT-R [6] is a mechanism of multi-dimensional trust and recommendations. Agents model the trustworthiness of others according to various criteria, such as cost, timeliness or success, depending on which criteria the agent considers important. Agents use their own direct experience of interacting with others, as well as recommendations. Distinguishing trust and recommendations for individual characteristics is valuable in identifying the service characteristics in

which the providing agents perform well, or less well. Trust information in multiple dimensions helps to maintain the original interaction data. Trust values are represented numerically in this approach due to the benefits of accuracy and the ease of comparison and update of values. However, MDT-R stratifies trust into levels (*à la* Ntropi) for ease of comparison. The sharing of information among agents often suffers from subjectivity, due to differences in interpretation. MDT-R deals with this by sharing summaries of relevant past interactions, instead of explicit values for trust.

3 Model Description

Our model is broadly based on MDT-R and adopts the multi-dimensionality of trust and recommendations, as well as the sharing of interaction summaries. We extend MDT-R by including information on recency and the experience of witnesses when sharing interaction summaries. This allows an evaluator to more accurately select witnesses, and thereby providers, as it further reduces the subjectivity of interpretation. Our model also considers the relevance of recommendations to better select recommenders and to assign them appropriate weights when calculating reputation.

3.1 Sources of Trust

As we have seen above, many different sources of information can be used to assess trust. Such sources must be available, relevant and accurate enough to be useful in selecting interaction partners. We view trust from direct interactions and recommendations from third parties as the two most important sources of information, since they are typically available with sufficient relevance and accuracy.

Direct interactions are an evaluator's main source of information about a target, and can be used to assess trust. This type of trust is called *direct trust*. The second information source is recommendations from third parties. We assume that witnesses give information about a target only if they have interacted with it. We do not currently consider indirect recommendations due to the added complexity of subjective opinions along a chain of witnesses. Trust from third party information is referred to as *witness reputation*. The term is adopted from FIRE [10] and refers to the same concept, but the way we build the reputation is different from FIRE, due to our use of multiple dimensions.

Our approach integrates these two types of information in different situations. Witness reputation is especially used when the evaluator has insufficient information from direct experience about a target to make an evaluation. Thus, in the event of insufficient information, the two information sources are combined to increase accuracy. In this paper, we do not consider collusion among agents, where a group of agents cooperate for their mutual benefit but impacting on others in the environment as a result. Any inaccuracies in recommendations arise due to differing circumstances, variations in behaviour of the target towards different witnesses, or malicious witness (giving false information). We will consider collusion in future work, as we aim to first ensure that the basic components of our model are efficiently improving agent interaction in a dynamic environment. We also assume that witnesses freely provide recommendations when requested.

3.2 Direct Trust

Trust information is captured in multiple dimensions, as in MDT-R [5, 6]. The separation into several dimensions enables information about specific service characteristics to be preserved. The subjectivity of trust, especially from recommendations, is an obstacle to making full use of the information obtained from witnesses. Sharing multi-dimensional trust information within interaction summaries [6], instead of calculated trust values decreases subjectivity. The dimensions correspond to the necessary characteristics that define a service. Any number of dimensions can be used, but for the purpose of illustration in this paper, we consider that an evaluator α models trust in target β along four dimensions [6]:

- success ($T^s_{\alpha\beta}$): the likelihood that β will successfully execute the task,
- timeliness ($T^t_{\alpha\beta}$): the likelihood that the task will be performed no later than expected by α,
- cost ($T^c_{\alpha\beta}$): the likelihood that the cost of performing the task will not be more than expected, and
- quality ($T^q_{\alpha\beta}$): the likelihood that the quality of the task performed by β will be met.

These trust values are derived from the past interactions of α and β. The evaluator stores information about each interaction in which β has performed a task on its behalf. Information about each interaction includes the service characteristics offered by β, as well as the actual values obtained on completion. The derived trust values refer to a specific task and so this is a type of *situational trust*. A successful interaction is one where β delivers results, irrespective of whether the other three characteristics were met. Meanwhile, a positive interaction with respect to the dimensions of timeliness, cost and quality refers to β performing as expected or better. Trust values are calculated when the evaluator needs to make a decision about whom to interact with. The range of the trust values in each dimension is $[-1, +1]$, where -1 means complete distrust and $+1$ means complete trust. The evaluator stores a history of past interactions with each provider for each task type. We denote the set of interactions in the history about provider β for the task type K as $HI_{\beta K}$. The size of the history corresponds to the number of interactions that the evaluator deems relevant. In future work, evaluators should be able to change the size of the history on a per-target basis to enable agents to store only the required information to assess trust.

The situational trust value $ST^d_{\alpha\beta K}$ is a function of the history of interactions with target β:

$$ST^d_{\alpha\beta K} = \frac{I^{d+}_{\alpha\beta K} - I^{d-}_{\alpha\beta K}}{I^{d+}_{\alpha\beta K} + I^{d-}_{\alpha\beta K}} \tag{1}$$

where $I^{d+}_{\alpha\beta K}$ is the number of positive interactions agent α has experienced with target β, of task type K in dimension d, and $I^{d-}_{\alpha\beta K}$ is the number of negative interactions.

The evaluator also stores the *general trust* of each provider it has interacted with and applies regardless of the service provided. General trust is used to assess the overall trustworthiness of an agent. It is useful when the evaluator does not have situational trust for a target for a specific task, as it gives an idea of how the target is likely to perform. The general trust $GT_{\alpha\beta}$ of evaluator α for target β is calculated as an average of the situational trust values in the success dimension:

$$GT_{\alpha\beta} = \frac{\sum_{k=1}^{allK} ST^s_{\alpha\beta K}}{allK} \tag{2}$$

where $allK$ is the size of the set of task types. We use only the success dimension to simplify calculation, since completing a task successfully has overriding priority when obtaining an agent's overall trustworthiness. If there are no previous interactions with β, then

general trust is equal to α's disposition, referred to as α's *initial trust*, denoted as $initialT_\alpha$.

MDT-R models confidence and trust decay as two important notions agents should consider when using past experience for trust assessment. In our model, confidence refers to the number of interactions between an evaluator and a target agent, and is calculated for each dimension, since not all dimensions are relevant in different interactions. C^d_β denotes the confidence level in the trust assessment of the target β for dimension d. Trust decay occurs when trust values become outdated due to lack of fresh interactions. The decay function (Equation 3) reduces the trust value according to how outdated the trust values are. A weight $\omega_{HI_{\beta K}}$ is assigned to an interaction according to recency; the more recent the interaction, the more weight it has, since more recent interactions give a more accurate reflection. The weight is based on the time since the interaction occurred and the frequency of interaction with β for the task type K. With fewer recent interactions, trust decays towards the initial trust value.

$$decay(ST^d_{\alpha\beta K}) = f(ST^d_{\alpha\beta K}, \quad initialT_\alpha, \quad \omega_{HI_{\beta K}}) \tag{3}$$

As proposed in MDT-R, trust values in our model are stratified at the time of comparison. When using numerical values, there is a risk of considering even insignificant differences in values to be important, and stratifying trust reduces this risk. Stratified trust is only used for comparisons and is not communicated to others. In our model, the number of strata used can be specified to allow for different levels of sensitivity. For example, if the number of strata is 10, trust values in the range $[0.8, 1]$ are taken to be the same. Thus, when two agents β and γ are being compared by situational trust in the success dimension, if $ST^s_{\alpha\beta K} = 0.85$ and $ST^s_{\alpha\gamma K} = 0.95$, both agents are taken to have similar trust values. A larger number of strata ensures a smoother transition between different strata, especially at the boundary between positive and negative trust [7].

3.3 Witness Reputation

Witness reputation is the trust of a target as communicated by third parties. The reputation of a target is sought when the evaluator has insufficient information to make a decision about whether to cooperate. A lack of information may occur for several reasons. For example, consider an evaluator α who wants to consider agent β for interaction, to perform a task K_1. In the first case, suppose α has never interacted with β and thus has no experience of β's behaviour. Alternatively, suppose α has previously interacted with β but for a different task K_2. Another case is when α has had too few interactions with β, or they are too outdated. In all these cases, α can ask the opinions of others who have interacted with β, in order to get a more accurate assessment of β's trustworthiness.

When an evaluator requires recommendations for an agent, it must decide which agents to ask. Such agents might have different kinds of experience with the target, and their opinions might not be useful to the evaluator. To decide who to ask, the evaluator uses *recommendation trust*, which estimates the accuracy and relevance of a witness' recommendation for the evaluator's purposes. Accuracy measures the similarity between the evaluator's own experience and the witness' opinion. Meanwhile, relevance relates to the usefulness of the recommendation, based on the recency of the interactions, the experience of the witness, and on how trustworthy the witness is in giving recommendations. We use recommendation trust rather than asking for recommendations about the reputation of a witness because the evaluator's past experience are more relevant for its decision making.

FIRE considers whether the witness has sufficient information about the target to give an opinion and its credibility [8]. Although this enables the evaluator to identify the accuracy of the recommendation by comparing it with its own experience, the relevance of a witness' trust information for the evaluator's purposes is not taken into account. In MDT-R, an agent selects witnesses by considering its most trusted interaction partners. However, it does not select witnesses based on the relevance of recommendations and there is no validation of whether the witness has given accurate information. The uncertainty lies in the possible difference in behaviour of the target towards different evaluators. Ntropi considers two factors when dealing with recommendations: (i) the closeness of the witness' recommendation and the evaluator's own judgement about the target, and (ii) the reliability of the witness in giving accurate opinions over time.

Our approach to reputation is influenced by Ntropi's consideration of accuracy and relevance when selecting witnesses. The relevance of recommendations is calculated by taking into account their recency, the experience of the witness, as well as the evaluator's recommendation trust and confidence in the witness. The accuracy of opinions is considered for interactions that have taken place following positive recommendations. The evaluator compares the outcome of the interaction with the recommendation previously obtained to assess how accurate it was. Recommendation trust is updated for each agent that has given recommendations. Initially, witnesses have a recommendation trust value equal to their general trust. This is later updated if the evaluator interacts with the recommended provider.

Witnesses provide recommendations to the evaluator in the form of interaction summaries for a specific task type where available. The summaries contain information such as the number of interactions the recommendation is based on, the recency of these interactions, and the proportion of positive and negative interactions in each trust dimension. The witness provides its general trust in the target if it does not have situational trust information. The use of interaction summaries is similar to that in MDT-R with the additional sharing of information about recency and experience, which can improve the evaluator's adaptation to changes in the behaviour of target agents. The evaluator combines the different recommendations by applying weights according to how relevant the witness' experience is, compared to the evaluator's. The weight $\omega_{WRR_{i\beta}}$ is the weight of the witness reputation relevance WRR of witness i in providing a recommendation for target β.

Thus, the witness reputation WR of target β's task type K in the dimension d is a function of the opinions received from witnesses and their respective weights:

$$WR^d_{\alpha\beta K} = \sum_{i=\gamma}^{\epsilon} \left(\frac{I^{d+}_{i\beta K} - I^{d-}_{i\beta K}}{I^{d+}_{i\beta K} + I^{d-}_{i\beta K}} \times \omega_{WRR_{i\beta}} \right) \tag{4}$$

where γ to ϵ are the set of selected witnesses for target β. $I^{d+}_{i\beta K}$ is the number of interactions between witness i and target β about service type K, for which β has met expectations for the dimension d. $I^{d-}_{i\beta K}$ is the number where expectations are not met. The weight of a recommendation is dependent on the witness' experience and its relevance. The relevance of witness i's recommendation about target β is calculated as:

$$\begin{aligned} WRR_{i\beta K} \quad = \quad & \left(\frac{t_{curr} - t_{median(HI_{\beta K})}}{t_{curr}} \right) \\ & + \frac{max_{WI}}{total_{WI}} + RT^i_\alpha + \omega_{C_{RT^i_\alpha}} \end{aligned} \tag{5}$$

where t_{curr} denotes the current time and $t_{median(HI_{\beta K})}$ is the recorded time of the median interaction as provided by the witness i for interaction with target β about task K. The inclusion of time in the calculation indicates the recency of the interactions on which the recommendation is based. The maximum number of interactions that the witnesses have used when giving recommendations is max_{WI}, and $total_{WI}$ is the total number of interactions actually used in that recommendation. The confidence of the evaluator α in its recommendation trust in the witness i is denoted as RT^i_α and the confidence weight $\omega_{C_{RT^i_\alpha}}$ shows the amount of influence this recommendation has compared to others.

The evaluator collects information about the witness from direct interactions and from previous recommendations the witness has provided. We do not assess the reliability of witnesses by collecting information from other agents because of the subjectivity of evaluating a witness' ability to provide recommendations to different agents.

3.4 Aggregation of Trust Sources

The evaluator α makes use of direct trust and witness reputation when assessing the trustworthiness of several potential providers for a task, and selects the best provider. The performance value of each provider is calculated as in MDT-R [6], with some changes to cater for the additional information when evaluating witness reputation.

The performance value for each potential provider is calculated as:

$$PV(\beta) = \prod_{i=1}^{n} (f_{\beta_i})^{\mu_i} \quad (6)$$

where there are n factors and f_{β_i} is the value for agent β in terms of the i'th factor and μ_i is the weighting given to the i'th factor in the selection of the agent's preferences. To assess trust using only direct trust, the values are stratified and the performance value is:

$$\begin{aligned} PV(\beta) \quad = \quad & (max_c + 1 - \beta_c)^{\mu_c} \times \beta_q^{\mu_q} \\ & \times stratify(ST^s_{\alpha\beta K})^{\mu_{ts}} \times stratify(ST^t_{\alpha\beta K})^{\mu_{tt}} \\ & \times stratify(ST^c_{\alpha\beta K})^{\mu_{tc}} \times stratify(ST^q_{\alpha\beta K})^{\mu_{tq}} \quad (7) \end{aligned}$$

where β_c and β_q are β's advertised cost and quality respectively, max_c is the maximum advertised cost of the agents being considered, μ_c and μ_q are the weightings given to the advertised cost and quality, and μ_{ts}, μ_{tt}, μ_{tc}, μ_{tq} are the weightings for the trust dimensions of success, timeliness, cost and quality respectively.

The calculation of the performance value, considering both direct trust and witness reputation is as follows:

$$\begin{aligned} PV(\beta) \quad = \quad & (max_c + 1 - \beta_c)^{\mu_c} \times (\beta_q)^{\mu_q} \\ & \times stratify(ST^s_{\alpha\beta K})^{\mu_{ts}} \times stratify(ST^c_{\alpha\beta K})^{\mu_{tc}} \\ & \times stratify(ST^t_{\alpha\beta K})^{\mu_{tt}} \times stratify(ST^q_{\alpha\beta K})^{\mu_{tq}} \\ & \times stratify(WR^s_{\alpha\beta K})^{\mu_{rs}} \times stratify(WR^c_{\alpha\beta K})^{\mu_{rc}} \\ & \times stratify(WR^t_{\alpha\beta K})^{\mu_{rt}} \times stratify(WR^q_{\alpha\beta K})^{\mu_{rq}} (8) \end{aligned}$$

where $WR^d_{\alpha\beta K}$ is the witness reputation for target β in the dimension d, and μ_{rs}, μ_{rc}, μ_{rt}, μ_r are the weightings for the witness reputation in the dimensions of success, timeliness, cost and quality respectively. (Note that the weights μ_i must sum to 1.)

4 Experimental Results

To validate our approach we have built a simulation environment, and have obtained a number of initial experimental results. Although more experimentation is required, our initial results are promising and demonstrate how trust and reputation can be used to facilitate more effective partner selection.

4.1 Effects of Size of Interaction History

We have investigated how our model behaves when agents change behaviour dynamically. Using a population of 50 agents we observe specific agent interactions. Half of the agents are malicious, and do not always complete the tasks. The remaining agents can be dishonest, and for instance, may charge more than advertised. We have simulated agent interactions over 1500 cycles, where one cycle allows every agent to have part of its tasks performed and to carry out tasks for others. We select one provider for a specific type of task and observe the evaluator's assessment of trust and performance of that provider.

The evaluator uses a history of interactions for each provider task type to predict that provider's likely future behaviour. We observe how the size of the history window affects the evaluator's decision making when others' behaviour changes. Tables 1 and 2 show the average number of cycles the evaluator takes to reach the updated behaviour of the target agent.

Table 1. Reliable to unreliable

Size	Average Duration
5	58.2
10	145.4
20	162.2
30	348.0

Table 2. Unreliable to reliable

Size	Average Duration
5	395.8
10	425.6
20	757.2
30	831.0

Tables 1 and 2 show that it takes longer for the evaluator to notice a change in provider behaviour with larger interaction window sizes. From these results, we expect that fewer failures will occur when the window size is smaller. In experiments where provider behaviour oscillates between good and bad , we also found that for smaller window sizes, the evaluator reacts faster to changes. Figure 1 shows the proportion of failed tasks for each window size.

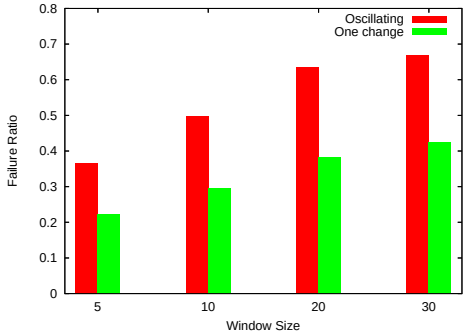


Figure 1. Failure ratio where provider behaviour oscillates between good and bad, compared to one change from good to bad

A malicious provider can however exploit an evaluator's small window to fail some interactions, while keeping the number of successful ones high enough for the evaluator to predict high reliability for that provider. We have set up an experiment where the malicious provider fails an interaction with the evaluator every 6 interactions. For window sizes 5, 10 and 20, the failure ratios are similar at around 0.16, while for the larger window size 30, we observe a slight decrease in failure of around 3%. Compared to Figure 1, smaller win-

dow sizes are not beneficial in recognising some behaviour changes. Hence, the evaluator needs to find the right balance between adaptation speed and guarding against such malicious behaviour.

4.2 Comparison of Evaluation Mechanisms

We have compared the effectiveness of using trust, and trust with reputation, against using single service characteristics in a number of settings. Again, we use a population of 50 agents, half of which are malicious. The simulation ran for 500 cycles with individual task execution taking several cycles, depending on execution speed and task duration. The set of agents offers the same task types over the simulation runs, but agent behaviour varies in terms of honesty.

The experiments are set up to observe the performance of evaluator $a1$. Agent $a1$ has a set of tasks to be performed and there are several alternative providers. We look at three evaluation mechanisms that $a1$ might use to assess providers: cost, trust and trust with reputation. We consider the number of tasks generated that have been successful, unsuccessful or incomplete. These are presented as a ratio of the total number of $a1$'s tasks. If the evaluator adds a new task type later in the simulation, it will have no previous interactions for this task and so will ask for recommendations.

Figures 2 and 3 show representative results for the distribution of task performance, where new task types are introduced during the simulation. The ratio of success (Success), execution failure (Failed-U), declined tasks (Failed-D) and any remaining tasks (Remaining) is shown. The evaluation mechanisms are denoted as C, T and TR for cost, trust and trust with reputation respectively. The results are affected firstly by the nature of the population, with more honest populations giving higher success rates, as expect. In the case of Figure 2 the evaluator was situated in a more cooperative environment. The results also show that using trust or trust and reputation improve the success rate compared to using the service characteristics (in this case, cost). In cooperative environments there is a small improvement, while in less honest populations the improvement is more significant (Figure 3). Our results also show that depending on the environment, trust or trust and reputation may give the best result. We are conducting ongoing experiments to identify the conditions that determine which method is best.

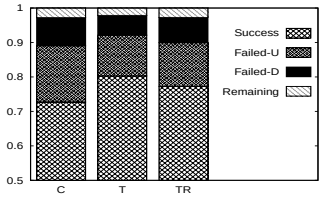


Figure 2. Population set 1

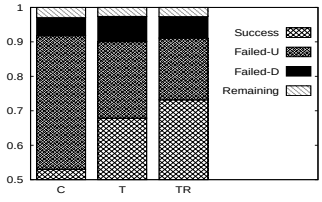


Figure 3. Population set 2

5 Conclusions and Future Work

From our experiments we observe that using trust and trust with reputation to select providers gives better results in most cases, than using service characteristics only. In some situations, the use of trust together with reputation is an improvement over the use of trust only. However, further experimentation is required to determine the circumstances in which the improvement is significant. The ability to recognise different situations can help an agent to better decide which evaluation mechanism to use for the maximum benefits. We have also considered how our model performs when agents change behaviour. Our aim is to enable an evaluator to quickly identify behaviour changes and adapt its strategy to maintain a high success rate. A smaller interaction window size enables the evaluator to reassess trust quickly. However, in certain cases, malicious agents can exploit this by periodically failing. The development of our model, and our initial results, highlight many questions that must be answered for effective use of trust and reputation. One important question is how to balance the potentially conflicting features that an evaluator needs, such as the compromise between the speed of adaptivity to behaviour changes and guarding against malicious behaviour. Future work will consider how agents can achieve this balance, and will investigate the circumstances under which trust or trust and reputation should be used.

REFERENCES

[1] A. Abdul-Rahman. *A Framework for Decentralised Trust Reasoning*. PhD thesis, Department of Computer Science, University College London, UK, 2005.

[2] A. Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. In *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS 2000)*, page 6007. IEEE Computer Society, 2000.

[3] V. Buskens. Social networks and the effect of reputation on cooperation. ISCORE Paper No. 42, Utrecht University, 1998.

[4] D. Gambetta, editor. *Trust: Making and Breaking of Cooperative Relations*. Department of Sociology, University of Oxford, 2000.

[5] N. Griffiths. Task delegation using experience-based multi-dimensional trust. In *Proceedings of the 4th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 489–496, New York, NY, USA, 2005. ACM Press.

[6] N. Griffiths. Enhancing peer-to-peer collaboration using trust. *International Journal of Expert systems with Applications*, 31(4):849–858, 2006.

[7] N. Griffiths. A fuzzy approach to reasoning with trust, distrust and insufficient trust. In *Cooperative Information Agents X*, volume 4149 of *Lecture Notes in Computer Science*, pages 360–374. Springer-Verlag, 2006.

[8] T. D. Huyhn. *Trust and Reputation in Open Multi-Agent Systems*. PhD thesis, Electronics and Computer Science, University of Southampton, June 2006.

[9] T. D. Huynh, N. R. Jennings, and N. Shadbolt. Developing an integrated trust and reputation model for open multi-agent systems. In *Proceedings of the 7th International Workshop on Trust in Agent Societies*, pages 65–74, New York, USA, 2004.

[10] T. D. Huynh, N. R. Jennings, and N. Shadbolt. An integrated trust and reputation model for open multi-agent systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 13(2):119–154, 2006.

[11] S. Marsh. *Formalising trust as a computational concept*. PhD thesis, Department of Computer Science, University of Stirling, 1994.

[12] J. Sabater and C. Sierra. REGRET: reputation in gregarious societies. In *Proceedings of the Fifth International Conference on Autonomous Agents*, Montreal, Canada, 2002. ACM Press.

[13] J. Sabater and C. Sierra. Social ReGreT, a reputation model based on social relations. *ACM SIGecom Exchanges*, 3(1):44–56, 2002.

Modelling and Administration of Contract-Based Systems

Simon Miles and **Nir Oren** and **Michael Luck** and **Sanjay Modgil** and **Nora Faci** [1]
Camden Holt and **Gary Vickers**[2]

Abstract. Mirroring the paper versions exchanged between businesses today, electronic contracts offer the possibility of dynamic, automatic creation and enforcement of restrictions and compulsions on agent behaviour that are designed to ensure business objectives are met. However, where there are many contracts within a particular application, it can be difficult to determine whether the system can reliably fulfill them all; computer-parsable electronic contracts may allow such verification to be automated. In this paper, we describe a conceptual framework and architecture specification in which normative business contracts can be electronically represented, verified, established, renewed, etc. In particular, we aim to allow systems containing multiple contracts to be checked for conflicts and violations of business objectives. We illustrate the framework and architecture with an aerospace example.

1 Introduction

It has often been argued that agents interacting in a common system, society or environment need to be suitably constrained in order to avoid and solve conflicts, make agreements, reduce complexity, and in general to achieve a desirable social order (e.g. [4, 5]). For many, this role is fulfilled by norms, which represent what *ought* to be done by a set of agents. Views of norms differ, and include fixed laws that must never be violated as well as more flexible social guides that merely seek to bias behaviour in different ways. Yet the obligations, prohibitions and permissions that may affect agent behaviour in a normative system can also be *documented* and communicated between agents in the form of *contracts*. Electronic contracts, mirroring the paper versions exchanged between businesses today, offer the possibility of dynamic, automatic creation and enforcement of such restrictions and compulsions on agent behaviour. However, where there are many contracts within a particular application, it can be difficult to determine whether the system can reliably fulfill them all; computer-parsable electronic contracts may allow such verification to be automated.

There are two pre-requisites to realistically applying an electronic contracting approach in real-world domains. First, to exploit electronic contracts, a well-defined conceptual framework for contract-based systems, to which the application entities can be mapped, is needed. Second, to support the management of contracts through all stages of the contract life-cycle, we need to specify the functionality required of a contract management architecture that would underly any such system, leading to ready-made implementations for particular deployments of that architecture.

[1] King's College London, UK, email: simon.miles@kcl.ac.uk
[2] Lost Wax, UK

The CONTRACT project[3] aims to do just this. Funded by the European Commission as part of its 6th Framework Program, the project seeks to develop frameworks, components and tools that "make it possible to model, build, verify and monitor distributed electronic business systems on the basis of dynamically generated, cross-organisational contracts which underpin formal descriptions of the expected behaviours of individual services and the system as a whole." In this context, this paper documents the CONTRACT project's work on both of the pre-requisites identified above. The aims and vision of the project are described in full elsewhere [18]. More specifically, the technical contributions described in this paper are: the specification of a model for describing contract-based systems; the specification of an architecture for managing such systems; and the mapping of an aerospace application to those models.

Our approach takes a distinct approach in several respects. First, its development is explicitly driven by a range of use cases provided by a diverse set of small and large businesses. One consequence of this diversity is that our approach must account for different practices and possibilities in each stage of the lifecycle of a contract-based system. It is therefore defined in terms of abstract *process types*, to be instantiated in different ways for different circumstances. We provide a non-exhaustive set of options for instantiating these process types, and technologies to support these processes. A more specific requirement addressed by our approach is in managing not just fulfilment or violation of contractual obligations, but also other states of the system with regard to those obligations, such as being in danger of violation, being expected to easily fulfil an obligation, and application-specific states.

In the next section, we provide an overview of the overall structure, introducing the conceptual framework and applying it to a running example. Then, in Section 3, we discuss how the contractual obligations imply *critical states* of the system that we may wish to detect and react to in order to effectively manage the system. In Section 4, we describe the architecture: the process types that are required to manage contract-based systems and components that can support such processes. We disuss related work in Section 5 and conclude with future work in Section 6.

2 CONTRACT Framework and Architecture

The models and procedures comprising the CONTRACT *framework* and *architecture* are shown in Figure 1. The primary component of this is the framework itself, depicted at the top of the figure, which is the conceptual structure used to describe a contract-based system, including the contracts and the agents to which they apply.

[3] `www.ist-contract.org`

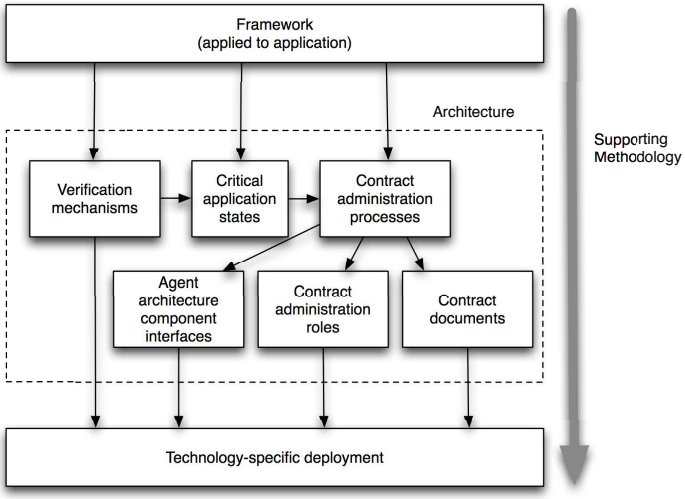


Figure 1. The overall structure of the CONTRACT architecture and framework

From the framework specification of a given application, other important information is derived. First, understanding the contractual obligations of agents allows us to specify the *critical states* that an application may reach. A critical state of a contract-based system with regard to an obligation essentially indicates whether the obligation is fulfilled or fulfillable, e.g. achieved, failed, in danger, etc. This is discussed in detail in Section 3. A state-based description, along with the deontic and epistemic implications of the specified contracts, can then be used to verify a system either off-line or at run-time [12] (though we do not discuss this further here).

The framework specification is used to determine suitable processes for adminstration of the electronic contracts through their lifetimes, including establishment, updating, termination, renewal, and so on. Such processes may also include observation of the system, so that contractual obligations can be enforced or otherwise effectively managed, and these processes depend on the critical states identified above. Once suitable administration processes are identified, we can also specify the roles that agents play within them, the components that should be part of agents to allow them to manage their contracts, and the contract documents themselves. Such process types and roles are described in Section 4.

2.1 A Contract-Based System Framework

We first specify a conceptual framework by which contract-based systems can be described. This framework provides a clear indication of how particular applications can exploit contracts and how they must be supported in managing them. By being abstract and generic, such a framework may also be used to translate contract data from one concrete format to another.

Contracts document *obligations*, *permissions* and *prohibitions* (collectively *clauses*) on agents and are *agreed* by those agents. Simply, obligations are statements that agents should do something and prohibitions are statement that they should not. Since agents are *autonomous*, external control can only be specified through normative structures such as contracts. Thus, permissions are defined as exceptions to prohibitions: if something was not prohibited, it is not meaningful for a permission to be granted.

The agents obliged, permitted and prohibited in a contract are *parties* to that contract, which specifies *contract roles*, played by agents within it. Each clause in a contract applies to roles, to which agents are *assigned*, and each agent can hold multiple contracts with the same or different parties. Finally, a *contract proposal* is one that has not yet been agreed. These concepts are summarised in Table 1.

Concept	Definition
Role	A named part that can be played by an agent in a system.
Obligation	A statement that an agent playing a given role should do something.
Prohibition	A statement that an agent playing a given role should not do something.
Permission	An exception to a prohibition for an agent playing a given role under given circumstances.
Clause	An obligation, prohibition or permission.
Assignment	A statement that an agent should play a given role.
Proposal	A document containing a set of clauses and assignments, where every role referred to which each clause applies has been assigned to an agent.
Contract	A proposal to which all assigned agents have agreed.

Table 1. The primary concepts in the CONTRACT framework

2.2 Aerospace Use Case

To test and illustrate the efficacy of our approach, we adopt an engineering application, based on the aerospace aftercare market, targeted by Lost Wax's agent-based Aerogility platform [1], and used as a running example through the paper.

The application concerns the continued maintenance of aircraft engines over their lifetime. In this domain, an engine manufacturer is contractually obliged to ensure operators' aircraft have working engines. For an engine to be working, it should not be overdue for regular servicing or left waiting to be fixed after a fault is discovered. An aircraft's engine can be replaced when it lands at some location if there is a suitable spare engine present at that location. As well as replacing engines to ensure continued operation of the aircraft, an engine manufacturer will service the engines it has removed, so that the serviced engine can be added back into circulation (the "engine pool") and used to replace other engines.

In addition to long-term contracts between engine manufacturers and operators, we consider short-term contracts regarding particular instances of servicing engines. These sit in the context of long-term contracts but, by being specified explicitly, allow the parties to use and commit to resources more flexibly. In a long-term contract between an aircraft operator and an engine manufacturer, the manufacturer agrees to service the operator's aircraft to some overall specified standard over the duration of the contract. Such a contract is provided in Table 2, using the framework concepts. Here, the operator specifies a preferred time within which the manufacturer must service an aircraft, and the manufacturer is obliged to meet this in 90% of cases. If the manufacturer does not meet short-term contract requirements (see below), penalties are deducted from the long-term payment the operator is obliged to make. The operator is obliged to provide adequate engine data so that the manufacturer can fulfil their servicing obligations. Finally, the operator may have demands on the provenance of an engine: operator A may be happy to re-use engines previously used by operators B or C but not those used by D.

In this context, a short-term contract concerns the servicing of a particular aircraft at a particular time (see Table 3). It is again between two parties: the aircraft operator and the engine manufacturer. In this case, the manufacturer has more specific obligations: that they must either service an aircraft in the preferred timescale or pay a penalty, and that they must service it within a maximum period. The limitations on provenance apply in the particular short-term servicing as they do in the long-term aftercare.

Roles	Manufacturer, Operator
Obligations	**O1** Manufacturer agrees to servicing contracts requested by operator during aftercare contract period.
	O2 Manufacturer services engines within the preferred time specified by the servicing contracts in at least 90% of cases.
	O3 Operator pays for servicing of engines, minus any penalties.
	O4 Operator must supply engine health data to the manufacturer in an adequate time to allow problems requiring unscheduled maintenance to be detected.
Prohibitions	**P1** Manufacturer is prohibited from supplying engine parts previously used by other operators not on an approved list (if one is given) or on a disapproved list (if one is given).
Permissions	**R1** Manufacturer is allowed to supply engine parts previously used by other operators on an approved list (if one is given).

Table 2. Long-term aftercare contract

Roles	Manufacturer, Operator
Obligations	**O5** Manufacturer services aircraft in preferred time, or pays penalty (taken out of aftercare contract payment from operator).
	O6 Manufacturer services engine in maximum time.
Prohibitions	**P2** Manufacturer is prohibited from supplying engine parts previously used by other operators not on an approved list (if one is given) or on a disapproved list (if one is given).
Permissions	**R2** Manufacturer is allowed to supply engine parts previously used by other operators on an approved list (if one is given).
	R3 Operator is allowed to take a penalty from the manufacturer if an aircraft is left on the ground for longer than the preferred time agreed.

Table 3. Short-term servicing contract

Such formal documents of agreements are important, especially when there are multiple agreements and when these agreements can interact, because they can reveal critical points of potential or actual conflict. If it is possible to examine such contracts, and determine where these critical points lie, so that they may be monitored for violations, or even modified to avoid the potential for violation. In what follows, these aims inform the elaboration of our architecture. For example, a short-term conflict between two servicing contracts in the aerospace domain occurs when a manufacturer is obliged to service two operators' aircraft at the same time, but can only service one due to a lack of resources. Long-term conflicts are also present, as in a conflict between a servicing contract and an aftercare contract arising when a manufacturer must choose between servicing one operator's aircraft within the maximum time limit and servicing another operator's aircraft within the preferred time, where the manufacturer is in danger of not having serviced the latter operator's aircraft within the preferred time limit for 90% of cases.

3 Critical States of Contract-Based Systems

By identifying the *critical states* of the system with respect to given contractual obligations, indicating whether the obligation is fulfilled or fulfillable, it is then easier to determine which of these needs to be checked for and acted upon to ensure that the system performs well. A state-based description can also be used as a basis for verifying whether the system will always result in a desirable state.

3.1 Obligation States

Each obligation implies a set of states for the system with regard to that obligation. In part, these can be specified independently of the application. For example, we classify obligations into three types:

- An obligation to *achieve* some state G, e.g. to pay some amount to another agent.
- An obligation to *maintain* some state H, e.g. to keep an aircraft in working order.
- An obligation to *behave* in some way, where that behaviour is to fulfil obligation $O(X)$ whenever event $E(X)$ occurs, e.g. when aircraft X requires servicing, to service X in an acceptable time.

For an achievement obligation, there are three significant states: *Pre-achievement*, *Succeeded* and *Failed*. Each of these has particular properties with regard to the goal state G, as shown in Table 4 (top). In *Pre-achievement*, the goal state is achievable but not yet achieved; in *Succeeded*, the system is in the goal state; and in *Failed*, the goal state is no longer achievable. Similarly, a maintenance obligation implies three significant states, as shown in Table 5 (top). In the *Maintained* state, the system is in the goal state; in *Succeeded*, the system can no longer leave the goal state; in *Failed*, the system has left the goal state. Finally, the significant states of a behaviour obligation depend on the obligation, O, triggered in reaction to each event, but it has some states of its own as shown in the top of Table 6. In the *Pre-trigger* state, the triggering event has not yet occurred; in the *Reaction Active* state, an event has occurred but the obligation it has triggered into taking force has not yet reached a *Succeeded* or *Failed* state; in *Reaction Failed*, that reaction obligation has reached a Failed state, and so the behaviour obligation as a whole has failed; in *Reaction Succeeded* state, the particular reaction obligation has succeeded; and in *Succeeded*, no more applicable events can ever occur and so the behaviour obligation as a whole has succeeded. All obligations also imply a state, *Cancelled*, when the obligation no longer holds.

State	Properties
Pre-achievement	Not G G achievable Agent obliged to achieve G
Succeeded	G
Failed	G unachievable Agent obliged to achieve G
Cancelled	No agent obliged to achieve G
Sub-State	**Additional Properties**
Initial	
Danger	G in danger of becoming unachievable
Likely Success	G' achieved, where G' is a significant subset of G

Table 4. Basic states (top) and sample pre-achievement sub-states (bottom) of an achievement obligation

State	Properties
Maintained	H Not H achievable Agent obliged to maintain H
Succeeded	Not H unachievable
Failed	Not H Agent obliged to maintain H
Cancelled	No agent obliged to maintain H
Sub-State	**Additional Properties**
Initial	
Danger	Not H in danger of becoming true

Table 5. Basic states (top) and sample maintained sub-states (bottom) of a maintenance obligation

3.2 Significant Sub-States

In addition to the application-independent system states above, applications often refer to significant sub-states part-way between an

obligation coming into force and its success or failure. Examples of these are shown in the bottom portion of Tables 4, 5 and 6. For example, an application may need to detect whether an obligation is in danger of violation and so allocate more resources to ensure that it is fulfilled instead, implying a *Danger* critical state of the system with regard to that obligation as shown in Table 4 (bottom). On the other hand, if an obligation is being fulfilled unexpectedly easily, an application may take advantage of this by transferring resources being used in support of this obligation to other tasks, e.g. the *Likely Complete* critical state shown in Table 6.

State	**Properties**
Pre-trigger	No new E(X) has occurred Agent obliged to achieve G(X), maintain G(X) or behave in way B(X) on every E(X)
Reaction Active	E(a) occurred As Pre-achievement, Maintenance or Pre-trigger state for G(a)/B(a) Agent obliged to achieve G(X), maintain G(X) or behave in way B(X) on every E(X)
Reaction Failed	E(a) occurred As respective Failure state for reaction G(a) or B(a) Agent obliged to achieve G(X), maintain G(X) or behave in way B(X) on every E(X)
Reaction Succeeded	E(a) occurred As respective Succeeded state for reaction G(a) or B(a) Agent obliged to achieve G(X), maintain G(X) or behave in way B(X) on every E(X)
Succeeded	E(X) can never occur again
Cancelled	No agent obliged to achieve G(X), maintain G(X) or behave in way B(X) on every E(X)
Sub-State	**Additional Properties**
Initial	
Imminent	E(X) is likely to occur imminently
Likely Complete	E(X) is unlikely to occur again

Table 6. Basic states (top) and sample pre-trigger sub-states (bottom) of a behaviour obligation

3.3 Example

As an example, in Table 7, we enumerate critical states for an achievement obligation, indexed **O2**, in the long-term aftercare contract of Table 2. It is an achievement obligation as it describes an eventual state of the system in which a state has been achieved, i.e. 90% of servicing cases were performed in the preferred time period. When the contract first comes into force, i.e. the system time is within the contract period, the state Pre-achievement: Initial holds. In this state, insufficient cases have been performed to determine whether success is likely. After 5% of cases are performed, the system will be in either Pre-achievement: Satisfactory or Pre-achievement: Danger states, and may vary between them over the contract period. Pre-achievement: Satisfactory holds where over 5% of cases are performed within the preferred time, while Pre-achievement: Danger holds where between 5% and 10% of cases exceeded that time. The value of taking account of these two states is that transfer of resources between fulfilment of different obligations can be triggered by changes of state. Eventually, the system will reach either Succeeded state, where the contract period is exceeded and over 90% of cases were performed on time, or Failure state, where over 10% have exceeded the preferred time. The choice of the appropriate sub-states (Pre-achievement: Satisfactory and Pre-achievement: Danger in this case) is entirely application dependent: considering more states allows finer control as appropriate, but may also add overheads.

Pre-achievement: Initial	Less than (estimated) 5% of servicing cases performed and within contract period
Pre-achievement: Satisfactory	Over 5% of cases performed, less than 5% exceeded preferred time and within contract period
Pre-achievement: Danger	Between 5% and 10% of cases exceeded preferred time and within contract period
Succeeded	Less than 10% of cases exceeded preferred time and beyond contract period
Failed	More than 10% of cases exceeded preferred time

Table 7. States of aftercare contract obligation **O2**

4 Architecture of Contract-Based Systems

Aside from modelling contract-based systems using the CONTRACT framework, we also address the issue of administration: how to manage the processes involved in creating, maintaining, acting on and otherwise processing contracts and contract proposals. We identify four key process types in the contract life-cycle.

Establishment brings about the existence of the contract.
Maintenance and Update ensures a contract's integrity over time.
Fulfilment brings about the fulfilment of obligations while observing its prohibitions.
Termination or Renewal end the normative force of the contract, or renew it to apply for a longer period.

Each of these process types can be instantiated in different ways, depending on the application and its deployment. The choice dictates the roles agents must play to fulfil the administration duties implied. Below, we examine each process type in turn.

4.1 Establishment

There are many potential ways to establish a contract, varying in complexity. To give an illustration, we present two below.

Full Proposal Establishment Process Here, one party, the *proposer*, creates a full proposal, excluding some assignments of roles to agents, and signs it. It then uses a *registry* to discover agents that may fulfil the unassigned contract roles. For each unassigned role in turn, it offers the proposal to an agent, a *potential party* it is satisfied can assume that role. If the party is willing, it signs the proposal and returns it. When the last role is filled, a contract is established

Template Discovery Establishment Process Alternatively, a process may be used in which an agent discovers a contract *template* that may be instantiated in a way that fulfils its goals. This implies the use of a *template repository*, where templates can be stored. Such templates may have some assigned roles; that is, they may describe services for which a provider is willing to negotiate terms.

4.2 Maintenance and Update

The continued existence and integrity of a contract after establishment is important in reliable systems. As with establishment, there are multiple ways in which this can be achieved, and the functionality that needs to be provided depends on the particular contract and application.

Contract Store Maintenance Process Here, contract parties use a *contract store* to maintain and control access to contracts. The store is obliged only to allow agents to change the contract when all parties send a signed agreement of the change to be made.

All Party Signature Maintenance Process In this process, integrity is preserved by the contract being signed by all parties in a way that prevents editing without detection; for example, digital signatures based on reliable certificates. The signed document includes the contract itself, and an indication of whether it is a revision of a previous version. Each party should check the signatures of the contract before accepting it as binding.

4.3 Fulfilment

For every contractual obligation and prohibition, there are certain processes that can be performed to help ensure they are fulfilled. As with the processes above, these imply particular administrative roles that must be played by agents. The administrative roles carry with them obligations, prohibitions and permissions, which may be documented in the same contract as the one that is the target of administration, or another contract. The processes below often refer to particular system states with regard to obligations: these are the states specified in Section 3.

Observation of Fulfilment Process An *observer* observes state changes to determine whether contractual obligations are being fulfilled. It can notify other agents when an obligation is being violated or in danger of violation[4]. An observer X is in an obligation pattern of the following form:

> X is obliged to observe for critical state S of contract clause C, and notify registered listeners when it occurs.

Management of Fulfilment Process A *manager* is an agent that acts when an obligation is not being fulfilled, is in danger of not being fulfilled or a prohibition is breached. It knows about the problem by (conceptually at least) registering to listen to the notifications from an observer. Manager is a role, and one agent may play the role of both manager and observer. The nature of the action taken by a manager may vary considerably. In highly automated and strict applications, an automatic penalty may be applied to a party. In other cases, a management agent may be a human who decides how to resolve the problem. Alternatively, a manager may merely provide analysis of problems over the long term, so that a report can be presented detailing which obligations were violated. A manager X is in an obligation pattern of the following form:

> X is obliged, whenever the system reaches a critical state S of contract clause C, to perform action A.

Note that we avoid the sometimes used term, *enforcer*, because enforcement implies action either in the case of failure or in the case of likely failure, and often sanctions or financial penalties. In the applications we are considering, likely success is also a good state to act upon, e.g. to reallocate resources used for the successful obligation, and actions may take softer forms, e.g. notified humans renegotiate the contract, so the more generic term, manager, is used.

An example of an observer's obligation in the aerospace application is shown in Table 8 (top), and of a manager's obligation in Table 8 (bottom). The observer, Checker, is obliged to check that a Danger state has not been reached for the number of suitable engines available at a given location, and the manager, Enforcer, listens to observations on this state and rectifies the situation when it occurs.

[4] Note that we do not use another term sometimes appearing in the literature, *monitor*, because it is also often used to refer to quantitative infrastructure-level measurements of system metrics: in the case of many business applications, observation is of discrete high-level states.

Roles	Checker, Manufacturer, Operator
Obligations	Checker monitors the number of engines available to the manufacturer at a given location that are suitable for a given operator, and notifies registered agents if it falls below a minimum quantity.

Roles	Enforcer, Checker
Obligations	Enforcer, on hearing from checker that the number of suitable engines at a location has fallen below a minimum level, transports a suitable engine from another location.

Table 8. Engine supply checking contract (top) and Engine supply enforcement contract (bottom)

4.4 Termination and Renewal

Termination of a contract means that the obligations and other clauses contained within it no longer have any force. A contract may be terminated in several ways: (i) it may terminate *naturally* if the system reaches a state in which none of its clauses apply, e.g. a contract's period of life expires or all obligations have been met; (ii) it may terminate *by design* if the contract has an explicit statement that the contract is terminated when a particular event occurs (e.g. if one party fails to meet an obligation, the contract is terminated and all others are released from their obligations); or (iii) it may terminate *by agreement*, if parties agree that the contract should no longer hold, and update it accordingly (in line with the process chosen for the Maintenance and Update type above). Renewal of a contract means that a contract that would have imminently terminated naturally is updated so that termination is no longer imminent (again depending on the Maintenance and Update process type above).

4.5 Administrative Roles and Components

The processes above all require the fulfilment of particular administrative roles, e.g. contract store, registry, observer, manager. For some of these components, we can provide generic implementations. For example, a contract store, based solely on contract documents and having nothing to do with the application itself, is easy to implement generically. Others, such as managers, need to have application-specific instantiations, as dealing with a contractual obligation not being fulfilled varies greatly between applications. Specifying the components further is largely a technology-dependent issue out of scope of this paper.

5 Related Work

There has been much previous work on various aspects of contract-based system modelling, enactment and administration, and our approach is intended to build on and be compatible with other ideas presented elsewhere. For example, there are many approaches to negotiation which may be used in the establishment of contracts [13], and the administration of contracts can integrate with other useful behaviour, such as observation of fulfilment and violation of obligations potentially feeding into a longer-term assessment of agents [7]. Work on multi-party contracts [19] adopt modelling techniques specifically designed to enable detection of parties responsible for contract violation, but do not use normative concepts to regulate agent behaviour, or model other contract administration processes.

In addition, the wider domains of normative systems and agreement in service-oriented architectures informs our work. Concepts such as norms specifying patterns of behaviour for agents, contract clauses as concrete representations of dynamic norms, management or enforcement of norms itself being a norm, are all already established in the literature [6, 7, 15, 8].

However, the approach in this paper is distinct in that it is concerned with the development of practical system deployments for business scenarios. In particular, business systems operate in the context of wider organisational and inter-organisational processes, so that *commitments*, providing assurance over the actions of others assumes great importance. While potentially less flexible over the short term, explicit contracts provide just such commitments and are therefore more appropriate for business systems than more flexible, less predictable *ad hoc* approaches [9, 17].

We also believe our system to be more widely applicable than some other approaches. By classifying processes into types with different instantiations, the architecture can be incorporated into a wider range of application domains and deployments than fixed protocols would allow. In addition, we describe how administrative functions, such as storing or updating a contract, can be achieved. This contrasts with specifications such as WS-Agreement and Web Services Service Level Agreement, where the specifications cover only part of the necessary administration [2, 16]. Abstract architectures for electronic contracting, and associated case studies, have been described elsewhere; most notably in the work of Grefen et.al [10, 3]. However, accommodation of deontic specifications in order to regulate agent behaviour is not modelled in this work. Our approach aims for broad observation and management of obligations and prohibitions, so as to verify whether they are being achieved, prevent failure when in danger of violation and take advantage of success when obligations are being easily met. Some existing work does consider system states with regard to contract clauses [14], but none, to our knowledge, classifies obligations and the critical states they imply as we have done in this paper, a necessary pre-requisite to observing and managing obligation fulfilment in accordance with a particular application.

Others have raised the issue that observers and managers have, themselves, to be observed and managed [11]. Here, by modelling observers and managers as agents, we allow for the same contract framework to apply to them. However, this clearly has its limits and at some point *trust* between agents must be explicitly modelled in the system, a topic to be addressed in future work.

6 Conclusions and Future Work

In this paper, we have presented the CONTRACT conceptual framework and architecture, and shown how they apply to aircraft aftercare. By creating a technology-dependent implementation along these lines, an application can take advantage of the reliable coordination provided by electronic contracts. The CONTRACT project aims to allow multi-agent systems to be verified on the basis of their contracts, building on work by Lomuscio et al. on deontic interpreted systems [12]. While this verification is beyond the scope of this paper, it places a requirement on our framework that the properties of the target system are identified and isolatable, and a requirement on the architecture that such information can be captured in order to pass to a verification mechanism. Perhaps equally importantly, we also aim for an open source implementation built on Web Services technologies, requiring the architecture to be compatible with such an objective. Finally, taking a very practical standpoint, we intend to construct a methodology to guide development of applications that use electronic contracts through the process from conceptual framework to deployment. To ensure wide applicability, this will be applied to CONTRACT's other test applications in insurance settlement, software provisioning and certification testing.

Acknowledgement: The CONTRACT project is co-funded by the European Commission under the 6th Framework Programme for RTD with project number FP6-034418. Notwithstanding this fact, this paper and its content reflects only the authors' views. The European Commission is not responsible for its contents, nor liable for the possible effects of any use of the information contained therein.

REFERENCES

[1] Aerogility. http://www.aerogility.com/, 2007.
[2] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, Jim Pruyne, J. Rofrano an S. Tuecke, and M. Xu, 'Web services agreement specification (ws-agreement)', Technical report, Global Grid Forum, (2004).
[3] S. Angelov and P. Grefen, 'A case study on electronic contracting in on-line advertising - status and prospects', in *; Network-Centric Collaboration and Supporting Frameworks - Proceedings 7th IFIP Working Conference on Virtual Enterprises*, pp. 419–428, (2006).
[4] R. Conte and C. Castelfranchi, 'Norms as mental objects. From normative beliefs to normative goals', in *MAAMAW93*, pp. 186–196, (1993).
[5] R. Conte, R. Falcone, and G. Sartor, 'Agents and norms: How to fill the gap?', *Artificial Intelligence and Law*, **7**, 1–15, (1999).
[6] C. Dellarocas, 'Contractual agent societies: Negotiated shared context and social control in open multi-agent systems', in *Workshop on Norms and Institutions in Multi-Agent Systems, 4th International Conference on Multi-Agent Systems*, Barcelona, Spain, (June 2000).
[7] F. Duran, V. Torres da Silva, and C. J. P. de Lucena, 'Using testimonies to enforce the behaviour of agents', in *AAMAS'07 Workshop on Coordination, Organization, Institutions and Norms in agent systems*, eds., Jaime Sichman and Sascha Ossowski, pp. 25–36, (2007).
[8] Andrés García-Camino, 'Ignoring, forcing and expecting concurrent events in electronic institutions', in *Coordination, Organization, Institutions and Norms in agent systems (COIN@AAMAS'07). Co-held in Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, Honolulu, Hawai'i, USA, (May 2007).
[9] M. Ghijsen, W. Jansweijer, and R. Wielinga, 'Towards a framework for agent coordination and reorganization, agentcore', in *AAMAS'07 Workshop on Coordination, Organization, Institutions and Norms in agent systems*, eds., J. Sichman and S. Ossowski, pp. 13–24, (2007).
[10] P. Grefen and S. Angelov, 'On τ, μ, π and ϵ-contracting', in *Proceedings CAiSE Workshop on Web Services, e-Business, and the Semantic Web*, pp. 68–77, (2002).
[11] Andrew J. I. Jones and Marek J. Sergot, *Deontic Logic in Computer Science: Normative System Specification*, chapter On the Characterisation of Law and Computer Systems: The Normative Systems Perspective, 275–307, John Wiley & Sons, 1993.
[12] A. Lomuscio and M. Sergot, 'Deontic interpreted systems', *Studia Logica*, **75**, (2003).
[13] H. Lopes Cardoso and E. Oliveira, 'Using and evaluating adaptive agents for electronic commerce negotiation', in *Proceedings of the International Joint Conference, 7th Ibero-American Conference on AI: Advances in Artificial Intelligence*, volume 1952 of *Lecture Notes In Computer Science*, pp. 96–105, (2000).
[14] H. Lopes Cardoso and E. Oliveira, 'A contract model for electronic institutions', in *AAMAS'07 Workshop on Coordination, Organization, Institutions and Norms in agent systems*, eds., J. Sichman and S. Ossowski, pp. 73–84, (2007).
[15] F. Lopez y Lopez, M. Luck, and M. d'Inverno, 'A normative framework for agent-based systems', *Computational and Mathematical Organization Theory*, **12**(2–3), 227–250, (2005).
[16] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck, 'Web service level agreement (wsla), language specification', Technical report, IBM Corporation, (January 2003).
[17] E. Muntaner-Perich, J. Lluis de la Rosa, and R. Esteva, 'Towards a formalisation of dynamic electronic institutions', in *AAMAS'07 Workshop on Coordination, Organization, Institutions and Norms in agent system*, eds., J. Sichman and S. Ossowski, pp. 61–72, (2007).
[18] S. Willmott, M. Dehn, M. Luck, M. Pechoucek, A. Lomuscio, J. Vazquez, and J. Dale, 'Contract based approaches to robust, verifyable cross-organisational web services applications', Technical report, Universitat Politcnica de Catalunya, (2007).
[19] Lai Xu, 'A multi-party contract model', *ACM SIGecom Exchanges*, **5**(1), 13–23, (2004).

Cooperation through Tags and Context Awareness

Nathan Griffiths [1]

Abstract. In recent years a range of techniques such as trust, reputation and social norms have been used to support cooperation. Attention has tended to focus on situations where a degree of reciprocity, either direct or indirect, exists between agents, and existing techniques typically rely on such reciprocity to engender cooperative behaviour. Increasingly, environments are emerging where large numbers of agents interact without ongoing repeat interactions, in which there is little or no reciprocity. In this paper, we propose a mechanism to support cooperation without requiring reciprocity. Our approach supplements tag-based cooperation with an assessment of neighbourhood context to cope with cheaters. Using a simple peer-to-peer scenario we show how cooperative behaviour is favoured, and the effect of cheating agents is reduced.

1 Introduction

A range of techniques including trust, reputation and social norms have been used to establish and maintain cooperation in multi-agent systems. Many successful approaches have been developed for a number of environments. However, the increased use of large distributed systems such as peer-to-peer (P2P) networks, and the emergence of ubiquitous computing environments, mean that enabling and maintaining cooperation remains an important question. Such environments typically contain a large number of agents that must cooperate, but the environment characteristics are such that repeat interactions between agents may be rare. Many of the common approaches for supporting cooperation in multi-agent systems, although helpful, are not a complete solution, since little may be known about potential interaction partners and there is a relatively low likelihood of any subsequent interactions with the same partner. In this paper we propose a mechanism, that combines ideas from biology and the social sciences, to support cooperation in such environments. Our approach is an extension of the tag-based mechanism proposed by Riolo, Cohen and Axelrod (RCA) [14]. The approach we propose in this paper is related to that we describe in [6], in which an alternative extension to RCA's approach is considered.

Most existing approaches to cooperation are based on reciprocity, namely the notion that repeated encounters imply that any altruistic or selfish act performed by an agent may eventually be returned by the recipient. Direct reciprocity is the simplest, and most common, approach where two agents have repeat interactions in which there is the opportunity to "cooperate" or "defect". The iterated "prisoner's dilemma" is a quintessential example of such a setting. In large scale systems, such as P2P networks, interactions between a given pair of agents are infrequent and often single-shot, and so there is minimal direct reciprocity present. An alternative is indirect reciprocity, where a third party is involved in repeat interactions. Agents are unlikely to have direct repeat interactions, but are likely to interact with others whose behaviour with third parties they have previously observed. Nowak and Sigmund characterise direct reciprocity through the principle of "You scratch my back, and I'll scratch yours". Similarly, indirect reciprocity is characterised as "You scratch my back, and I'll scratch someone else's" or "I scratch your back and someone else will scratch mine" [11]. In some circumstances, however, even indirect reciprocity might be limited, and we may need to enable cooperation without reliance on reciprocity of any form, for example if there is no memory of past encounters [14].

Trust and reputation are the most common approaches to supporting cooperation in multi-agent systems [9, 12, 13]. However, such techniques are based on the notion of reciprocity and so are of limited use in situations where reciprocity is lacking. In this paper we extend RCA's tag-based mechanism [14], to provide a model for establishing and maintaining cooperation that does not assume reciprocity, and is suitable for situations where repeat interactions are rare. In the following section we introduce the theoretical approaches upon which our model is based, along with the promising initial results obtained by others. The model itself is introduced in Section 3. Our experimental setting, in the form of a simple P2P system, and selected experimental results are discussed in Section 4, and Section 5 concludes the paper.

2 Related Work

Indirect reciprocity is not an novel idea: biologists and social scientists have long considered cooperation in environments where the individuals concerned may not directly meet again, but where cooperative strategies are favoured [1, 3, 10]. Furthermore, theoretical models of cooperation exist that do not require any reciprocity, but instead are based on the recognition of cultural artifacts, such as the "green beard effect" and "kin" recognition [2, 4]. Promising results have recently been obtained using "tags" [8] as cultural artifacts to enable cooperation without reciprocity [14], which in has in turn led to a technique to improve cooperation in P2P networks [7]. Existing work on tags, however, has given only limited consideration to the existence of "cheaters" in the population, and it is this issue that we address in this paper.

Riolo, Cohen and Axelrod describe a tag-based approach to cooperation in which an agent's decision to cooperate is based on whether an arbitrary "tag" associated with it is sufficiently similar to that associated with the potential recipient [14]. RCA illustrate their approach using a simple "donation scenario" in which each agent is chosen to act as a potential donor with a number of neighbours. If the agent donates it incurs a cost c and the recipient receives a benefit b, otherwise both agents receive nothing (it is assumed that $b > c$). RCA use parameter values of $b = 1$ and $c = 0.1$. (These values are in turn adopted from Nowak and Sigmund, and the addition of a cost of 0.1 is to avoid negative payoffs [10].)

[1] University of Warwick, UK, email: nathan@dcs.warwick.ac.uk

In RCA's model each agent i is initially randomly assigned a tag τ_i and a tolerance level T_i with a uniform distribution from $[0, 1]$. An agent A will donate to a potential recipient B if B's tag is within A's tolerance threshold T_A, namely $|\tau_A - \tau_B| \leq T_A$. Thus, agents with a high tolerance will donate to others with a wide range of tags, while those with a low tolerance only donate to others with very similar tags [14]. RCA have performed simulations in which each agent acts as a potential donor in P interaction parings, after which the population of agents is reproduced in proportion to their relative scores. Each offspring's tag and tolerance is subject to a potential mutation, such that with some small probability a new (randomly selected) tag is received or the tolerance is mutated by the addition of Gaussian noise (with mean 0 and a small standard deviation). RCA found that a high cooperation rate can be achieved with this simple model, in which no reciprocity is required. Their results show oscillations in which a cooperative population is established, only to be invaded by a mutant whose tag is similar (and so receives donations) but with low tolerance (and so does not donate). Such mutants initially do well and take over the population, lowering the overall rate of cooperation. Eventually, the mutant tag becomes the most common and cooperation again becomes the norm [14].

RCA's approach is an effective mechanism for achieving cooperation without relying on reciprocity, but their model relies on an assumption that no cheaters are present in the population. A cheating agent is one that accepts donations, but will not donate to others, even if the "rules" of the system dictate that it should. Thus, a cheater in RCA's scenario would accept donations, but never donate to others regardless of tag similarity. We assume that cheaters follow the usual rules of reproduction in terms of offspring characteristics (e.g. tag and tolerance), but that their offspring will also be cheaters.

Hales and Edmonds (HE) apply RCA's approach in the context of a P2P network, with two important changes [7]. The first change is to adopt RCA's "learning interpretation" of the reproduction phase, such that each agent compares itself to another and adopts the other's tag and tolerance if the other's score is higher (again subject to potential mutations) [14]. The second change is that HE interpret a tag as an agent's set of neighbours in the P2P network. Thus, adopting another agent's tag is equivalent to re-wiring the P2P network such the other agent's connections are adopted [7]. Again, there is a small probability of mutation, which is interpreted as replacing a randomly selected neighbour with another node in the network. Simulations performed by HE have shown this approach to be very promising in situations where agents are able to re-wire the network, and in which there are no cheaters. In this paper, motivated by HE's promising results, we focus on achieving cooperation in the presence of cheaters, without permitting agents to re-wire their network neighbourhoods. Our approach is based on RCA's model, and HE's application of it (minus re-wiring), supplemented by a mechanism to cope with cheaters.

3 Extending Tags through Context Assessment

In this paper we use a P2P network as an illustrative scenario, and although we intend our approach to be fairly generic, our discussion will focus on a P2P setting. We consider a network of nodes, or agents, in which each agent has a fixed number n of connections to neighbours. The network topology is assumed to be fixed, and we do not permit agents to re-wire their network connections. Furthermore, unlike RCA and HE we assume that a proportion of the population will be cheaters, meaning that they will take all the benefits offered to them but will always refuse to act cooperatively towards others. For simplicity, we adopt the "donation scenario" used by RCA, along with their parameter values of $b = 1$ and $c = 0.1$ for the recipient benefit and donor cost respectively. It should be noted that although this is an artificial scenario, it could be extended in the manner of HE to more realistic P2P applications such as file sharing [7]. In HE's approach a node's tag is interpreted as being its specific set of neighbours, and tolerance measures the similarity between these sets. Other interpretations are possible, since a tag is simply a discernible attribute or trait. Thus, in a P2P setting a tag might also correspond to service characteristics as well as network properties (as in HE's approach). For example, tags could be interpreted as the set of services offered by a node, or the set of users of a particular node.

Our approach is founded upon RCA's tag-based technique, but we incorporate a simple mechanism to combat cheaters in which agents assess their current context, in terms of their neighbours' donation behaviour, as part of the decision to donate. Each agent i is initially assigned an arbitrary tag τ_i and tolerance T_i with uniform distribution from $[0, 1]$[2]. As in RCA's model, an agent A will donate to a potential recipient B if B's tag is within a certain threshold of its own. To demonstrate the impact of cheaters, initially suppose that this threshold corresponds to A's tolerance (as per RCA's model), meaning that if $|\tau_A - \tau_B| \leq T_A$ then A will donate to B. Later, we will expand this definition to include A's assessment of its current context. RCA's learning interpretation of reproduction is adopted (i.e. that used by HE) such that after a fixed number P of interaction pairings an agent compares itself to another selected at random. If the other agent is more successful than itself then the other's details (its tag and tolerance) are copied, meaning that the other agent reproduces, otherwise no change is made. If the parent agent is a cheater, then its offspring will also be a cheater, regardless of its other characteristics. After reproduction there is a potential mutation of the offspring's tag and tolerance, with probabilities m_τ and m_T respectively. In the reproduction stage it is the tag and tolerance values that are copied, and not the network neighbourhood. Each offspring has the same set of neighbours as the node that initiated the reproduction (i.e. the node that compared itself with another). Since we are using RCA's learning interpretation of reproduction this means that the network topology remains static for all generations, and it is the tag and tolerance values that are "learnt".

In common with RCA we find that a relatively stable donation rate (i.e. cooperation) over a large number of generations is established for appropriate parameters, provided that cheaters are not introduced into the population. Figure 1 shows the dynamics of the donation rate for a configuration that mirrors RCA's setting. Specifically, we use the parameter values $m_\tau = m_T = 0.01$ and $P = 3$. Note that in our P2P setting an agent has a restricted set of neighbours (in this case $n = 49$ where the network size $N = 100$) whereas in RCA's approach an agent has all others in the population as "neighbours" in this sense. Our values differ from those used by RCA in that the probability of tag mutation and tolerance mutation are lower (RCA use $m_\tau = m_T = 0.1$). Using these parameters the form of our results matches those obtained by RCA in [14]. If we use RCA's parameter values for m_τ and m_T we get a significantly lower donation rate than in their simulations. The reasons for this are unclear, and require future investigation. However, Edmonds and Hales notice similar differences from RCA's results, and suggest that bias in reproducing agents with equal scores and automatic donation to "tag

[2] More strictly we allow tolerance to have a lower bound of -10^{-6} to address Roberts and Sherratt's concerns that RCA's approach forces agents with identical tags to always cooperate [15]. The results discussed in this paper permit this small negative tolerance.

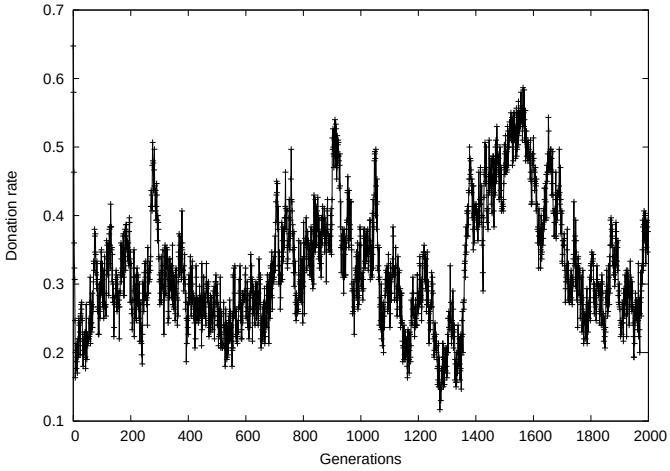


Figure 1. Donation rate with no cheaters using RCA's approach.

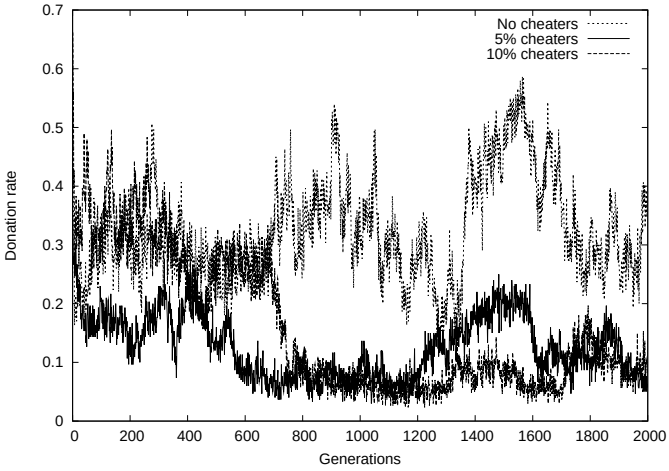


Figure 2. The effect of cheaters on donation rate with RCA's approach.

clones" in RCA's simulation are potential contributory causes [5].

Figure 1 shows that RCA's model (parameter values aside) allows cooperation to be established in the absence of cheaters. Unfortunately, when cheaters are introduced cooperation soon disappears. Figure 2 shows the effect of creating a population where a proportion of agents act as cheaters, who accept donations from others but never donate (regardless of tag similarity). Where there are no cheaters (the upper dotted line) cooperation is established as before. Introducing 5% of the population as cheaters reduces the donation rate (the solid line) and allowing 10% of agents to be cheaters (the dashed line) leads to minimal cooperation (with under 10% of interactions being cooperative). Without modification, therefore, RCA's approach soon fails to provide cooperation in the presence of cheaters, with even relatively small proportions of cheaters significantly reducing the average donate rate. Note that Figures 1 and 2 show a single simulation run to illustrate the evolution of the system. The results presented later in this paper are based on an average across multiple runs.

To cope with the presence of cheaters we extend RCA's approach such that the decision to donate is related to the context in which an agent is situated, in addition to its tolerance. Each agent has a fixed set of n connections to its neighbours, and we assume that these neighbours are able to observe the agent's donation behaviour. This observation assumption is realistic in many real-world settings. For example, in a file sharing system nodes can observe whether other nodes' downloads have completed, or in a communication network nodes can detect whether packets have been forwarded. Using the observations of its neighbours' donation behaviour, an agent is able to assess the context in which it is situated, with respect to how cooperative its neighbours are (i.e. how often they donate). Agents have a fixed length memory, in which they records the last l donation interactions observed for each of their neighbours. Where the neighbour donated to another agent a value of $+1$ is recorded, and where it refused to donate a value of 0 is recorded. The memory operates as a FIFO queue, such that new entries are appended until the maximum capacity of l is reached, at which point the oldest entry is removed from the head of the queue to allow the new entry to be appended. Based on the set of observations across all neighbours an agent can estimate the current context. Note that this memory is fairly sparse, since the number of interactions is relatively small compared to the number of agents, and so the overhead incurred is fairly small.

In order to assess its current context an agent considers each of its neighbours in turn, and taking them together builds an assessment of how cooperative its context is (in terms of donations). The contribution to the context c_n of neighbour n is simply the proportion of observed interactions in which the neighbour donated, given by:

$$c_n = \begin{cases} \frac{\sum_{j=1}^{l_n} o_n^j}{l_n} & \text{if } l_n > 0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where o_n^j represents the j'th observation of n (i.e. $+1$ for a donation and 0 for a refusal, as defined above), and l_n is the number of observations of n's donation behaviour ($l_n < l$). By considering the donation behaviour of each of its n neighbours, an agent can assess its current context C_A as follows:

$$C_A = \frac{\sum_{i=1}^{n} c_n}{n} \quad (2)$$

An agent can now consider its context when deciding whether or not to donate. Our assumption is that an agent is more likely to donate if in a cooperative context. This is related to the notion of indirect reciprocity in that agents "expect" that by donating they are likely to receive a donation from some other (observing) agent in the future. However, because the number of interactions is small compared to the number of agents, this is a *weak* notion of indirect reciprocity. Specifically, we do not assume that a donor will have directly observed a recipient's past behaviour, but only that donors are able to make a general assessment of their current context. The notion of context is incorporated into the model by adapting the decision to donate, such that both tolerance and context are considered. To ensure that an agent has sufficient observations on which to base its assessment we introduce a minimum observations threshold σ. If the total number of observations exceeds σ then context is incorporated into the donation decision, otherwise RCA's standard approach is used. Thus, if $\sum_{i=1}^{n} l_n \geq \sigma$ then context is incorporated into the donation decision, while if $\sum_{i=1}^{n} l_n < \sigma$ tolerance alone is used as per RCA's approach. Assuming that there are sufficient observations, then an agent A will donate to B if:

$$|\tau_A - \tau_B| \leq (1 - \gamma).T_A + \gamma.C_A \quad (3)$$

where T_A is A's tolerance and C_A its assessment of the current context. The parameter γ allows us to tune the model. A value of $\gamma = 0$

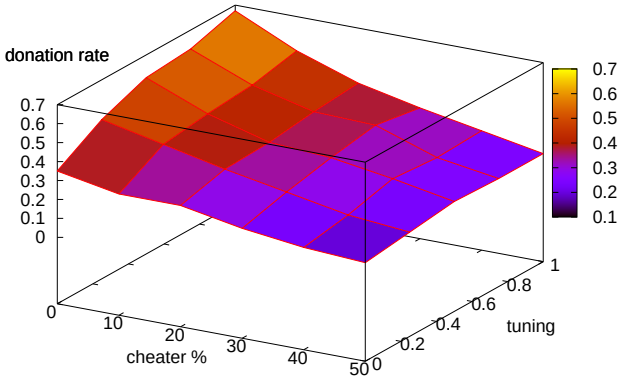


Figure 3. Donation rate using "standard" reproduction.

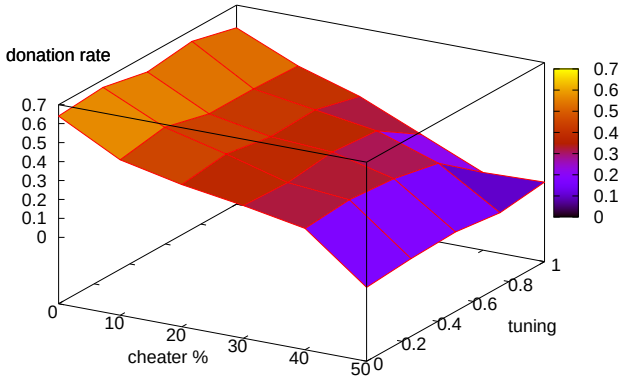


Figure 4. Donation rate using "context-based" reproduction.

means that the model is identical to RCA's approach, while a value of $\gamma = 1$ implies that the decision to donate is based solely on the agent's assessment of its context, with tolerance having no bearing. Values between 0 and 1 allow both tolerance and context to influence the donation decision.

Our approach differs from typical approaches to achieving cooperation through trust and reputation, since there is less reliance on the existence of specific observations. Trust and reputation mechanisms typically assume that, taken together, a group of agents will have sufficient information about an individual's past behaviour to estimate its reputation [9, 13]. Such information is not guaranteed in a P2P setting, and so we use a general assessment of an agent's context, rather than attempting to assess an individual's cooperative nature. Our experimental results show that using this approach we can still achieve a significant improvement in cooperation. (Although certainly, if sufficient information was available to use a more standard reputation mechanism, then it would be likely to perform better.)

The second area in which we consider an alternative to RCA's approach is with respect to reproduction. In RCA's model, after a certain number of interactions an agent will compare itself to another at random. If the other agent is more successful then its tag and tolerance values are copied (subject to minor mutations), i.e. the successful agent reproduces. In addition to replacing tolerance in the decision to donate by a combination of tolerance and context, we also consider using context for reproduction. If on comparison with another agent the other is more successful, its tag is copied, as is its assessment of its context. For the resulting offspring, the decision to donate becomes a consideration of a combination of its current context and its parent's context. Thus, offspring A will donate to B if:

$$|\tau_A - \tau_B| \leq (1 - \vartheta).C_{parent(A)} + \vartheta.C_A \qquad (4)$$

where $C_{parent(A)}$ refers to A's parent's assessment of its context (at time of reproduction), and ϑ is a tuning parameter that allows us to determine the influence of the current and parent's context assessments. If ϑ is 1 then only the current context assessment is considered, while a value of 0 means that only the parent's context is considered. Note that the parent's context is only considered by its immediate children, since any subsequent children only inherit the current assessment of context from their parents, which is independent from the grandparent's context assessment.

4 Results and Discussion

We have performed a number of simulations to investigate the effectiveness of our model, the influence of the tuning parameters (γ and ϑ), and the alternative reproduction mechanisms. Our simulations are built using the PeerSim P2P simulator[3]. We have experimented with various networks sizes, neighbourhood sizes and parameter values. Our simulations typically ran for 1500 generations, although longer simulations have been undertaken to check for long term stability. In this section we discuss the main findings based on our results. In the previous section, Figures 1 and 2 show how the donation rate evolves and oscillates over generations in a single simulation run. In this section, however, we average the donation rate over the whole simulation (1500 generations), and further take an average over 10 runs of the simulation. This allows us to compare donation rates for different configurations, without having to consider the inherent oscillations that the tag-based approach produces.

The initial tag and tolerance assigned to an agent in the simulation results presented here are randomly selected uniformly from $[0, 1]$. We have also followed RCA and explored high initial tolerance ($T = 0.5$) and low initial tolerance ($T = 0.005$) settings. Our results mirror those found by RCA in that other than for short transients the end results are not substantially different from using a random initial tolerance [14].

The main characteristics that determine the donation rate in our model are the tuning parameters γ and ϑ for the 'standard' reproduction and 'context-based' reproduction approaches respectively. Figures 3 and 4 show how the donation rate is affected by the proportion of cheaters for standard reproduction and context-based reproduction, for varying values of the tuning parameters. The results are based on a network size $N = 100$ with each agent having $n = 49$ neighbours, a minimum observation threshold of $\sigma = 3$, and a history window size of $l = 5$. The standard deviation of the donation rate in all settings shown in these results is fairly low (below 0.05), showing that the donation rate achieved is fairly consistent.

As expected, higher proportions of cheaters significantly reduce the donation rate achieved using both reproduction approaches. Figure 3 allows us to compare the effectiveness of using context in the donation decision in comparison to RCA's approach. Where the tun-

[3] `http://peersim.sourceforge.net/`

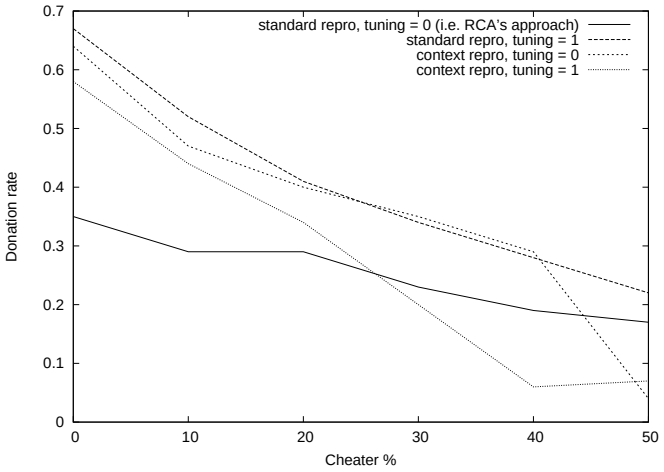


Figure 5. Donation rate for the tuning parameter extremes of both reproduction methods.

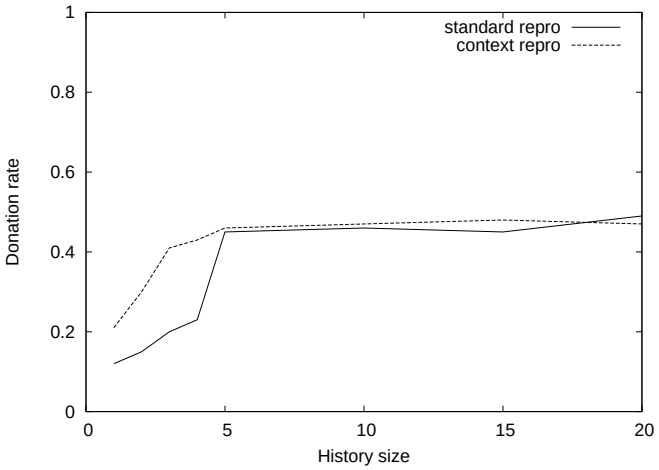


Figure 6. Donation rate for varying history window sizes.

ing parameter γ is set to 0 our model is identical to RCA's, and as the proportion of cheaters rises from 0% to 50% the donation rate drops from around 0.35 to 0.15. It is clear from Figure 3 that increasing the influence of context in the donation decision, by increasing the tuning parameter γ, improves the donation rate. This improvement is most significant at low cheater proportions, but remains even at high cheater rates. Comparing Figure 4 and Figure 3 we note that using context-based reproduction also gives an improvement over RCA's unmodified approach. Again, the improvement is most pronounced at low to medium cheater proportions (below 40%). However, for high proportions of cheaters (50%) using context-based reproduction actually gives a worse performance than RCA's mechanism regardless of the tuning parameter ϑ. This effect can be better observed if we consider the donation rate for standard reproduction versus context-based reproduction for the extremes of the tuning parameters, as shown in Figure 5.

It is clear from Figure 5 that we can improve on the donation rate achieved by RCA's unmodified approach (standard reproduction with $\gamma = 0$), shown by the solid line, for all settings of cheater proportion. The best results are obtained using our modification for considering context in the donation decision rather than tolerance (tuning parameter $\gamma = 1$), but using RCA's standard reproduction method, shown by the upper dashed line. Context-based reproduction focusing on the parent's context ($\vartheta = 0$), the short-dashed line, instead of standard reproduction gives a slight reduction in performance for less than 20% cheaters, a very small increase for 20–40%, and a very signifcant decrease (much worse than RCA) for above 40% cheaters. Context-based reproduction where the current context is considered rather than the parent's context, shown as the dotted line in Figure 5, performs better than RCA, but worse than our other configurations, for cheater rates of around 0–25%. For rates above 25% it performs worse than all the other approaches.

From the results presented so far we can conclude that our modification to include context in the donation decision does give a significant improvement over RCA's approach. Using context-based reproduction focusing on the parent's context gives little advantage in low–medium cheater proportions (for $\vartheta = 0$) and performs worse than standard reproduction for high cheater proportions (or where the current context is emphasised with $\vartheta = 1$).

The effect of different history window sizes (l) on the donation rate is given in Figure 6. Note that for these results the number of pairings between reproduction cycles P was increased accordingly. It can be seen that above a window size of 5 the effect of window size on donation rate is minimal for both reproduction methods (both tuning parameters are set to 1). There is a very small increase in donation rate when larger histories are considered, but the improvement is negligible. Furthermore, the memory overhead of maintaining longer observation histories for each neighbour is likely to outweigh the small improvement in donations for most settings. For window sizes below 5 the donation rate is reduced, significantly so below a history window of 3 interactions. Where a small window size is used the context-based reproduction method gives slightly improved performance.

We also consider the effect of an agent's neighbourhood size on the donation rate. Figure 7 shows the donation rate for both reproduction methods (with tuning parameters $\gamma = \vartheta = 1$) in a population of 10% cheaters. Neighbourhood size is shown as the percentage of the total nodes in the network N that are in an agent's neighbourhood, i.e. $n/N \times 100$. In this case the network size was restricted to 100 for efficiency of simulation, but we have obtained selected corresponding results for larger networks of up to 2500 nodes (the current practical limit of our simulator's capabilities). It is clear that regardless of network size, using standard reproduction with context in the donation decision again outperforms the use of context in both the donation decision and reproduction. Higher donation rates are generally achieved for larger neighbourhood sizes. For the standard reproduction approach donation rate improves with neighbourhood size up to 60% (i.e. an agent has 60% of the network as neighbours), after which there is a slight decline in performance. It should be noted, however, that large neighbourhoods (e.g. above 40%) are likely to be impractical in most real-world systems, due to the large numbers of agents involved, and so the results for below 40% are the most relevant to real-world applications. For the context-based reproduction approach we also see a significant increase in donation rate as the neighbourhood is initially expanded. This increase is again reduced for medium to large neighbourhoods, resulting in a slight decline for very large sizes. Figure 7 also shows the standard deviation of the runs used to obtain the donation rates. For the standard reproduction approach the standard deviation is fairly low and consistent (around 0.05). However, using context-based reproduction gives an inconsistent standard deviation (in the range of 0.05–0.2), illustrating the instability of this approach in comparison to standard reproduction.

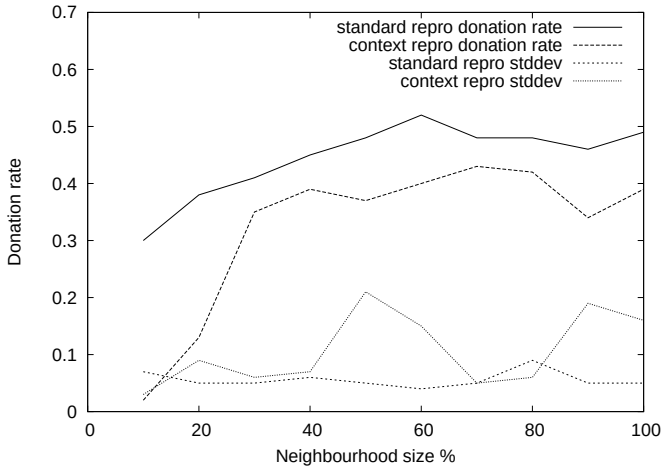


Figure 7. Donation rate for varying neighbourhood sizes.

5 Conclusions

In this paper we have described a mechanism for establishing cooperation amongst agents without a reliance on reciprocity. Building on RCA's tag-based approach we have shown how incorporating an assessment of an agent's current context into the donation decision improves the donation rate. Context assessment is dependent on the extent of the interaction history recorded and on the neighbourhood size. Our results show that the history window size has minimal impact on donation rate, while increasing neighbourhood size does increase donation rate (at least for practical neighbourhood sizes below approximately 40%). We also considered an alternative to RCA's reproduction method, in which a parent's context was inherited by its offspring and subsequently used in donation decisions. Our results demonstrate that this alternative reproduction method was not generally effective. Overall, our simulations show that augmenting RCA's approach with context assessment for the donation decision is successful, and gives a significant increase in cooperation (of over 30% in some settings), but that RCA's standard method for reproduction is the most effective.

There are several areas of ongoing work. Primarily, we aim to explore a more sophisticated mechanism for assessing context, and to consider alternative methods for enabling offspring to use their parent's context assessment in the donation decision. Our aim is to investigate whether the donation rate can be further improved, without relying on reciprocity. Further in the future we will explore incorporating a simple trust model to exploit the limited reciprocity that exists, even in the kind of large scale environment we consider. Finally, we aim to simulate our approach in a more realistic P2P setting, such as the file-sharing example used by HE [7].

REFERENCES

[1] R. D. Alexander, *The Biology of Moral Systems*, Aldine de Gruyter, 1987.
[2] P. D. Allison, 'The cultural evolution of beneficent norms', *Social Forces*, **71**(2), 279–301, (1992).
[3] R. Boyd and P. J. Richerson, 'The evolution of indirect reciprocity', *Social Networks*, **11**, 213–236, (1989).
[4] R. Dawkins, *The Selfish Gene*, Oxford University Press, 1976.
[5] B. Edmonds and D. Hales, 'Replication, replication and replication — some hard lessons from model alignment', *Journal of Artificial Societies and Social Simulation*, **6**(4), (2003).
[6] N. Griffiths, 'Tags and image scoring for robust cooperation', in *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS-08)*, (2008, to appear).
[7] D. Hales and B. Edmonds, 'Applying a socially inspired technique (tags) to improve cooperation in P2P networks', *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, **35**(3), 385–395, (2005).
[8] J. H. Holland, *Hidden order: How adaptation builds complexity*, Addison-Wesley, 1995.
[9] A. Jøsang, R. Ismail, and C. Boyd, 'A survey of trust and reputation systems for online service provision', *Decision Support Systems*, **43**(2), 618–644, (2007).
[10] M.A. Nowak and K. Sigmund, 'Evolution of indirect reciprocity by image scoring', *Nature*, **393**, 573–577, (1998).
[11] M.A. Nowak and K. Sigmund, 'Evolution of indirect reciprocity', *Nature*, **437**, 1291–1298, (2005).
[12] S. D. Ramchurn, D. Huynh, and N. R. Jennings, 'Trust in multi-agent systems', *Knowledge Engineering Review*, **19**(1), 1–25, (2004).
[13] P. Resnick, R. Zeckhauser, E. Friedman, and K. Kuwabara, 'Reputation systems', *Communications of the ACM*, **43**(12), 45–48, (2000).
[14] R. Riolo, M. Cohen, and R. Axelrod, 'Evolution of cooperation without reciprocity', *Nature*, **414**, 441–443, (2001).
[15] G. Roberts and T. N. Sherratt, 'Does similarity breed cooperation?', *Nature*, **418**, 499–500, (2002).

An Argumentation-based Computational Model of Trust for Negotiation

Maxime Morge[1]

Abstract. The fact that open multiagent systems are vulnerable with respect to malicious agents poses a great challenge: the detection and the prevention of undesirable behaviours. That is the reason why techniques such as trust and reputation mechanisms have been proposed. In this paper, we explore the cognitive science background which captures the notions of trust, reputation and confidence to provide a computational trust mechanism applied to negotiations within artificial societies. For this purpose, we formalize here these notions and we apply to them a particular argumentation technology for allowing agents to initiate, evaluate, reason, decide, and propagate reputation values.

1 Introduction

In the last decades, multiagent systems have been proposed as a new paradigm of computation based on cooperation and autonomy. The decentralization and the openess are main characteristics of these systems. There is no central point of control and new agents can enter/leave the system during the execution. Therefore, these systems are vulnerable with respect to malicious agents. This fact poses a great challenge: the detection and the prevention of undesirable behaviours. That is the reason why techniques such as trust and reputation mechanisms have been proposed.

In this paper, we explore the cognitive science background which captures the notions of trust, reputation, and confidence to provide computational trust mechanisms applied to negotiations within artificial societies. For this purpose, we formalize here these notions and we apply to them the argumentation technology proposed in [6] for allowing agents to initiate, evaluate, reason, decide, and propagate reputation values.

The paper is organised as follows. In Section 2, we introduce the walk-through example to motivate our approach. Section 3 presents the background of our proposal. We focus on the notion of interpersonal confidence and we characterize it. Section 4 introduces a conceptual framework for analyzing the decision problem related to the confident behaviour of agents. Section 5 presents our computational Argumentation Framework (AF) for decision making. Section 7 outlines the social interaction amongst agents. This interaction is illustrated by an example of information-seeking dialogue to propagate reputation values. The paper is summarised in section 8 where we also discuss related and future work.

2 Walk-through example

In order to illustrate our model, we consider an e-procurement scenario where two different types of agents, referred to below as A-type and B-type agents, negotiate a service. We consider here a specific case where A-type agents look for a service S. The A-type agents requesting S are `Al` and `Alice`; the B-type agents providing S are `Bob` and `Barbara`. `Carla` is a C-type agent which is an external and neutral observer. A-type agents are responsible for selecting B-type agents. A possible contract may concerns the concrete service $\mathtt{S}(\mathtt{c})$ and may be between `Al` and `Bob`. `Al`'s goal consists of finding and agreeing to a service S provided by a B-type agent. According to its preferences and constraints, the undermining of the quality specified in the contract is preferred than the overcharge of the price specified in the contract. Taking into account its goal and preferences/constraints, `Al` needs to solve a decision-making problem where the decisions amount to a service and a provider for that service. On the other hand, the goal of the provider `Bob` consists of agreeing to provide a service. According to `Bob`'s preferences and constraints, the seller must pay the service he buys. Taking into account its goal and preferences/constraints, `Bob` needs to solve a decision-making problem where the decisions amount to a service and a client for that service. The two decision making processes take place in a dynamic setting, whereby information about other agents, the services they require/provide and the characteristics of these services is obtained incrementally within the dialogues. The outcome of this process may be the contract obliging `Bob` to deliver on time the service to `Al`.

Within our computational model of trust for negotiation, the reasoning of each agent is supported by argumentation. In the remainder of this paper, we will focus on the reasoning of `Al`. In the concrete use case, `Al`, as a requester, uses argumentation for collecting information on the providers. For instance, `Al` can ask `Carla` its opinion about `Bob` and can ask to justify it (by providing an argument for it). `Al`, as a requester, uses argumentation for deciding which provider it selects taking into account its preferences/constraints and possibly the inconsistency of information it has gathered. Moreover, through argumentation, the participants provide an interactive and intelligible explanation of their choices. For instance, `Al` can argue that the selection of `Bob` is justified since the latter will not overcharge the price. Thus, agents can use argumentation for reasoning about trust for negotiating.

3 Reputation model

Cognitive science provides a pertinent and well-grounded background for developing computational models of trust. In this section, we focus on the notion of interpersonal confidence. We characterize it in order to provide a reasoning tool.

According to the Merriam-Webmester, the confidence is the " faith or belief that one will act in a right, proper, or effective way". In

[1] Universitá di Pisa, Italy, email: morg@di.unipi.it

the literature, this notion is defined in many different ways by psychologists, sociologists, economists. We restrict ourself here on the *interpersonal confidence*, which is the confidence of an agent about another agent in accordance with the information the first one has about the interactions of the second one with other agents. Considering the interpersonal confidence, [7] distinguishes the beliefs about the confidence that we called *reputation*, the reasoning with these beliefs, and the confident behaviour which is deduced. Considering a reputation, [7] distinguishes a *target*, i.e. the agent which is observed and evaluated, the *observers* i.e. the agents which observe the target, and the *evaluator*, i.e. the agent which evaluates and benefits of the reputation. Actually, we consider here the case where the observation of the target can be done by several agents, possibly *propagated* (i.e. communicated) to the evaluator.

Let us consider the properties of the reasoning over reputation that we will consider in this paper. The reputation is *subjective*, since it depends on the beliefs of the evaluator. The reputation is *uncertain*, since agents needs to accept information with reserve. The reputation is *multidimensional*, since it depends on the competences, the honesty, the foreseeability, the integrity, and the good will of the target. The reputation is *dynamic*, since it depends on the interactions of the target with other agents. The reputation is *defeasible*, since it can be challenged or reinstantiated in the light of new information not previously available. In this paper, we consider that the reputation is *qualitative* and not quantitative, since an evaluator can trust more a target than another one but there is no unit to measure reputation.

Amongst the computational model of trust and reputation, [7] distinguishes:

- the *decision tools*, i.e. which compute the reputation values and the confident behaviour;
- the *evaluation tools*, i.e. decision tools with a revision mechanism to update the reputation values in the light of new information;
- the *reasoning tools*, i.e. evaluation tools with a propagation mechanism to communicate reputation values;

Obviously, we want to provide here a reasoning tool.

4 Decision analysis

Our methodology is to decompose the complex problem related to the confident behaviour of agents into elements that can be analyzed and can be brought together to create an overall representation. We use here *influence diagrams* which are simple graphical representations of decision problems [3] including the decisions to make amongst the possible courses of action (called *decision nodes*, represented by squares), the value of the specific outcomes that could result (called *value nodes*, represented by rectangles with rounded corners), and the uncertain events which are relevant information for decision making (called *chance nodes*, represented by ovals). In order to show the relationship amongst these elements, nodes are put together in a graph connected by arrows, called *arcs*. We call a node at the beginning of an arc a *predecessor* and one at the end of an arc a *successor*. The nodes are connected by arcs where predecessors are independent and affect successors. Influence diagrams which are properly constructed have no cycles. In order to capture multi-criteria decision making, it is convenient to include additional nodes (called *abstract value nodes*, represented by double line) that aggregate results from predecessor nodes. While a *concrete value* is specified for every possible combination of decisions and events that feed into this node, an abstract value is specified for every possible combination of values that feed into this node, and so the multiple attributes are represented with a hierarchy of values where the top, abstract values aggregate the lower, concrete values. We assume that influence diagrams are provided by users via a GUI which allows them to communicate user-specific preferences.

We consider here the decision problem related to the confident behaviour of `Al` (cf Fig. 1). Throughout the paper we adopt the following convention: variables are in italics and constants are in typescript font. The evaluation of the contract (`contract`) depends on the selection of the proposal from the provider x about the service y. This top main value is is split into abstract values, the provision of the service (`provision`) and the provider (`reputation`). The evaluation of the service depends on its cost (`cost`) and its quality (`qos`). The evaluation of these criteria depends on the agent knowledge, namely the information about the services provided by the interlocutors (e.g. `price` and `warranty`). The evaluation of the partners depends on its reliability (e.g. competences or honesty) to deliver the service with the same quality (respectively price) as specified in the contract, denoted `rqos` (respectively `rcost`). The evaluation of the provider is influenced by its expected behaviour (`will`(x, do)) which depends on the testimonials about it previous behaviour, `Test`$(tid, x, done)$. Each testimonial `tid` can be provided by a neutral (`type`$(\texttt{tid}, \texttt{neutral})$) or a concurrent agent (`type`$(\texttt{tid}, \texttt{concurrent})$). The interaction of the testimonial `tid` can be direct (`int`$(\texttt{tid}, \texttt{direct})$) or observed (`int`$(\texttt{tid}, \texttt{observed})$). Moreover, the testimonial can be more or less recent, `time`$(\texttt{tid}, t)$. The user represented by the agent `Al` also provides, through the GUI, her preferences and constraints. For instance, the undermining of the quality specified in the contract is preferred than the overcharge of the price specified in the contract. Since concurrent agent can hide or lie, we prefer testimonials from neutral agents. In this way, we consider sociological information about the roles of agents. Even if witness information is usually the most abundant, direct experience is the most reliable source of information.

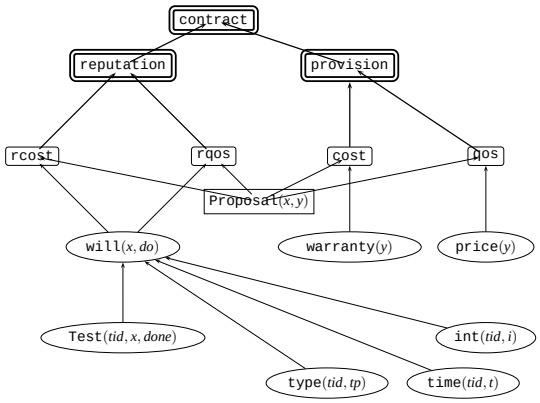


Figure 1. Influence diagram for the confident behaviour.

5 Argumentation framework

According to the approach of defeasible argumentation of [4], arguments are reasons supporting claims which can be defeated[2] by other arguments.

Definition 1 (AF) *An* argumentation framework *is a pair* $\texttt{AF} = \langle \mathcal{A}, \texttt{defeats} \rangle$ *where* $\mathcal{A}$ *is a finite set of arguments and* `defeats`

[2] The defeat relation is called attack in [4].

is a binary relation over $\mathcal{A}$. We say that a set S of arguments defeats an argument a if a is defeated by at least one argument in S.

[4] also analysis when a set of arguments is collectively justified.

Definition 2 (Semantics) *A set of arguments $\mathsf{S} \subseteq \mathcal{A}$ is:*

- conflict-free *iff* $\forall \mathsf{a}, \mathsf{b} \in \mathsf{S}$ *it is not the case that* a *defeats* b*;*
- admissible *iff* S *is conflict-free and* S *defeats every argument* a *such that* a *defeats some arguments in* S*.*

For simplicity, we restrict ourself to admissible semantics.

5.1 Decision framework

Since we want to instantiate our AF for our example, we need to specify a particular framework capturing the decision problem.

Definition 3 (Decision framework) *A* decision framework *is a tuple $\mathcal{D} = \langle \mathcal{L}, \mathcal{A}sm, \mathcal{I}, \mathcal{T}, \mathcal{P} \rangle$, where:*

- $\mathcal{L}$ *is the* object language *which captures the statements about the decision problem;*
- $\mathcal{A}sm$*, is a set of sentences in $\mathcal{L}$ which are taken for granted, called* assumptions*;*
- $\mathcal{I}$ *is the* incompatibility relation*, i.e. a binary relation over atomic formulas which is asymmetric. It captures the mutual exclusion between the statements;*
- $\mathcal{T}$ *is the* theory *which gathers the statements;*
- $\mathcal{P} \subseteq \mathcal{T} \times \mathcal{T}$ *is a (partial or total) preorder over $\mathcal{T}$, called the* priority *relation, which captures the uncertainty of beliefs, the priority amongst goals, and the expected utilities of the decisions.*

In the object language $\mathcal{L}$, we distinguish six disjoint components:

- a set of *abstract goals* (resp. *concrete goals*), i.e. some propositional symbols which capture the abstract values (resp. concrete values) that could result;
- a set of *decisions*, i.e. some predicate symbols which capture the decision nodes;
- a set of *alternatives*, i.e. some constants symbols which capture the mutually exclusive actions for each decision;
- a set of *beliefs*, i.e. some predicate symbols which capture the chance nodes;
- the *names* of rules in $\mathcal{T}$ which are unique.

In $\mathcal{L}$, we consider strong negation (classical negation) and weak negation (negation as failure). A strong literal is an atomic first-order formula, possible preceded by strong negation $\neg$. A weak literal is a literal of the form $\sim L$, where L is a strong literal.

We explicitly distinguish *assumable* (respectively *non-assumable*) literals which can (respectively cannot) be taken for granted, meaning that they can or cannot be assumed to hold as long as there is no evidence to the contrary. Decisions (e.g. $\texttt{Proposal}(x, y) \in \mathcal{A}sm$) as well as some beliefs (e.g. $\texttt{int}(tid, \texttt{observed}) \in \mathcal{A}sm$) can be taken for granted. In this way, $\mathcal{D}$ can capture incomplete knowledge.

The *incompatibility relation* captures the conflicts. We have $L \mathcal{I} \neg L$, $\neg L \mathcal{I} L$, and $L \mathcal{I} \sim L$ but we do not have $\sim L \mathcal{I} L$. We say that two sets of sentences Φ_1 and Φ_2 are incompatible ($\Phi_1 \mathcal{I} \Phi_2$) iff there is at least one sentence ϕ_1 in Φ_1 and one sentence ϕ_2 in Φ_2 such as $\phi_1 \mathcal{I} \phi_2$.

A *theory* gathers the statements about the decision problem.

Definition 4 (Theory) *A* theory $\mathcal{T}$ *is an extended logic program, i.e a finite set of rules R: $L_0 \leftarrow L_1, \ldots, L_j, \sim L_{j+1}, \ldots, \sim L_n$ with $n \geq 0$, each L_i being a strong literal in $\mathcal{L}$. The literal L_0, called the* head *of the rule, is denoted $\mathsf{head}(R)$. The finite set $\{L_1, \ldots, \sim L_n\}$, called the* body *of the rule, is denoted $\mathsf{body}(R)$. The body of a rule can be empty. In this case, the rule, called a* fact*, is an unconditional statement. R, called the unique* name *of the rule, is an atomic formula of $\mathcal{L}$. All variables occurring in a rule are implicitly universally quantified over the whole rule. A rule with variables is a scheme standing for all its ground instances.*

For simplicity, we will assume that the names of rules are neither in the body nor in the head of the rules thus avoiding self-reference problems. Considering a decision problem, we distinguish:

- *goal rules* of the form R: $G_0 \leftarrow G_1, \ldots, G_n$ with $n > 0$. Each G_i is a goal literal in $\mathcal{L}$. The head of the rule is an abstract goal (or its strong negation). According to this rule, the abstract goal is promoted (or demoted) by the goal literals in the body;
- *epistemic rules* of the form R: $B_0 \leftarrow B_1, \ldots, B_n$ with $n \geq 0$. Each B_i is a belief literal of $\mathcal{L}$. According to this rule, B_0 is true if the conditions $B_1, \ldots, B_n$ are satisfied;
- *decision rules* of the form R: $G \leftarrow D(a), B_1, \ldots, B_n$ with $n \geq 0$. The head of the rule is a concrete goal (or its strong negation). The body includes a decision literal ($D(a) \in \mathcal{L}$) and a set of belief literals possibly empty. According to this rule, the concrete goal is promoted (or demoted) by the decision $D(a)$, provided that conditions $B_1, \ldots, B_n$ are satisfied.

Due to our representation of decision problems, we assume that the elements in the body of rules are independent, the decisions do not influence the beliefs, and the decisions have no side effects.

In order to evaluate the previous statements, all relevant pieces of information should be taken into account, such as the uncertainty of knowledge, the priority between goals, or the expected utilities of the decisions. In this work, we consider that all rules are potentially defeasible and that the priorities are extra-logical and domain-specific features. We consider that the *priority* $\mathcal{P}$ which is a reflexive and transitive relation considering possible *ex æquo*. $R_1 \mathcal{P} R_2$ can be read "R_1 has priority over R_2". $R_1 \not\mathcal{P} R_2$ can be read "R_1 has no priority over R_2", either because R_1 and R_2 are *ex æquo* or because R_1 and R_2 are not comparable. The priority over concurrent rules depends on the nature of rules. Rules are *concurrent* if their heads are identical or incompatible. We define three priority relations:

- the priority over *goal rules* comes from the *preferences* overs goals. The priority of such rules corresponds to the relative importance of the combination of (sub)goals in the body as far as reaching the goal in the head is concerned;
- the priority over *epistemic rules* comes from the *uncertainty* of knowledge. The prior the rule is, the more likely the rule holds;
- the priority over *decision rules* comes from the *expected utility* of decisions. The priority of such rules corresponds to the expectation of the conditional decision in promoting/demoting the goal literal.

In order to illustrate the previous notions, let us consider the goal rules, the epistemic rules, and the decision rules of our example which are represented in Tab. 1. According to the decision rules, the goal rcost (respectively rqos) is reached if the partner does not overcharge the price (respectively undermine the quality) of the service, r_{31} (respectively r_{41}). According to the goal rules, the undermining of the quality specified in the contract is preferred than

the overcharge of the price specified in the contract. Indeed, achieving the goals rcost and rqos are required to reach reputation (cf r_{134}). However, these constraints can be relaxed and the achievement of rqos can be relaxed ($r_{134}\mathcal{P}r_{13}$). According to the epistemic rules, the testimonials from neutral agents are preferred to the testimonials from concurrent agents ($\mathsf{r}_1\mathcal{P}r'_1$) and the direct interactions are more reliable than the observed interactions ($\mathsf{r}_2\mathcal{P}\mathsf{r}'_2$).

5.2 Arguments

Since we want that our AF not only suggests some actions but also provides an intelligible explanation of them, we adopt here the tree-like structure for arguments proposed in [10] and we extend it with suppositions on the missing information.

Definition 5 (Argument) *An* argument *built upon $\mathcal{D}$ is composed by a conclusion, a top rule, some premises, some suppositions, and some sentences. These elements are abbreviated by the corresponding prefixes. An argument a can be:*

1. *a* hypothetical argument *built upon an unconditional ground statement. If L is an assumable literal (possibly its negation), then the argument built upon a ground instance of this assumable literal is defined as follows*[3]*: $\mathsf{conc}(\mathsf{a}) = L$, $\mathsf{top}(\mathsf{a}) = \theta$, $\mathsf{premise}(\mathsf{a}) = \emptyset$, $\mathsf{supp}(\mathsf{a}) = \{L\}$, $\mathsf{sent}(\mathsf{a}) = \{L\}$.*
   or
2. *a* built argument *built upon a rule such that all the literals in the body are the conclusion of arguments.*

   (a) *If f is a fact in $\mathcal{T}$ (i.e. $\mathsf{body}(f) = \emptyset$), then the* trivial *argument a built upon this fact is defined as follows: $\mathsf{conc}(\mathsf{a}) = \mathsf{head}(f)$, $\mathsf{top}(\mathsf{a}) = f$, $\mathsf{premise}(\mathsf{a}) = \emptyset$, $\mathsf{supp}(\mathsf{a}) = \emptyset$, $\mathsf{sent}(\mathsf{a}) = \{\mathsf{head}(f)\}$.*

   (b) *If r is a rule in $\mathcal{T}$ with $\mathsf{body}(r) = \{L_1, \ldots, L_j, \sim L_{j+1}, \ldots, \sim L_n\}$ and there is a collection of arguments $\{\mathsf{a}_1, \ldots, \mathsf{a}_n\}$ such that, for each strong literal $L_i \in \mathsf{body}(r)$, $\mathsf{conc}(\mathsf{a}_i) = L_i$ with $i \leq j$ and for each weak literal $\sim L_i \in \mathsf{body}(r)$, $\mathsf{conc}(\mathsf{a}_i) = \sim L_i$ with $i > j$, we define the* tree *argument a built upon the rule r and the set $\{\mathsf{a}_1, \ldots, \mathsf{a}_n\}$ of arguments as follows: $\mathsf{conc}(\mathsf{a}) = \mathsf{head}(r)$, $\mathsf{top}(\mathsf{a}) = r$, $\mathsf{premise}(\mathsf{a}) = \mathsf{body}(r)$, $\mathsf{supp}(\mathsf{a}) = \cup_{\mathsf{a}' \in \{\mathsf{a}_1,\ldots,\mathsf{a}_n\}} \mathsf{supp}(\mathsf{a}')$, $\mathsf{sent}(\mathsf{a}) = \cup_{\mathsf{a}' \in \{\mathsf{a}_1,\ldots,\mathsf{a}_n\}} \mathsf{sent}(\mathsf{a}') \cup \mathsf{body}(r) \cup \{\mathsf{head}(r)\}$. The set of arguments $\{\mathsf{a}_1, \ldots, \mathsf{a}_n\}$ are called the set of* subarguments *of a (denoted $\mathsf{sbarg}(\mathsf{a})$).*

The set of arguments built upon $\mathcal{D}$ is denoted $\mathcal{A}(\mathcal{D})$.

Notice that the subarguments of a tree argument concluding the weak literals in the body of the top rule are hypothetical arguments. Indeed, the conclusion of an hypothetical argument could be a strong or a weak literal while the conclusion of a built argument is a strong literal. As in [10], we consider composite arguments, called *tree* arguments, and atomic arguments, called *trivial* arguments. Contrary to other definitions of arguments (set of assumptions, set of rules), our definition considers that the different premises can be challenged and can be supported by subarguments. In this way, arguments are intelligible explanations. Moreover, we consider *hypothetical* arguments which are built upon missing information or a decision. In this way, our framework allows to reason further by making suppositions related to the unknown beliefs and over possible decisions.

In our example, the argument b (respectively a), Bob concludes that Bob will (respectively not) overcharge the service if we suppose that the interaction was observed. The tree arguments a and b are composed of four subarguments: one hypothetical argument concluding that the interaction is observed and three trivial arguments concluding the other premises.

6 Interactions

The interactions amongst arguments may come from their conflicts, from their nature (hypothetical or built), and from the priority of rules. We examine in turn these different sources of interaction.

Since their sentences are conflicting, the arguments interact with one another. For this purpose, we define the following attack relation.

Definition 6 (Attack relation) *Let $\mathsf{a}, \mathsf{b} \in \mathcal{A}(\mathcal{D})$ be two arguments.* a attacks b *iff $\mathsf{sent}(\mathsf{a})\ \mathcal{I}\ \mathsf{sent}(\mathsf{b})$.*

This relation encompasses both the direct (often called *rebuttal*) attack due to the incompatibility of the conclusions, and the indirect (often called *undermining*) attack, i.e. directed to a "subconclusion". According to this definition, if an argument attacks a subargument, the whole argument is attacked.

Since arguments are more or less hypothetical, we define the size of their suppositions.

Definition 7 (Supposition size) *Let $\mathsf{a} \in \mathcal{A}(\mathcal{D})$ be an argument. The* size of suppositions *for a, denoted $\mathsf{suppsize}(\mathsf{a})$, is the number of suppositions of a: $\mathsf{suppsize}(\mathsf{a}) = |\mathsf{supp}(\mathsf{a})|$.*

The size of suppositions for an argument is the number of decision literals and assumable belief literals in the sentences of the argument.

Since arguments have different natures (hypothetical or built) and the top rules of built arguments are more or less strong, we define the strength relation as follows.

Definition 8 (Strength relation) *Let A_1 be a hypothetical argument, and A_2, A_3 be two built arguments.*

1. *A_2 is stronger than A_1 (denoted $A_2\ \mathcal{P}^{\mathcal{A}}\ A_1$);*
2. *If $(\mathsf{top}(A_2)\mathcal{P}\mathsf{top}(A_3)) \wedge \neg(\mathsf{top}(A_3)\mathcal{P}\mathsf{top}(A_2))$, then $A_2\ \mathcal{P}^{\mathcal{A}}\ A_3$;*
3. *If $(\mathsf{top}(A_2)\mathcal{P}\mathsf{top}(\mathrm{A}_3)) \wedge (\mathsf{suppsize}(\mathrm{A}_2) < \mathsf{suppsize}(\mathrm{A}_3))$, then $A_2\ \mathcal{P}^{\mathcal{A}}\ A_3$;*

Since $\mathcal{P}$ is a preorder on $\mathcal{T}$, $\mathcal{P}^{\mathcal{A}}$ is a preorder on $\mathcal{A}(\mathcal{T})$. Since it is preferable to consider fewer suppositions as possible, built arguments are preferred to hypothetical arguments. Moreover, we want to take into account the preferences captured by the priorities. That is the reason why we consider that an argument is stronger than another argument if the top rule of the first argument has a proper higher priority than the top rule of the second argument, or if it is not the case but the number of suppositions made in the first argument is properly smaller than the number of suppositions made in the second argument.

In order to adopt Dung's seminal calculus of opposition, we define the defeat relation.

Definition 9 (Defeat relation) *Let $\mathsf{a}, \mathsf{b} \in \mathcal{A}(\mathcal{D})$ be two arguments.* a defeats b *iff: i) a attacks b; ii) $\neg(\mathsf{b}\ \mathcal{P}^{\mathcal{A}}\ \mathsf{a})$.*

[3] θ denotes that no literal is required.

$\mathcal{T}$

$r_1(tid)$: will$(x, do) \leftarrow$ Test(tid, x, do), type$(tid,$ neutral$)$, int(tid, i), time(tid, t)
$r_2(tid)$: will$(x, do) \leftarrow$ Test(tid, x, do), type(tid, tp), int$(tid,$ direct$)$, time(tid, t)

$r'_1(tid)$: will$(x, do) \leftarrow$ Test(tid, x, do), type$(tid,$ concurrent$)$, int(tid, i), time(tid, t)
$r'_2(tid)$: will$(x, do) \leftarrow$ Test(tid, x, do), type(tid, tp), int$(tid,$ observed$)$, time(tid, t)

f_1: Test$(\text{Carla}_1,$ Bob, $\neg$overcharge$)$
f_2: Test$(\text{Carla}_1,$ Bob, $\neg$underquality$)$
f_3: type$(\text{Carla}_1,$ neutral$)$
f_4: time$(\text{Carla}_1, 1)$
f_5: Test$(\text{Alice}_1,$ Bob, overcharge$)$
f_6: Test$(\text{Alice}_1,$ Bob, underquality$)$
f_7: type$(\text{Alice}_1,$ concurrent$)$
f_8: time$(\text{Alice}_1, 1)$
f_9: Test$(\text{Al}_1,$ Barbara, overcharge$)$
f_{10}: Test$(\text{Al}_1,$ Barbara, $\neg$underquality$)$
f_{11}: type$(\text{Al}_1,$ neutral$)$
f_{12}: int$(\text{Al}_1,$ direct$)$
f_{13}: time$(\text{Al}_1, 3)$
f_{14}: Test$(\text{Carla}_2,$ Barbara, $\neg$overcharge$)$
f_{15}: Test$(\text{Carla}_2,$ Barbara, underquality$)$
f_{16}: type$(\text{Carla}_2,$ neutral$)$
f_{17}: time$(\text{Carla}_2, 2)$
f_{18}: price$($d, high$)$
f_{19}: warranty$($d, low$)$
f_{20}: price$($c, low$)$
f_{21}: warranty$($c, high$)$
f_{22}: price$($e, low$)$
f_{23}: warranty$($e, low$)$
f_{24}: price$($f, high$)$
f_{25}: warranty$($f, high$)$

$\mathcal{T}$

r_{012}: contract $\leftarrow$ reputation, provision
r_{134}: reputation $\leftarrow$ rcost, rqos
r_{256}: provision $\leftarrow$ cost, qos

r_{01}: contract $\leftarrow$ reputation
r_{13}: reputation $\leftarrow$ rcost
r_{25}: provision $\leftarrow$ cost

r_{02}: contract $\leftarrow$ provision
r_{14}: reputation $\leftarrow$ rqos
r_{26}: provision $\leftarrow$ qos

$\mathcal{T}$

r_{31} : rcost $\leftarrow$ Proposal(x, y), will$(x, \neg$overcharge$)$
r_{41} : rqos $\leftarrow$ Proposal(x, y), will$(x,$ underquality$)$
r_{51} : cost $\leftarrow$ Proposal(x, y), price$(y,$ low$)$
r_{62} : qos $\leftarrow$ Proposal(x, y), warranty$(y,$ high$)$

r_{52} : cost $\leftarrow$ Proposal(x, y), price$(y,$ high$)$
r_{61} : qos $\leftarrow$ Proposal(x, y), warranty$(y,$ low$)$

Table 1. The epistemic rules (at left), the goal rules (at upper right), and the decision rules (at lower right).

Let us consider our previous example. The arguments a and b attack each other. Since the top rules of a is r_1 and the top rule of b is r'_1, a is stronger than b, and so a defeats b. If we consider now the whole problem, there is an argument concluding that contract is reached if we consider the service S(c) provided by Bob due to the provision of the service and the reputation of the potential suppliers. This argument is in an admissible set, and so this proposal can be adopted and justified by Al.

7 Social interaction

The social statements are exchanged during dialogues and notified in the *dialogical commitments*. Our agent drives the interactions by the adherence to protocols.

The computation of the reputation values, the confident behaviour, the revision mechanism and the propagation mechanism are driven according to the individual/social statements concerning the goals of agents (their own goals and the goals of their interlocutors), the decisions they make, the knowledge, and preferences over them. The social statements are exchanged during dialogues and notified in the *dialogical commitments* which are internal data structures which contain propositional/action social obligations involving the agent, namely with the agent being either the debtor or the creditor. The choice amongst actions is made according to the agent's statements and the preferences over them. The dialogical commitments of Al include commitments involving Al: either Al is the creditor of the commitment, for instance a commitment is added when Bob suggests a concrete service; or Al is the debitor of the commitment, for instance a commitment is added when Al accepts it.

In this way, agents reasons and take decision about the proposals and arguments which are exchanged during the dialogues in accordance to the reputation values. For instance, Al can built an argument concluding that its goal related to the cost of the service is reached if the price of the concrete service $S(\text{c})$ is low. This argument is useful for Al to justify its choice, $S(\text{c})$, in front of Bob.

Dialogue protocols are required to conduct the interaction. For this purpose, the social reasoning uses a boot strap mechanism that initiates the required protocol, the role the agent will play in that protocol, and the other participants. The protocol engine determines the appropriate message to be sent given those parameters. When there is a decision to be made either between the choice of two locutions (e.g. an accept or a challenge) to be sent or the instantiation of the content of the locution (e.g. the definition of an assert), the protocol engine uses a precondition mechanism to prompt the reasoning of the agent. Upon the satisfaction of the precondition, the protocol engine sends the locution. A similar mechanism is used for incoming messages. If it is necessary to update the commitments of the agent, this can be done with the post condition mechanism which operates in a similar manner.

The agents utter messages to exchange goals, decisions, and knowledge. The syntax of messages is in conformance with a common agent communication language. We assume that each message: has an identifier, M_k; is uttered by a speaker (S_k); is addressed to a hearer (H_k); responds to a message with identifier R_k; is characterised by a speech act A_k composed of a locution and a content. The locution is one of the following: question, assert, accept, why, withdraw (see Table 2 below for examples). The content is a triple consisting of: a goal G_k, a decision D_k, and a knowledge K_k[4]

Fig. 2 depicted our information-seeking protocol from the initiator viewpoint with the help of a deterministic finite-state automaton. The choice of locutions to send depends on the way the reasoning fulfills preconditions. For example, the following rule dictates whether the protocol engine sends accept, assert or why:

```
IF receive assert(G,D,K) from inter THEN {
  update commit(interlocutor,[G,D,K]);
  IF evaluate(G,D,K) THEN{
```

[4] We will use θ to denote that no goal is given and $\emptyset$ to denote that no knowledge is provided.

```
        send accept(G,D,K) to inter;
        commit(me,[G,D,K]);}
ELSEIF send why(G,D,K) to inter;}
```

In this rule `me` denotes the reasoning agent and `inter` denotes the agent it dialogues with. `evaluate(G,D,K)` is a predicate which evaluates if the goal G is supported by an admissible argument built upon the decision D and the knowledge K. According to this rule, the dialogical commitments are updated when a proposal is received. If an admissible proposal have been suggested, then the speech act is an `accept`. Otherwise the speech act is a `why`.

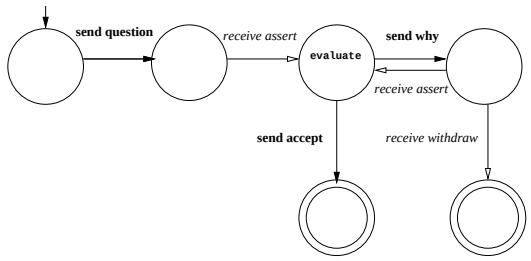


Figure 2. Information-seeking protocol for the initiator.

Tab. 2 shows the speech acts exchanged between `Al` and `Carla` playing an information-seeking dialogue. The first move is for `Al` to pose a question to `Carla`, M_0. This locution seeks the expected behaviour `Bob`. This expected behaviour is good (cf M_1) and argued by the testimonial that `Bob` did not overcharge the price specified in the contract in a previous interaction. Therefore, `Al` consider `Bob` as a trusted supplier and they can negotiate.

M_k	S_k	H_k	A_k	R_k
M_0	`Al`	`Carla`	`question`(`rcost`, `Proposal`(`Bob`, y), [`will`(`Bob`, *do*)])	θ
M_1	`Carla`	`Al`	`assert`(`rcost`, `Proposal`(`Bob`, y), [`will`(`Bob`, $\neg overcharge$)])	M_0
M_2	`Al`	`Carla`	`why`(`rcost`, `Proposal`(`Bob`, y), [`will`(`Bob`, $\neg overcharge$)])	M_1
M_3	`Carla`	`Al`	`assert`(`rcost`, `Proposal`(`Bob`, y), [[`Test`($\texttt{Carla}_1$, `Bob`, $\neg$`overcharge`), `time`($\texttt{Carla}_1$, 1)])	M_2
M_4	`Al`	`Carla`	`accept`(`rcost`, `Proposal`(`Bob`, y), [`Test`($\texttt{Carla}_1$, `Bob`, $\neg$`overcharge`), `time`($\texttt{Carla}_1$, 1)])	M_3

Table 2. Information seeking dialogue

8 Conclusions

In this paper, we have proposed a computational model of trust for negotiation. For this purpose, we have provided an AF for decision-making to perform the reasoning of agents about the reputations. In order to valid this approach, we use MARGO[5].

As the computational model of trust proposed by [8], our model uses the reputation values to guide the negotiation, The computational model of trust in [8] guides the negotiation by (i) choosing the partners, (ii) devising the set of negotiable issues, and (iii) determining negotiation intervals. Our computational model of trust guides the negotiation by collecting information on the partners and considering the trust issues to chooses one of the partners.

In order to compare our computational model of trust, we can consider the analysis grid of [9]. The game theoretical approach is the predominant paradigm for the design of computational model of trust. These models have given good results for simple scenarios but the reduction of reputation to a risk probability in decision making is too restrictive in scenarios as considered in the ARGUGRID[6] project. For this purpose, we have adopted a social and cognitive approach where the mental states that leads to trust another agent as well as its consequences and its propagations are essential. Our model takes into account direct experiences (direct and observed interactions), witness information (also called word-of-mouth), and sociological information (the fact that the observer is a concurrent). However we do not consider prejudice, i.e. the use of signs that identify the agent as a member of a group. In this paper, we have considered the reputation values as subjective (rather than global) and multi-context (i.e. multidimensional). Our model assumes that agents can hide or lie (level 2 in agent behaviour assumptions of [9]). Our model can consider boolean information as well as continuous measures if we adopt the extension of our AF for quantitative preferences [5]. The information is composed rather than aggregates through a dialectical process.

We have used here the argumentation-based mechanism for decision making proposed in [6]. The framework of [1, 6] incorporates abduction on missing information, while the frameworks of [2, 6] can be applied to a multi-criteria decision making. To the best of our knowledge, the framework of [6] is the only one integrating both of these proprieties required by our application. Moreover, the other existing frameworks do not come with a conceptual framework for creating a model and a representation of decision problems. By relying on [6], the decision problem related to the confident behaviour is firstly analyzed, and so treated.

ACKNOWLEDGEMENTS

This work is supported by the Sixth Framework IST programme of the EC, under the 035200 ARGUGRID project. We would like to thank the referees for their comments which helped improve this paper.

REFERENCES

[1] A.K. and P. Moraitis, 'Argumentative-based decision-making for autonomous agents', in *Proc. of AAMAS*, pp. 883–890. ACM Press, (2003).
[2] L. Amgoud, 'A general argumentation framework for inference and decision making', in *Proc. of the 21st Conference on Uncertainty in Artif. Intell.*, AUAI Press, pp. 26–33, (2005).
[3] R. T. Clemen, *Making Hard Decisions*, Duxbury. Press, 1996.
[4] P.M. Dung, 'On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games', *Artif. Intell.*, **77**(2), 321–357, (1995).
[5] *Arguing about quantitative aspects of decisions*, ed., M. Morge, ARGUGRID report, 2008.
[6] M. Morge, 'The hedgehog and the fox. an argumentation-based decision support system', in *Argumentation in Multi-Agent Systems*, volume 4946 of *Lecture Notes in Artificial Intelligence*, pp. 114–131. Springer-Verlag, (2008).
[7] G. Muller, *Utilisation de normes et de réputations pour détecter et sanctionner les contradictions*, Ph.D. dissertation, ENS Mines de Saint Etienne, Decembre 2006.
[8] S.D. Ramchurn, N.R. Jennings, C. Sierra, and L. Godo, 'Devising a trust model for multi-agent interactions using confidence and reputation', *Int. J. of Applied Artificial Intelligence*, **18**(9-10), 833–852, (2004).
[9] J. Sabater, 'Review on computational trust and reputation models', *Artif. Intell. Review*, **24**(1), 33–60, (2005).
[10] G. Vreeswijk, 'Abstract argumentation systems', *Artif. Intell.*, **90**(1-2), 225–279, (1997).

[5] http://margo.sourceforge.net

[6] http://www.argugrid.eu

Handling Mitigating Circumstances for Electronic Contracts

Simon Miles[1] and **Paul Groth**[2] and **Michael Luck**[3]

Abstract. Electronic contracts are a means of representing agreed responsibilities and expected behaviour of autonomous agents acting on behalf of businesses. They can be used to regulate behaviour by providing negative consequences, *penalties*, where the responsibilities and expectations are not met, i.e. the contract is *violated*. However, long-term business relationships require some flexibility in the face of circumstances that do not conform to the assumptions of the contract, that is, *mitigating circumstances*. In this paper, we describe how contract parties can represent and enact policies on mitigating circumstances. As part of this, we require records of what has occurred within the system leading up to a violation: the *provenance* of the violation. We therefore bring together *contract-based* and *provenance* systems to solve the issue of mitigating circumstances.

1 Introduction

Commitments between business parties are generally regulated through *contracts*. These documents allocate responsibility for particular outcomes, allow parties to know what to expect of each other and provide a basis for redress should those responsibilities and expectations not be met. In many contexts, autonomous software agents can be used to advantageously represent businesses' interests in an automated way, including preparing, agreeing on, reasoning over, acting on and enforcing contracts in an electronic form. Much research has been conducted on how best to instantiate *contract-based systems* [2, 3, 10].

For the purposes of this paper, we consider a contract to be a set of *clauses*, each of which states some responsibility of an agent. A clause may state an *obligation*, a *prohibition* or a *permission*. The set of agents to which clauses apply are called the *contract parties*. One crucial aspect of an autonomous approach to electronic contracting is the handling of *violations* of a contract clause, i.e. where the stated responsibilities have not been fulfilled. There are different ways that a violation could be dealt with. For example, most contract-based systems will include a notion of payments, and so violations may automatically incur financial penalties.

However, relationships in business are important and a company that handled all violations of a contract clause equally could damage its long-term relationships with partners. In situations in which unexpected circumstances have led a contract party to be unable to fulfil their responsibilities, other parties may act more leniently than they are contractually permitted to, to maintain the long-term business relationship. Such circumstances are called *mitigating circumstances*.

In current electronic contracting approaches, mitigating circumstances are addressed (if at all) by passing the decision on how to handle a violation up to a human. However, organisations often have standard, if not publicised, policies for handling mitigating circumstances, and so automation is certainly possible. We would like to extend contract-based systems to allow agents to autonomously consider, and react appropriately to, mitigating circumstances.

A pre-requisite to providing this extended functionality is the ability to determine whether there were, or may have been, mitigating circumstances for a violation, which requires reliable documentation of what has occurred and how that *caused* the violation. It is only through such documentation that mitigating circumstances will be evident. The problem of obtaining the relevant documentation of a violation's causes is exacerbated by the fact that violations may only be dealt with some time after they occurred, for instance where it is only through the accumulation of multiple failures over time that a contract clause is violated.

In this paper, we describe how recording and reasoning over the causes of violations can help to better manage the behaviour of parties in the system. This allows contracting parties to handle problems more flexibly, and encourage better coordination. Specifically, the technical contributions of this paper are as follows.

- An algorithm for handling mitigating circumstances in contract violation based on technologies for electronic contracting and for determining the *provenance* of violations, i.e. what caused them to occur.
- A re-usable model for expressing mitigating cirumstance policies.
- Application of this algorithm and model to an aerospace scenario.

The rest of the paper is structured as follows. Section 2 describes a motivating example application in the aerospace domain. Section 3 introduces our electronic contracting approach, and discusses the use of *provenance* to determine the cause of violations. Section 4 then details our algorithm, which is applied to the example application in Section 5. We finish the paper with a discussion of related work in Section 6 and conclusions in Section 7.

2 Example Scenario

Our example scenario is based on the aftercare market for aircraft engines. It is an extended version of that considered by Lost Wax's Aerogility application [1].

[1] Department of Computer Science, King's College London, UK, email: simon.miles@kcl.ac.uk
[2] Information Sciences Institute, University of Southern California, USA, email: pgroth@isi.edu
[3] Department of Computer Science, King's College London, UK, email: michael.luck@kcl.ac.uk

2.1 Contract

In this scenario, aircraft operators (e.g. airlines) establish contracts with engine manufacturers whereby the manufactures are obliged to ensure the aircraft have engines in working order. To achieve this, an engine manufacturer regularly removes an engine from an aicraft for which it is responsible and replaces it with an already serviced engine to allow the aircraft to continue flying. This replacement must be performed in a timely fashion, so that the aircraft remains usable. As well as regular servicing, the engine manufacturer must respond to possible faults in an engine by similarly replacing it. Once removed, an engine is serviced and then returned to the pool of engines available for swapping into other aircraft.

The core contract between aircraft operator and engine manufacturer specifies the following:

- An engine requires servicing after every X flights, as well as when its health data indicates a possible fault.
- When an aircraft's engine requires servicing, the engine manufacturer must remove the engine and replace it with a serviced one.
- The aircraft operator is permitted to penalise the engine manufacturer if an aircraft is left on the ground for more than Y hours due to an engine not being available.

The core contract may be extended by extra constraints on the engine manufacturer in particular cases.

- Engines are ultimately composed of parts supplied by parts manufacturers. An aircraft operator may constrain an engine manufacturer to only use parts from given named suppliers in engines used in their aircraft.
- For best use of resources in fulfilling multiple contracts, an engine manufacturer will often take and service an engine from one operator's aircraft and put it into the aircraft of another operator. In some cases, one operator may not trust another. An operator may therefore constrain an engine manufacturer never to put engines into their aircraft that have been previously used by a particular other operator's aircraft.

2.2 Mitigating Circumstances

Where an aircraft has been grounded due to lack of working engine, an aircraft operator will want to recoup their costs by penalising the manufacturer. However, the two companies wish to retain a good working relationship, and particular mitigating circumstances may be considered. Whether the operator makes these circumstances clear to the manufacturer in advance is a choice of the individual business.

In this scenario, we consider two mitigating circumstances.

Late Health Data A manufacturer is aware of a potential fault in an engine through analysing the engine's health data. This is recorded in the aircraft, and so must be supplied by the operator. If supplied late, the manufacturer is delayed in servicing the engine.

Part Supplier Late If an operator restricts the manufacturer as to where it can source engine parts, and the required part manufacturer was late in supplying parts, then this can affect the manufacturer's ability to provide a working engine on time.

2.3 Managing Violations

The way that violations of the contract are handled should depend on circumstances. In the scenario, one or more of the following broad actions can be performed by the aircraft operator given a violation (e.g. aircraft remaining on the ground too long).

Full Penalty Operator deducts 30% from the monthly payment to the engine manufacturer.

Reduced Penalty Operator sends a formal notice reprimanding the manufacturer but acknowledging mitigating circumstances.

Reconsider Policy Operator starts reconsideration of its constraining policies in the contract.

The choice of a specific action is entirely based on the goals of the business, and is out of scope of this paper. For convenience, we assume that a reduced penalty will be the action taken in all subsequent examples. We now discuss the two primary technologies that our algorithm for handling mitigating circumstances depends upon.

3 Contracts and Provenance

Our approach brings together two technologies, described in detail below. The application is based on *contract-based systems* to support regulation of agents' behaviour through explicit contracts agreed between agents. The policies for mitigating circumstances use *provenance systems* to record documentation on what occurs within a system and use this to determine why a particular violation occurred.

3.1 Contract-Based Systems

A contract-based system is one in which agents agree to documents which specify requirements (obligations and prohibitions) or permissions on their behaviour. For our purposes, we define a *contract* to be an assignment of *clauses* to agents that have agreed to fulfil them.

For agents, acting on behalf of multiple organisations, to set up and rely on contracts for mutual benefit, we require a supporting infrastructure. This can be expressed in terms of agents playing administrative roles, such as for storing contracts to ensure access to them and for preserving their integrity over their lifetime. It may also include *monitor* roles, which require the agents playing them to check that clauses are being fulfilled and, where they are not, to notify the *enforcer* of that contract clause, i.e. the agent responsible for handling that clause's violation.

Current work, such as that conducted in the CONTRACT project [2], has begun to bring together existing technologies to specify contract languages, frameworks for defining contract-based applications, administrative architectures containing those infrastructural roles needed to manage the contracts and model checking techniques for verifying that agents in an application are able to fulfil its contractual responsibilities. In this paper, we assume the presence of a contract language and administrative infrastructure. In general, we will not refer to these further, as they are out of scope of the work. However, the monitoring of the fulfilment of contract clauses is a vital part of understanding the context in which a violation occurs. For explanatory purpose, we will assume a single agent playing a monitor role for checking the fulfilment of all contract clauses. In reality, it is often the case that many such agents need to exist for an application, as monitoring may use and/or produce information private to individual contract parties.

3.2 Provenance and Causation

As previously stated, reliable documentation is necessary in order to determine whether the causes of a violation are sufficient to mitigate it. Thus, we need to be able to determine the *provenance* or the what *caused* a particular event (e.g. violation) to occur as it did. In the study of art, the provenance of an artwork can include the artist,

the materials used in creating it, the restoration done over time, the different locations where it has been stored or exhibited and so on. All of these ultimately caused the artwork to be as it is now. In prior research, we studied provenance in the context of a wide range of sciences [8]. Knowing the provenance of results is important in science experiments for many purposes, e.g. peer reviewers determining if an experiment was rigorous and sound, understanding where an error may have occurred which affected results, re-use of configuration of successful experiments, etc.

With regard to the violation of a contract clause, causes can include the actions, or absence of actions, of the responsible party, but may also include actions of other agents and occurrences more widely within the application environment.

Determining the provenance of an occurrence therefore requires data on its causes. As we often do not know in advance that something particular will occur, agents must *record* what occurs and causal connections between occurrences around the time that they happen. The documentation forms a *causal graph*, depicting where A was caused by B, which was caused by C etc. Below, we will denote that A was caused by B (A is effect, B is cause) as $\boxed{A} \rightarrow \boxed{B}$

In order to ensure the availability of such a causal graph for our algorithm, we only consider applications that have be made *provenance-aware*, which entails that most, if not all, software agents are designed or adapted to record what they do and what caused them to do it (messages received, their goals etc.) [9]. In a contract-based environment, this includes both the contract parties and the infrastructure agents, such as the monitor. When it is not possible to record the causal connections between occurrences, it is often possible to infer that they exist from what has been recorded.

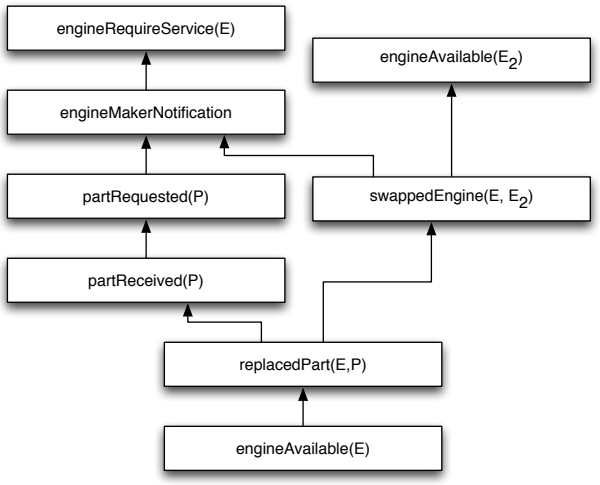


Figure 1. Provenance of an engine as a causal graph

To illustrate how provenance provides better understanding of an occurrence, we describe the provenance of engine being made available after servicing in Figure 1. We begin at the top of the figure. Originally, it is determined that an engine, E, requires service. This occurrence causes the engine maker to be notified. The engine maker then requests and receives a part, P. In parallel to the engine maker being notified, another engine becomes available. Together the occurrences of a engine becoming available and the engine maker being notified cause the engine, E, to be swapped out for engine E_2. Once engine E is taken out of the airplane, its defective part is replaced with the part ordered by the engine maker. This replacement of parts causes E to be made available once again for use in other aircraft.

4 Handling Mitigating Circumstances

In this section, we bring together the contract-based and provenance technologies described above to give an algorithm for handling mitigating circumstances when a violation is detected. We summarise the algorithm below, and then describe each step in more detail.

1. **Violation Detection** From checking the environment, monitor determines a clause was violated, informs relevant enforcer.
2. **Cause Determination** The enforcer uses heuristics to infer possible undocumented causes of the violation.
3. **Mitigating Circumstances** The enforcer uses policies to determine, from the recorded and inferred causes of the violation, whether there were mitigating circumstances.
4. **Remedy** If mitigating circumstances were found, then, again according to its policy, the enforcer acts to remedy the situation and ensure that violations are less likely to occur in future.

4.1 Violation Detection

In order to detect the violation, the monitor must observe its environment on the basis of what is expected from fulfilling the contract clause. When the violation occurs, it notifies the enforcer for that clause. The enforcer is often a party to the contract, i.e. the agent that gains from the clause's fulfilment.

Being provenance-aware, the monitor records several pieces of documentation about what it does: the clause-related observation of the environment, the signalling of a violation, and the causal connection between the two (the former causes the latter).

4.2 Cause Determination

When a violation has occurred, the enforcer first checks whether there are undocumented causes it can infer from the available documentation. This provides a more complete picture from which it can then determine whether there were mitigating circumstances. Inference is achieved by applying *inference rules*.

An inference rule expresses a heuristic by which the enforcer determines new causal connections from existing facts. Its antecedent is an expression composed of parametrised predicates, its consequent takes the form of causal graph edges between occurrences. The antecedent's predicates are facts from one of four sources:

Domain Knowledge Timeless knowledge about the domain available to the enforcer.
Contract Clauses Clauses of the contract that has been violated.
Contract Party Documentation Documentation recorded by the contract parties of what they know to have occurred.
Monitor Documentation Documentation recorded by the monitor of what it knows to have occurred.

The antecedent may also contain mathematical expressions resolving to true or false based on the predicate variables, e.g. $A > B$. The consequent of an inference rule contains a set of causal connections between predicates from the antecedent (i.e. known occurrences).

An example of a whole rule is given below. In this rule, the antecedent is a conjunction of two facts documented by agents in the

system and a relationship between them. The facts are that an engine's health data was received at time T_1 and that the aicraft with that engine was unserviced at time T_2; the relationship expresses that T_2 was less than 10 hours after T_1. The consequent of the rule, i.e. that implied by any pattern of occurrences matching the former facts, is that the fact that the engine was not serviced at T_1 was caused by the health data being received at T_2. Whenever the documented facts of a violation match the antecedent, the consequential causal connection will be added to the facts from which mitigating circumstances will be assessed.

Antecedent
$receivedHealthData(E, T_1) \wedge$
$unserviced(A, E, T_2) \wedge$
$T_2 < T_1 + 10$
Consequent
$unserviced(A, E, T_2) \rightarrow receivedHealthData(E, T_1)$

4.3 Mitigating Circumstances

Determination of mitigating circumstances is achieved by a *policy* setting out where the causes of a violation suggest mitigating circumstances, and what action to take in each such case. Such a policy could be included as part of a contract document, in which case other parties may use it to reason about what they can get away with, or may be private to the owning contract party, if they prefer to keep the mitigating circumstances considerations secret. There are likely to be several different kinds of mitigating circumstance, such as the three given for the example in Section 2.2. For each kind, there will be a pre-condition and a remedy.

The pre-condition is a causal graph between occurrences, in the form of a tree with a violation occurrence as its root. All occurrences in the tree can be parametrised with variables. As a whole, the pre-condition graph acts as a template for chains of causes of a particular form leading to a violation. The template graph is then matched against the documentation recorded and inferred. If they match, mitigating circumstances have been found, and the remedy enacted.

An example of a mitigating circumstances policy statement is given below. The precondition is a tree of causal connections from the violation of a clause concerning a particular aircraft. The precondition is a template which can be matched against documented, or inferred, facts. In this case, the violation must have been caused by the aircraft's engine not being serviced at a given time, which in turn must have been caused by the engine health data being received at a given time. If this pattern is found within the documentation, then the pre-condition is matched, the policy applies, and the appropriate action is taken: reduced penalty, in this case.

Precondition	**Remedy**
$violation(A) \rightarrow$ $unserviced(A, E, T_2) \rightarrow$ $receivedHealthData(E, T_1)$	Reduced Penalty

4.4 Remedy

For the purposes of this paper, we consider the remedy to be a simple action by the enforcer, such as reducing the penalty that would otherwise be placed on the violating agent. In future work, we will consider more sophisticated mechanisms, such as negotiating to adjust the contract to more realistically suit the working environment.

4.5 Algorithm for Handling Mitigating Circumstances

The algorithm can be expressed in pseudo-code as follows.

- C: the set of contract clauses of which to detect violations
- R: the set of inference rules
- G: a graph (V, E) where vertexes represent occurrences and edges causal relationships between them
- G_{KB}: the union of the knowledge sources (domain, contract, contract party, monitor)
- P: a mapping from violation types to sets of policies concerning those types
- A: an array of actions to be taken indexed by a policy

- RETRIEVEVIOLATION() - retrieves a violation from the monitior
- RETRIEVEOBSERVATION(v) - retrieves the observation that caused a violation
- CONSEQUENTOF(r) - retrieves the edge representing the consequent of a rule
- APPLYRULE(r, G) - apply an inference rule, r to the graph G
- UNION(G_1, G_2) - perform a union between the two graphs (i.e. combine there edges and vertexes)
- EXTRACTSUBGRAPH(v, G) - given a vertex extract the subgraph beginning at that vertex
- TEMPLATEISOMORPHISM(G_1, G_2) - determine whether the two graphs are isomorphic, implementations may define isomorphism in terms of attributes associated with vertexes and edges
- EXECUTE(a) - execute a given action, a

```
VIOLATIONDETECTION(C)
1  if the monitor detects a violation of clause, cl ∈ C
2     then v ← RETRIEVEVIOLATION()
3          c_v ← RETRIEVEOBSERVATION(v)
4          return new graph edge (v, c_v)
```

```
CAUSEDETERMINATION(c_v, R, G_KB)
1  G_KB^new = ∅
2  for each inference rule r ∈ R
3       do (e, c) ← CONSEQUENTOF(r)
4          if e = c_v
5            then G ← APPLYRULE(r, G_KB)
6                 G_KB^new ← UNION(G_KB^new, G)
7  return G_KB^new
```

```
MITIGATINGCIRCUMSTANCES(v, P, G_KB)
1  for each policy p ∈ P[v]
2  G_{c_v} ← EXTRACTSUBGRAPH(v, G_KB)
3  if TEMPLATEISOMORPHISM(G_v, p)
4     then return p
```

```
REMEDY(p, A)
1  a ← A[p]
2  EXECUTE(a)
```

5 Applying to the Example

In this section, we apply our algorithm to the scenario presented in Section 2. We start by defining the facts that may be documented or

Predicate	Description
Domain Knowledge	
$owns(A, O)$	Operator O owns aircraft A
Contract Clauses	
$constrainedPartSupplier(S, P)$	Contractual obligation to use supplier S for part P
$disallowedPriorUse(O)$	Contractual prohibition from using engines previously used by operator O
Contract Party Documentation	
$engineAvailable(E)$	Engine E is serviced and available for use
$partReceived(S, P, T)$	Manufacturer received part P from supplier S at time T
$partRequested(S, P, T)$	Manufacturer requested part P from supplier S at time T
$receivedHealthData(E, T)$	Manufacturer received health data about engine E at time T
$replacedPart(E, P, T)$	Part P was replaced in engine E at time T
$swappedEngine(A, E_1, E_2, T)$	Engine E_1 was removed, engine E_2 inserted into aircraft A at time T
Monitor Documentation	
$unserviced(A, E, T)$	Engine E of aircraft A requires but has not received servicing at time T
$violation(A)$	Violation of the contractual obligation regarding servicing aircraft A

Table 1. Example knowledge predicates

inferred by an aircraft operator agent in the scenario. These are expressed using predicate logic and described in Table 1. Using statements of this form, we can construct propositions about what is documented or believed at any one time.

We now describe two use cases in which there are mitigating circumstances that the aircraft operator, acting in the role of enforcer, takes into account, matching those described in Section 2.2. In both use cases below, the monitoring mechanism discovers that an engine has not serviced at a given time, even though it contractually should have been. In each case, a different mitigating circumstance has occurred, and so a reduced penalty is applied. For each use case, we show how the algorithm in the previous section is applied.

5.1 Late Health Data

The following operation of the contract parties is documented.

- The engine manufacturer, as part of its operation, receives the health data for an engine at a given time: $receivedHealthData(e, t_1)$. This is the engine of an aircraft, a, which has earlier been recorded as requiring servicing.

Violation Detection The monitor determines that an aircraft requires servicing but has not been serviced at this moment: $unserviced(a, e, t_2)$. It further determines that, contractually, it should have been determined before now. It therefore, reports a violation: $violation(a)$. The causal connection between these two occurrences is documented: $\boxed{violation(a)} \rightarrow \boxed{unserviced(a, e, t_2)}$.

Cause Determination The operator believes that the health data should have been received at least 10 hours in advance for the manufacturer to be able to complete the job in time. This belief implies a causal connection between an engine not being serviced and that engine's health data being received late (the former was due to the latter). The operator first infers the causal connection from the available data using the following inference rule.

Antecedent
$receivedHealthData(E, T_1) \wedge$ $unserviced(A, E, T_2) \wedge$ $T_2 < T_1 + 10$
Consequent
$\boxed{unserviced(A, E, T_2)} \rightarrow \boxed{receivedHealthData(E, T_1)}$

Mitigating Circumstances From this, we then have a sequence leading to a violation matching the pre-condition of the following rule: receiving engine health data at a given time caused the engine not to be serviced, which caused a violation.

Precondition	Remedy
$\boxed{violation(A)} \rightarrow$ $\boxed{unserviced(A, E, T_2)} \rightarrow$ $\boxed{receivedHealthData(E, T_1)}$	Reduced Penalty

5.2 Part Supplier Late

The following operation of the contract parties is documented.

- The contract constrains the manufacturer to use a given part supplier for parts of a particular type: $constrainedPartSupplier(s, p)$.
- At some point, the manufacturer requires a part of this type and orders it from the supplier: $partRequested(s, p, t_1)$.
- The supplier eventually provides the part: $partReceived(s, p, t_2)$.
- The engine manufacturer is required to service an engine that requires a part of the above type. When the part is available, the manufacturer puts the new part into the engine: $replacedPart(e_2, p, t_3)$.
- This repaired engine is later used to swap into an aircraft requiring a service: $swappedEngine(a, e_1, e_2, t_4)$.
- A causal chain is recorded: the engine swap required the replacement of the part, which required the part to be received, which required the part to be requested from the supplier, which was made to that supplier because of the contract clause: $\boxed{swappedEngine(a, e_1, e_2, t_4)} \rightarrow \boxed{replacedPart(e_2, p, t_3)} \rightarrow \boxed{partReceived(s, p, t_2)} \rightarrow \boxed{partRequested(s, p, t_1)} \rightarrow \boxed{constrainedPartSupplier(s, p)}$

Violation Detection The monitor determines that an aircraft requires servicing but has not been serviced at this moment: $unserviced(a, e_2, t_5)$. It further determines that, contractually, it should have been determined before now. It therefore, reports a viola-

tion: $violation(a)$. The causal connection between these two occurrences is documented: $\boxed{violation(a)} \rightarrow \boxed{unserviced(a, e_2, t_5)}$.

Cause Determination The operator believes that if the part supplier supplied a part late, this can lead to problems servicing aircraft (specifically, the part is supplied less than 48 hours beore servicing is due). This belief expresses a causal connection between the engine not being serviced and the part being received late. It first infers the causal connection from available data using the following rule.

Antecedent
$partReceived(S, P, T_2) \wedge$
$swappedEngine(A, E_1, E_2, T_4) \wedge$
$replacedPart(E_2, P, T_3) \wedge$
$unserviced(A, E_2, T_5) \wedge$
$T_3 < T_1 + 48$
Consequent
$\boxed{unserviced(A, E_2, T_5)} \rightarrow \boxed{partReceived(S, P, T_2)}$

Mitigating Circumstances From this, we have a sequence leading to a violation matching the pre-condition of the rule below: a constraint on the part supplier caused a supplier to be used in requesting a part which caused the part to be delivered at a particular time (late) which caused the engine not to be serviced, which caused a violation.

Precondition	**Remedy**
$\boxed{violation(A)} \rightarrow$ $\boxed{unserviced(A, E_2, T_5)} \rightarrow$ $\boxed{partReceived(S, P, T_2)} \rightarrow$ $\boxed{partRequested(S, P, T_1} \rightarrow$ $\boxed{constrainedPartSupplier(S, P)}$	Reduced Penalty

6 Related Work

In recent work on normative systems and agreement in service-oriented architectures, *norms* specifying patterns of behaviour for agents, *contract clauses* as concrete representations of dynamic norms, management or enforcement of norms itself being a norm, are all already established in the literature [2, 3, 7, 10]. Such work has focused on the infrastructure needed to support such systems and handling of violations is often through the mechanism of immediately issuing contractually fixed penalties. There are notable exceptions, e.g. longer-term issues are considered by Duran et al. [4], who examine how observation of fulfilment and violation of obligations can feed into a longer-term assessment of agents through *testimonials*.

There have been many recent approaches to the recording causal documentation so that the provenance of occurrences can be determined. It is applicable to a wide range of applications [8], and has particularly been considered in the context of workflow enactment, i.e. automatically recording documentation as each step of a workflow is executed [5, 11]. In our own work we have examined how provenance can be used to interpret and ask questions about the validity of experimental results [6].

7 Conclusions

When a contract clause between parties is violated, a single fixed penalty is an inflexible way to manage the situation. In many real world cases, the party permitted to enact the penalty may wish to take into account *mitigating circumstances*, for the sake of the long-term business relationship. Mitigating circumstances, and how to act in case of them, can be expressed in a policy document, but in order to judge circumstances against the policy we need a reliable record of what led to the violation occurring.

In this paper, we have provided an algorithm, and accompanying data structures, for evaluating whether violations were caused by mitigating circumstances, and acting accordingly. This makes use of a contract-based framework, by which we can define the contract clauses, and provenance technology, by which agents can document the *causes* of what occurs. In combination, this allows us to express and enact mitigating circumstances policies. We have shown how this applies in a concrete example in the aerospace domain.

This is preliminary work, which needs to be tested in practical applications. Future work will concern the re-usability of (parts of) mitigating cirumstances policies, and methodological guidelines for constructing them, both aimed at easing the process of implementing such policies in diverse applications.

ACKNOWLEDGEMENTS

This work was partly supported by the European CONTRACT IST project [2]. We would also like to thank Lost Wax, and Camden Holt in particular, for discussions on the aerospace use case.

REFERENCES

[1] Aerogility. http://www.aerogility.com/, 2007.
[2] IST CONTRACT project. http://www.ist-contract.org, 2007.
[3] Chrysanthos Dellarocas, 'Contractual agent societies: Negotiated shared context and social control in open multi-agent systems', in *Workshop on Norms and Institutions in Multi-Agent Systems, 4th International Conference on Multi-Agent Systems (Agents-2000)*, Barcelona, Spain, (June 2000).
[4] Fernanda Duran, Viviane Torres da Silva, and Carlos J. P. de Lucena, 'Using testimonies to enforce the behaviour of agents', in *AAMAS'07 Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN)*, eds., Jaime Sichman and Sascha Ossowski, pp. 25–36, Honolulu, Hawai'i, (May 2007).
[5] Juliana Freire, Claudio T. Silva, Steven P. Callahan, Emanuele Santos, Carlos E. Scheidegger, and Huy T. Vo, 'Managing rapidly-evolving scientific workflows', in *Proceedings of the International Provenance and Annotation Workshop 2006 (IPAW 2006)*, Lecture Notes in Computer Science. Springer, (2006). To appear.
[6] Paul T. Groth, *The Origin of Data: Enabling the Determination of Provenance in Multi-institutional Scientific Systems through the Documentation of Processes*, Ph.D. dissertation, University of Southampton, September 2007.
[7] Fabiola Lopez y Lopez, Michael Luck, and Mark d'Inverno, 'A normative framework for agent-based systems', *Computational and Mathematical Organization Theory*, **12**(2–3), 227–250, (2005).
[8] Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau, 'The requirements of using provenance in e-science experiments', *Journal of Grid Computing*, **5**, 1–25, (2007).
[9] Simon Miles, Steve Munroe, Michael Luck, and Luc Moreau, 'Modelling the provenance of data in autonomous systems', in *Proceedings of Autonomous Agents and Multi-Agent Systems 2007*, pp. 243–250, Honolulu, Hawai'i, (May 2007).
[10] Edward Muntaner-Perich, Josep Lluis de la Rosa, and Rosa Esteva, 'Towards a formalisation of dynamic electronic institutions', in *AAMAS'07 Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN)*, eds., Jaime Sichman and Sascha Ossowski, pp. 61–72, Honolulu, Hawai'i, (May 2007).
[11] Jun Zhao, Carole Goble, Robert Stevens, and Daniele Turi, 'Mining taverna's semantic web of provenance', *Concurrency and Computation: Practice and Experience*, (2007).

Automated Requirements-Driven Definition of Norms for the Regulation of Behavior in Multi-Agent Systems

Martin Kollingbaum [1] and **Ivan J. Jureta** [2] and **Wamberto Vasconcelos** [3] and **Katia Sycara** [4]

Abstract. The engineering of heterogenous distributed systems is a complex task. Traditional software engineering methods fail to account for new demands of flexibility and adaptability in the construction of software systems. On the other hand, concepts of Virtual Organizations and Electronic Institutions cater for the need of open, heterogenous software environments, where agents may dynamically organize themselves into organizational structures, determined by roles, norms and contracts. Our work aims to facilitate the engineering of heterogenous and distributed systems by providing only a specification of the desired overall system behavior, expressed as a set of norms, and rely on capabilities and properties of individual agents that allow them to dynamically form the desired complete software system. In particular, we present a framework, called *Requirement-driven Contracting* (RdC), for automatically deriving executable norms from requirements and associated relevant information. RdC facilitates the governance of MAS by ensuring that all requirements, along with runtime changes of requirements are appropriately and automatically reflected in the norms regulating the behavior of MAS.

1 INTRODUCTION

Specifying requirements is in general a difficult and critical task – even more so for heterogenous systems, in which software developed, maintained, and operated by, and distributed across various organizations should cooperate in order to achieve joint goals. The perceived quality of the future system is determined by the fit between its behavior and the requirements. Moreover, such systems are expected to continually operate and adapt to changes in highly volatile environments. Changes lead to situations in which requirements (i) may not all be known before/during development, (ii) become obsolete over the course of development, and (iii) vary at runtime. Currently, established requirements and software engineering processes are built with homogenous, closed systems in mind. Classical requirements and software engineering processes start from the goals of the system, then identify and specify operations whose execution satisfies the goals, and finally design and implement components (agents) capable of executing the said operations. If requirements or environment conditions change, redesign and redeployment occur – this usually takes long and is costly, often leading system owners to miss opportunities for which they introduced the systems in the first place.

[1] The Robotics Institute, Carnegie Mellon University, email: mkolling@cs.cmu.edu
[2] PReCISE, University of Namur, Belgium, email: iju@info.fundp.ac.be
[3] Dept. of Computing Science, University of Aberdeen, UK, email: wvasconcelos@acm.org
[4] The Robotics Institute, Carnegie Mellon University, email: katia@cs.cmu.edu

Addressing these problems requires a methodological shift from the assembly of passive components towards systems that are based on active autonomous entities. Research into multi-agent systems (MASs) supports such a methodological shift, as it investigates the creation of systems involving dynamic, heterogenous, distributed and autonomous agents.

Virtual Organizations (VOs) and Electronic Institutions facilitate the dynamic formation of agent-based systems by introducing an organizational structure into an agent community and prescribing certain "rules of engagement" or norms, to regulate the actions and interactions of these agents. Agents may take on roles in such an organizational structure by signing contracts and thereby committing to observe established behavioural standards, i.e., *norms*.

Norms, that is, obligations, permissions, and prohibitions are useful abstractions for expressing rights and duties, and thereby regulating the behavior of heterogenous agents that act on behalf and in interest of their owners. Norms are critical for VOs, for they specify the organizational structure: agents adopt roles within the VO, and thereby the norms ascribed to the roles. By enabling agents to process norm specifications, a *separation of concerns* occurs: agents are not only described at an individual level, in terms of their capabilities, but a separate specification is also introduced to describe, at a social level, the compulsory, allowed, or forbidden behaviors. Norm-governed VOs, therefore, are an attractive approach to the engineering of heterogenous systems in changing environments, as executable normative specifications dynamically direct and tune the behavior of heterogenous agents.

It is *only if norms are appropriately defined* that agents can fulfil the purpose of the VO. As the VO's purpose is defined by the requirements of VO stakeholders (i.e., owners, users, etc.), *norms are appropriate only if they regulate the VO so that these requirements are satisfied to the most desirable (and feasible) extent.* Given, e.g., a requirement for a Banking application engineered on VO principles, that a *Letter of Credit* cannot be issued if there is no deposit, norms must ensure that this is the case at runtime. Approaches to the engineering of norm-governed MAS [11, 6, 10] focus on writing norms, either using limited requirements conceptualizations or leaving out requirements-level notions. Stakeholders, however, use richer notions than norms to communicate requirements. Their statements instead speak of VO goals in terms of functionality to provide and quality to achieve, how to provide/achieve these, and what to comply to (e.g., Sarbanes-Oxley Act). Stakeholders have preferences over alternative functionalities and quality levels, and priorities over preferences that cannot be jointly satisfied to the highest extent.

We therefore encounter two challenges if we are to further facilitate VO engineering: (a) provide a rich set of concepts for representing requirements, and (b) automatically derive appropri-

ate norms from the requirements. We present a framework, called *Requirements-driven Contracting* (RdC) that responds to both of these challenges, by integrating rich requirements-level concepts, a corresponding specification language, and algorithms for automatically deriving norms from requirements-level information. RdC thereby ensures that all requirements, along with runtime changes thereof are appropriately and automatically reflected in the norms regulating a VO. The RdC proceeds in three steps: (1) The VO engineer elicits the stakeholders' natural language statements about the purpose of the system. According to the speech act in which each statement is given, the VO engineer identifies requirements, domain assumptions, preferences, and priorities (Sect.2). (2) Using a set of templates and formal language constructs, the VO engineer writes the *environment specification* (ES) to transform requirements into system goals (Figure 2), domain assumptions into domain constraints, and define preferences over goals and priorities over conflicting preferences (Sect.3). (3) The VO engineer inputs the ES into RdC algorithms (Figures 3 and 5) to obtain an executable specificatons of norms that subsequently regulate the VO (Sect.4).

We consider these steps of the RdC process in detail in Sect.2–4. Sect.5 overviews related work; Sect.6 outlines conclusions, limitations, and directions for future work.

2 UNDERSTANDING REQUIREMENTS

Consider the transaction in which a Letter of Credit (LoC) is issued by a banker agent for use by a customer agent to finance the acquisition of goods from a supplier agent. The customer makes a deposit with the bank, then receives an LoC. The banker informs the supplier that an LoC has been issued to the customer, so that the customer can provide the LoC to the supplier, who can subsequently transfer the goods to the customer. To obtain the funds, the supplier sends the LoC to the bank. To engineer the VO for this setting, the engineer first elicits stakeholders' statements in natural language about the process, determines whether each of the statements is a requirement or otherwise, subsequently produces a specification that is translated into contracts comprising norms. In doing so, the engineer moves across three levels of abstraction covered by RdC: the *requirements*, the *ES*, and the *contracts* level. Figure 1 shows key useful RdC notions (only those discussed herein) at each of these levels.

At Level 1, statements about the high-level requirements, which the VO must satisfy, are elicited. A *functional requirement*, such as "Issue an LoC", will communicate what is desired or intended, whereas a *domain assumption* will indicate what is already the case [12], and is therefore an assertive or a declaration. A *nonfunctional requirement*, such as "Quickly issue an LoC" places additional constraints on a functional requirement by communicating an attitude thereon (through an expressive). Nonfunctional requirements typically involve gradable adjectives, such as quick, convenient, secure, or useful, efficient, accurate, and so on. Expressives therefore communicate preferences in an indirect manner: asking for "quick" implies that faster is preferred to slower. The same applies, e.g., for security, efficiency, maintainability, usability, and so on. We make the preference order explicit and thus order both nonfunctional and functional requirements. Introducing preferences entails the need for *priorities*, as often all preferences cannot be satisfied by the MAS. When aiming to satisfy one preferred requirement affects negatively the ability to satisfy some other requirement, we say that the involved preferences are conflicting. In such a case, a priority is defined to indicate which of the two preferences it is more important to satisfy.

We classify any requirement also as either compulsory or optional, and either conditional or free. The MAS must satisfy *compulsory requirements*, while it will (hopefully, though not necessarily) satisfy *optional requirements*. A requirement is *conditional* if it needs to be satisfied only when some particular conditions hold; otherwise, it is *free*. A domain assumption is either *free* or *conditional*, in the same sense as a requirement can be conditional or free. All mentioned taxonomic dimensions are relevant as they affect how the requirements and assumptions transform into norms (e.g., compulsory requirements give obligations and sactions).

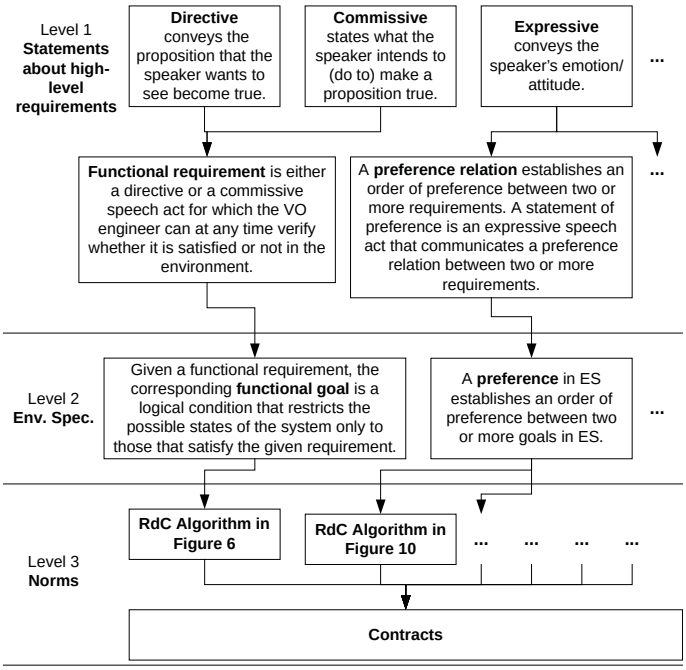


Figure 1. Part of RdC notions.

3 ENVIRONMENT SPECIFICATION

Given requirements, domain assumptions, preferences and priorities, we proceed to manually write the corresponding ES. The ES is divided into four parts. *Functional ES* specifies *functional goals* that need to be achieved (including *preferences* thereon), *plans* that should be followed, and *domain constraints* that must not be violated if the MAS is to satisfy functional requirements and domain assumptions. Achieving some functional goals requires that roles cooperate: we define *dependencies* between roles when the agent occupying a role can achieve a functional goal only if the agent occupying another role provides assistance. *Nonfunctional ES* define measures on the VO for monitoring purposes; *nonfunctional goals* define preferences over the values of these measures – if these values are observed at runtime, nonfunctional requirements and preferences thereon are satisfied. *Priorities ES* indicates *priorities* between conflicting preferences. Finally, *Terminological ES* specifies the domain ontology relevant for the VO, so as to better delimit the meaning intended for terms used throughout the ES, and subsequently carried over to norms. The ES is written as a collection of templates, one for each functional and nonfunctional goal (Figure 2), plan, domain constraint, priority, and dependency. Preferences are defined through the Preferences slot in templates (see, Figure 2; i.e., a preference order over functional goals appears in the Preferences slot of each of the goals in that preference order). Below, we discuss only functional goals and preferences thereon. The entire RdC framework is presented in the longer report [3].

3.1 Functional Goals

The compulsory and conditional functional requirement "Record a deposit in electronic and paper form if the deposit is received and approved" leads to the template in Figure 2. A unique identifier is given to the goal in UID for cross-referencing in ES and any graphical representation thereof. Type states whether the goal is functional or nonfunctional, compulsory (if the corresponding requirement is compulsory) or optional (if the requirement is optional), and conditional or free. Preferences gives the preference order in which the given goal appears. Element on the left hand side of $>>$ is preferred to the one on the right. As usual, the $>>$ relation is modeled as a strict partial order. A goal can be refined, that is, we can identify a set of goals (possibly easier to achieve) whose joint achievement is equivalent to achieving the refined goal. Supergoals lists those goals in whose refinement the given goal participates.

UID:	Record deposit in el. & paper form
Type:	Goal (Functional/Compulsory/Conditional)
Preferences:	(Record deposit in electronic form only $>>$ Record deposit in el. & paper form)
Supergoals:	*none*
BelongsTo:	Role (Banker)
Source:	Record a deposit in electronic and paper form if the deposit is received and approved.
Source type:	Requirement (Functional/Compulsory/Conditional)
Parameters:	d: Deposit
Satisfy:	$eL : elLog,\ pL : paLog\ logs(eL, pL, d) \leftarrow isPaperLog(pL, d), isElectronicLog(eL, d)$
Conditions:	$received(d), approved(d)$
Concepts:	Deposit = (and Amount (> 0) (exists has-purpose Credit) (all in-currency aset(USD,EUR))); paLog = (and Log (all has-purpose TransactionRecord) (all recorded-on Database)); elLog = (and Log (all has-purpose TransactionRecord) (all recorded-on Paper))

Figure 2. Instantiated Template for a Funct. Goal.

BelongsTo identifies the role or dependency to which the goal is associated. We then relate the goal to the requirement which it specifies: Source identifies the requirement at RdC Level 1 and SourceType indicates the classification of that requirement. Satisfy indicates the logical conditions that must be brought about in order to satisfy the goal's Source requirement. Logical conditions are written as Horn clauses. A general form of Horn clause is $a_0 \vee (\neg a_1) \vee \ldots \vee (\neg a_n)$, where each $a_i, i = 1, \ldots, n$ is an atom. We adopt here the standard notation: $a_0 \leftarrow a_1, \ldots, a_n$; whereby a_0 is true if $a_1, \ldots, a_n$ are true. Since the problem of testing a set of Horn clauses for satisfiability is known to have linear time solution algorithms, the ES supports rather efficient inferences compared to RE specification formalisms, which rely on variants of linear temporal first-order logic. For conditional goals, we write down in Conditions the logical conditions that must hold for the agents to know that the goal is to be achieved. Concepts lists the definitions of concepts whose instances are referred to in the template. We use the ITL (Information Terminological Lang. [7]) to define the domain ontology; therein, conceptual knowledge about a given domain is defined by a set of concepts and roles these concepts play in relationships, in which they take part. Each term intended to define a concept C is a conjunction of logical constraints, which are necessary for any object to be an instance of C.

4 NORMATIVE SPECIFICATION

We show in this section that the ES can be directly mapped to and expressed with normative concepts investigated in VO and Electronic Institutions research [5, 8]. Norms support the development of flexible as well as open VOs. Agents may join and leave VOs by adopting the normative standards of a VO via automated negotiation and signing of contracts. The normative specification defines contracts, whereby each contract is a set of norms. Norms explicitly specify behavioral directives for agents. Obligations, permissions and prohibitions of agents make requirements explicit at the normative level. Normative specifications, therefore, ensure that agents act only in ways that satisfy requirements and preferences, and are in accord with domain assumptions and priorities.

We use an available normative model [4], which is based on the notation outlined in [6]. We show how the ES relates to various governance measures, including norms, contracts, and roles and how they are derived from the ES. The building blocks of this notation are first-order terms, that is, constants, variables and functions (applied to terms). According to the chosen model, agents form social or organizational structures by taking on specific roles. These roles are determined by a set of norms, that is: the obligations the agent has to fulfill in the course of its actions and interactions with other agents, its prohibitions and permissions. The *role* concept allows us to abstract from individual agents and formulate patterns of behaviour that agents may adopt and conform to, with contracts defining these roles and the organizational structure of a VO. The set of norms Ω, determining the normative state of a complete VO, is described in the following way:

Definition 1. *A global normative state Ω is a finite and possibly empty set of norms.*

Ω describes the current overall normative state. Norms that are contained in this set are relevant to the agent – for example, an obligation contained in Ω must be fulfilled. When it is fulfilled, it has to be removed. Such a maintenance of the normative state has to be accommodated in order to capture the fact that requirements from the ES may be relevant to the overall system under specific circumstances only. A practical approach to the maintenance of a normative state is outlined in [2]. As a simplification, we add so-called *activation* and *expiration* conditions to norm specifications in order to capture circumstances when norms will be added to or removed from Ω.

In addition, the normative model we use introduces constraints. With that, the actual influence of norms on the agent achieving specific states of affairs can be restricted. These constraints are defined as follows:

Definition 2. *A constraint γ is any construct of the form $\tau \lhd \tau'$, where $\lhd \in \{=, \neq, >, \geq, <, \leq\}$.*

With that, we put forward following definition of norms:

Definition 3. *A norm ω is a tuple $\langle \nu, t_d, A, E \rangle$, where*

- *ν is any construct of the form $\mathsf{O}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ (an obligation), $\mathsf{P}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ (a permission) or $\mathsf{F}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ (a prohibition), where*
- *τ_1, τ_2 are terms, with τ_1 specifying a set of agents and τ_2 specifying a set of roles;*
- *φ is an atomic first-order formula, expressing the achievement of a state of affairs;*
- *$\gamma_i, 0 \leq i \leq n$, are constraints restricting the domains of variables occurring in φ;*
- *t_d is a time stamp recording the time of declaration of the norm;*
- *$A = \bigwedge_{j=0}^{n} \psi'_j, 0 \leq j \leq n$, is the activation condition comprising a conjunction of first-order predicates*
- *$E = \bigwedge_{k=0}^{n} \psi''_k, 0 \leq k \leq n$, is the expiration condition comprising a conjunction of first-order predicates*

In this formulation of norms, term τ_1 identifies the agent(s) to whom the norm is applicable. Term τ_2 is the role (or set of roles) of these agents. For example, $\mathsf{O}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n}\gamma_i$ thus represents an obligation on agent τ_1 taking up role τ_2 to bring about φ, subject to constraints γ_i, $0 \leq i \leq n$. The obligation *activates* (is added to Ω) when $\bigwedge_{j=0}^{n}\psi'_j$ holds, whereas the obligation deactivates (is removed from Ω) when $\bigwedge_{k=0}^{n}\psi''_k$ holds (the same holds for permissions and prohibitions). The γ_i's express constraints on those variables occurring in φ.

As pointed out earlier, the ES contains compulsory *and* optional goals. In both cases, an agent must be motivated to act. In addition to represent mandatory goals as obligations, means have to be put in place to *enforce* the fulfilment of obligations. A traditional means of enforcing law-abiding behavour in a social context is the specification of sanctions, i.e., actions typically performed by an authorised third party in case of norm violation. The purpose of sanctions is to either keep individuals from violating their duties or to compensate for an agent's behaviour. It is important to understand that sanctions are obligations for such an authorised party to act. As a consequence, the introduction of norms and contracts also requires the introduction of specific organizational structures where the role of such an authorised third party is established. Agents adopting such a role are permitted and obliged to pursue the activities defined by sanctions. In order to specify sanctions in a contract (or, in context of this paper, to generate them from the ES), obligations have to be specified for this specific authority role. Sanctions are then activated once a state of affairs indicating a violation of obligations holds. We define sanctions as follows:

Definition 4. *The concept of a sanction amounts to an obligation* $\mathsf{O}_{\tau_1:auth}\varphi \wedge \bigwedge_{i=0}^{n}\gamma_i$*, assigned to a specific Authority role* ***auth****, expressing an obligation for an agent in this role to achieve the state of affairs expressed by term* φ*.*

We ensure within RdC algorithms that sanctions are defined to react to violations of obligations, themselves derived from compulsory goals. In order to capture optional goals (corresponding to optional functional or nonfunctional requirements) with norms, we need the concept of *incentive*. Intuitively, an incentive motivates the agent to act in a certain way by indicating a reward for taking the desired actions. In our work, an incentive means that an agent will be motivated and not sanctioned if it is not successful. In other words, instead of rewarding, we do not sanction. Optional functional goals will therefore amount to obligations, for which sanctions are not defined, whereas compulsory functional goals will give rise to obligations with corresponding sanctions.

4.1 RdC Algorithms

As the purpose of contracts is to ensure that agents in a VO behave according to stakeholders' requirements and obey domain assumptions, we derive the normative specification, that is, contracts as sets of norms, using algorithms. The full framework contains algorithms for converting all ES-level information into norms. Below, we present two of them for transforming functional goals while accounting for preferences and dependencies. First, we have a simple definition for the *contract* concept:

Definition 5. *A contract* ***C*** *is of the form*

$$\langle\langle r_1, \{\omega_1^{r_1}, \ldots, \omega_m^{r_1}\}\rangle, \ldots, \langle r_q, \{\omega_1^{r_q}, \ldots, \omega_p^{r_q}\}\rangle\rangle$$

where each r *is a role identifier, and each* ω *a norm.*

For simplicity, we observe some notational conventions. Let FG be a functional goal, and FG.UID be the value of the UID slot in the template for FG (e.g., Figure 2). We refer to values in other template slots in the same obvious way (e.g., FG.Satisfy for the value of the Satisfy slot in the template for FG). Following earlier discussions, we assume that the value of the Satisfy and Uphold attributes is of the form $a_0 \leftarrow a_1, \ldots, a_n$ whereby $n \geq 1$ and a_0 can be empty. Also let the value of Conditions slot wherever it appears in ES be of the form $b_0 \leftarrow b_1, \ldots, b_m$ whereby $m \geq 1$ and b_0 can be empty. Recall that any $a_i, 0 \leq i \leq n$ and any $b_j, 0 \leq j \leq m$ is an atomic first-order formula. Also, we write $(\cdot)$ for content of no interest in the particular discussion or algorithm – e.g., if we write the norm $\langle(\cdot),(\cdot),\bigwedge_{j=0}^{n}\psi'_j,(\cdot)\rangle$, we are interested only in $\bigwedge_{j=0}^{n}\psi'_j$ while the content of the other elements can be any allowed by the norm definition.

4.1.1 Contracts from Functional Goals

In this section, we present the RdC algorithm, shown in Figure 3, for deriving a contract from a functional goal. We illustrate the output of the algorithm by converting the functional goal in Figure 2 into the contract shown in Figure 4. The algorithm proceeds as follows. Given the ES, we first select a functional goal that has not yet been subjected to the algorithm in Figure 3. We consider only primitive goals in the goal hierarchy. We generate as many obligations as needed to cover the Horn clause in the goal template's Satisfy slot, whereby each obligation deactivates as soon as it is realized (i.e., $\bigwedge_{k''=0}^{r''}\psi''_{k''} = a_i$). If the functional goal is conditional, the activation condition for each obligation corresponds to the condition for that functional goal (lines 3–8 in Figure 3). If the goal is also compulsory, we create sanctions, that is obligations for the authority role. Sanctions activate if the corresponding obligations are not realized (9–11). If the goal is optional, we create no sanctions (12–14). We then create a new contract (15–17), in which the roles vary if the functional goal belongs to a role or a dependency. The algorithm returns for each goal a set of norms according to the type of the functional goal. Obligations and sanctions are generated if the goal is compulsory, while no sanctions are created if the goal is optional. Contracts are created and norms are distributed between the role being supervised and the authority (i.e., supervisor) role. The supervisor role (Authority) receives responsibility for sanctions, while the supervised role receives obligations. The algorithm terminates, as each of the **for each** loops goes through finite sets and considers one element at a time. We have the contract $\langle\langle Banker, \{\omega_1, \omega_2\}\rangle, \langle BankerAuth, \{\omega_1^{sanct}, \omega_2^{sanct}\}\rangle\rangle$ on the functional goal "Record deposit in electronic & paper form" with norms and sanctions shown in Figure 4.

The $BankAuth$ is the role that acts as the authority for the $Banker$ role. The functional goal gives us two obligations for the $Banker$ role and the corresponding sanctions enforced by the $BankAuth$ role. $\mathsf{O}_{a:BankAuth}\phi'$ and $\mathsf{O}_{a:BankAuth}\phi''$ are defined by the VO engineer; these obligation determine how the sanction is applied in case of violation.

4.1.2 Preferences over Contracts on Functional Goals

In this section, we present the RdC algorithm for adjusting norms according to preferences, shown in Figure 5. Preferences establish an order over requirements, and therefore over goals in the ES. The preference order defined over functional goals corresponds to a preference order over contracts. We do not, however, carry over the very

Contract from Functional Goal

`Input` : One functional goal FG such that: (i) FG has not been transformed previously using this algorithm; (ii) there are no functional goals in ES for which FG is the supergoal.

`Output` : One contract **C** including norms for ensuring that logical conditions of FG must or can be brought about.

`begin`

`1` **For each** $a_i, 1 \leq i \leq n$ in FG.Satisfy **do**

`2` Create obligation $\omega_i = \langle \mathsf{O}_{a:r}\phi \wedge \bigwedge_{k=1}^{r} \gamma_k, t_d, \bigwedge_{k'=0}^{r'} \psi'_{k'}, \bigwedge_{k''=0}^{r''} \psi''_{k''} \rangle$ where $\phi = a_i$ and $\bigwedge_{k''=0}^{r''} \psi''_{k''} = a_i$ and t_d is recorded automatically.

`3` **If** FG.Type is Conditional **then**

`4` **For each** $b_j, 1 \leq j \leq m$ **do** $r' = m$ and $\psi'_{k'} = b_j$

`5` **End If.**

`6` **If** FG.Type is Free **then**

`7` every b_j is empty and every $\psi'_{k'}$ remains unchanged.

`8` **End If.**

`9` **If** FG.Type is Compulsory **then**

`10` Create sanction $\omega_i^{sanct} = \langle \mathsf{O}_{a:auth}\phi \wedge \bigwedge_{k=1}^{r} \gamma_k, t_d, \bigwedge_{k'=0}^{r'} \psi'_{k'}, \bigwedge_{k''=0}^{r''} \psi''_{k''} \rangle$ where $\bigwedge_{k'=0}^{r'} \psi'_{k'} = \neg a_i$ and $\bigwedge_{k''=0}^{r''} \psi'_{k''} = a_i$, and the VO engineer is asked to provide $\mathsf{O}_{a:auth}\phi \wedge \bigwedge_{k=1}^{r} \gamma_j$.

`11` **End If.**

`12` **If** FG.Type is Optional **then**

`13` no sanctions are defined for FG.

`14` **End If.**

`15` Create contract $\mathbf{C}_{\mathsf{FG}} = \langle \langle r_1, \{\omega_i | 1 \leq i \leq n\} \rangle, \langle r_2, \{\omega_i^{sanct} | 1 \leq i \leq n\} \rangle \rangle$ where:

`16` **If** FG.BelongsTo is Role **then** r_1 is the name of that role and r_2 is an authority role $auth$. **End If.**

`17` **If** FG.BelongsTo is Dependency **then** r_1 is the name of the depender role and r_2 is an authority role $auth$. **End If.**

`end`

Figure 3. RdC Algorithm for Deriving a Contract from a Functional Goal.

$$\omega_1 = \langle \mathsf{O}_{a:Banker} isPaperLog(pL,d), (\cdot), received(d) \wedge approved(d), isPaperLog(pL,d) \rangle$$

$$\omega_2 = \langle \mathsf{O}_{a:Banker} isElectronicLog(pL,d), (\cdot), received(d) \wedge approved(d), isElectronicLog(pL,d) \rangle$$

$$\omega_1^{sanct} = \langle \mathsf{O}_{a:BankAuth}\phi', (\cdot), \neg isPaperLog(pL,d), isPaperLog(pL,d) \rangle$$

$$\omega_2^{sanct} = \langle \mathsf{O}_{a:BankAuth}\phi'', (\cdot), \neg isElectronicLog(pL,d), isElectronicLog(pL,d) \rangle$$

Figure 4. Norms Obtained from the Functional Goal Given in Figure 2.

notion of preference order onto the normative specification. We instead use an approach that does not involve extending the conceptualizations at the normative level. Overall, we know that a preference for a functional goal A over B means that honoring the contract on A is more desirable than honoring the contract on B. In absence of preference orders to establish relative desirability at the normative level, we must rely on activation and expiration constraints in norms. Namely, we can place activation constraints on norms of contract B so that those norms are activated (i.e., agents go about honoring the contract on B) only if the contract on A cannot be honored. Consequently, we can constrain the activation of norms in B to cases when norms in A cannot be honored. There is, however, no absolute criterion for knowing whether a contract cannot be honored. We therefore leave it to the VO engineer to choose a set of *critical* obligations in the contract on A that, once violated, mean that the contract on A cannot be honored, and that the contract on B is to be activated. In summary, if we have a preference order $A >> B >> C$, then if the contract obtained on A is not honored (i.e., critical obligations in the contract on A are violated), we activate the contract on B, and if the contract on B is not honored, we activate the contract on C. To activate the contract on B, we introduce additional activation constraints to those already defined within the norms in B: if the additional constraints hold, the contract on A is not honored.

The algorithm in Figure 5 considers each preference pair in a preference order (line `1` in Figure 5). Given a pair of functional goals, we take the more preferred functional goal FG_l, and consider individually each of the critical obligations appearing in the contract on that goal. For each critical obligation (`4`–`6`), we have the violation of the obligation's ϕ as an additional activation condition to each of the norms appearing in the contract on the less preferred goal FG_{l+1} (`5`). We also (`6`) add expiration conditions so that the contract on the less preferred goal is not activated when the critical obligations on the more preferred one are honored. By doing so, we ensure that the contract on the less preferred goal will only be considered if all critical obligations on the more preferred goal are violated. The algorithm in Figure 5 ensures that the norms on each less preferred options in the given preference order are activated only when norms on more preferred options cannot be honored. The algorithm always terminates as all of the **for each** loops move through finite sets, and always process one element at a time.

5 RELATED WORK

Frameworks and methodologies for the engineering of multi-agent systems [1, 11, 10] start from high-level goals of the system, then identify operations to achieve these goals, and design components that will perform the operations. They do not use the autonomy and adaptability inherent in agents to facilitate the engineering and development of MAS.

Preferences over Norms from Functional Goals

`Input :` One preference order (from ES and among functional goals) that has not been transformed previously using this algorithm. Assume for simplicity that the order involves w goals, and is of the form $\mathsf{FG}_1.\mathsf{UID} >> \mathsf{FG}_2.\mathsf{UID} >> \mathsf{FG}_3.\mathsf{UID} >> \ldots >> \mathsf{FG}_w.\mathsf{UID}$, and that a "preference pair" from that order is $\mathsf{FG}_l.\mathsf{UID} >> \mathsf{FG}_{l+1}.\mathsf{UID}$, for $1 \leq l \leq w-1$.

`Output:` Per preference pair, a norm on each less preferred goal in the preference pair, updated with constraints ensuring that the norms in the contract activate only if the norm on the more preferred goal cannot be honored.

`begin`

`1` **For each** preference pair $\mathsf{FG}_l.\mathsf{UID} >> \mathsf{FG}_{l+1}.\mathsf{UID}$, $1 \leq l \leq w-1$ **do**

`2` Choose a set of *critical* obligations in the contract on FG_l;

`3` **For each** critical obligation $\omega_i = \langle \mathsf{O}_{a:r}\phi \wedge \bigwedge_{k=1}^{r} \gamma_k, (\cdot), (\cdot), (\cdot) \rangle$ **do**

`4` **For each** norm $\langle (\cdot), (\cdot), \bigwedge_{k'=0}^{r'} \psi'_{k'}, \bigwedge_{k''=0}^{r''} \psi''_{k''} \rangle$ in the contract on FG_{l+1} **do**

`5` Replace $\bigwedge_{k'=0}^{r'} \psi'_{k'}$ by $\psi'_{r'+1} \wedge \bigwedge_{k'=0}^{r'} \psi'_{k'}$ where $\psi'_{r'+1} = \neg\phi$;

`6` Replace $\bigwedge_{k''=0}^{r''} \psi''_{k''}$ by $\psi''_{r''+1} \wedge \bigwedge_{k''=0}^{r''} \psi''_{k''}$ where $\psi''_{r''+1} = \phi$;

`end`

Figure 5. RdC Algorithm for Deriving Norms from a Preference Order over Functional Goals.

In our approach, the standard MAS engineering process is more efficient since it needs to describe only the desired overall system behavior and relies on the capabilities and properties of individual agents to assemble complete software systems, negotiate their roles therein, and operate according to the given normative specification. Our approach therefore relies on the premise that functionality is available in the form of individual autonomous agents, designed, developed, and maintained by, and distributed across many organizations (e.g., Google, Microsoft, etc.). The VO engineer therefore need not specify and implement individual agents, but instead determine how to regulate their interactions so that requirements are satisfied to the most desirable extent.

Our work has been influenced by the Tropos methodology [1], which features rich requirements models. Tropos does not, however, integrate preferences, priorities, and norms. Precise specification in RdC relies on a less expressive though computationally more attractive formalism. The role of a domain ontology is implicit in Tropos, while it is explicit in RdC, again due to the focus on VO. Tropos does not focus on governed MAS, but instead assumes that agents are implemented according to the requirements. RdC automates some activities that are manual in Tropos: given already designed agents, RdC governs their behavior through the conversion of requirements to norms.

6 CONCLUSIONS

We introduced Requirements-driven Contracting (RdC), which combines research into rich requirements engineering conceptualizations with research on norm-based specifications of multi-agent systems. The framework allows the specification of rich requirements models and the automatic derivation of executable normative specifications that express the obligations, permissions, and prohibitions regulating behaviors of agents participating in multi-agent systems.

Future work will address several points. First, automated resolution of conflicts between norms at our governance level is available [9], where first-order term unification is employed to find out if and how norms overlap in their influence. By integrating this work with RdC, we will be able to address inconsistencies in an automated manner at the lower, governance level. More work is necessary for combining inconsistency detection and resolution at both the ES and governance levels. Second, we rely on supervised interaction to ensure supervision of agent behavior; more elaborate role structures can be obtained by introducing additional role relationships (e.g., if an agent occupies one role, it cannot occupy some other). Third, additional normative concepts, such as that of power and loyalty (i.e., matching between individual and MAS goals), must be studied. Calculating loyalty would facilitate the allocation of resources for supervision, so that, e.g., higher initial trust may be assigned to agents with higher loyalty values.

REFERENCES

[1] Jaelson Castro, Manuel Kolp, and John Mylopoulos, 'Towards requirements-driven information systems engineering: the tropos project', *Information Systems*, **27**(6), 365–389, (2002).

[2] Andrés García-Camino, Juan-Antonio Rodríguez-Aguilar, Carles Sierra, and Wamberto Vasconcelos, 'A Rule-based Approach to Norm-Oriented Programming of Electronic Institutions', *ACM SIGecom Exchanges*, **5**(5), 33–40, (January 2006).

[3] Ivan J. Jureta, Martin Kollingbaum, Stephane Faulkner, John Mylopoulos, and Katia Sycara, 'Requirements-driven contracting for norm-governed multi-agent systems', Technical report, University of Namur, (October 2007; Also available online: http://www.jureta.net/papers/RDC.pdf).

[4] Martin J. Kollingbaum, Wamberto W. Vasconcelos, Andrés García-Camino, and Timothy J. Norman, 'Conflict Resolution in Norm-Regulated Environments via Unification and Constraints', in *DALT*, (2007).

[5] T.J. Norman, A. Preece, S. Chalmers, N.R. Jennings, M. Luck, V.D. Dang, T.D. Nguyen, V. Deora, J. Shao, W.A. Gray, and N.J. Fiddian, 'Agent-based Formation of Virtual Organisations', *Knowledge Based Systems*, **17**, 103–111, (2004).

[6] Olga Pacheco and José Carmo, 'A role based model for the normative specification of organized collective agency and agents interaction', *Autonomous Agents and Multi-Agent Systems*, **6**(2), 145–184, (2003).

[7] Katia P. Sycara, Seth Widoff, Matthias Klusch, and Jianguo Lu, 'Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace', *Autonomous Agents and Multi-Agent Systems*, **5**(2), 173–203, (2002).

[8] W. Vasconcelos, 'Norm Verification and Analysis in Electronic Institutions', in *AAMAS 2004 Workshop Declarative Agent Languages and Technologies (DALT 2004)*, (2004).

[9] Wamberto Vasconcelos, Martin J. Kollingbaum, and Timothy J. Norman, 'Resolving conflict and inconsistency in norm-regulated virtual organizations', in *Proceedings of AAMAS*, (2007).

[10] Javier Vázquez-Salceda, Virginia Dignum, and Frank Dignum, 'Organizing multiagent systems', *Autonomous Agents and Multi-Agent Systems*, **11**(3), 307–360, (2005).

[11] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge, 'Developing multiagent systems: The gaia methodology', *ACM Trans. Softw. Eng. Methodol.*, **12**(3), 317–370, (2003).

[12] Pamela Zave and Michael Jackson, 'Four dark corners of requirements engineering', *ACM Trans. Softw. Eng. Methodol.*, **6**(1), (1997).

Intelligent Contracting Agents Language

Sofia Panagiotidi [1] and **Javier Vázquez-Salceda** [1] and **Sergio Álvarez-Napagao** [1] and **Sandra Ortega-Martorell** [2] and **Steven Willmott** [1] and **Roberto Confalonieri** [1] and **Patrick Storms** [3]

Abstract. This paper presents ongoing work in the definition of a contracting language, which can be used not only to specify agreed behaviour in service-oriented architectures but also for agent-mediated systems. The contract clauses are based in deontic notions such as obligations, permissions and prohibitions. The language not only covers the contract document itself but several layers of communication, including the messages and the protocols for contract handling. An example of usage is presented.

1 Introduction

New generations of electronic business technologies promise significant advances in automation, interoperation and flexibility of business systems. Current trends are focusing in the use of Service Oriented Architectures (SOA) and Web services in particular as the technological choice for distributed business systems. Recently the community has shown interest in the creation of some form of (contractual) agreements as part of the specification of a distributed business process. But current work is either too abstract (focusing in XML representations capable to fully express *human contracts*) or too concrete (focusing in the specification of *service-level agreements* (SLAs) based in a few set of computer-observable parameters, also called *metrics* [8]). Despite its expressivity power, abstract-human-level approaches [3] cannot be used directly by a computational system to monitor and control the terms of the contract between software services at runtime, as this kind of contractual documents do not define how activities are to be monitored. In the case of concrete, service-level approaches [4, 8], monitoring is limited to the tracking of the aforementioned metrics.

We propose a step further from SOA approaches, introducing solutions coming from agent-oriented research. More concretely:

- The introduction of intentional semantics within the communication between services. This allows to properly compare between actual and intended actor behaviour within a distributed scenario.
- The creation of a *contractual language* able to express a set of intended behaviours the parties agree on by means of deontic-like clauses.
- The introduction of behavioural control mechanisms, based on the extraction of some higher-level concepts such as *commitments*, *obligations* and *violations*, which can be derived thanks to some intentional stance extracted from the communication semantics.

[1] Universitat Politècnica de Catalunya, Spain, email: {panagiotidi, jvazquez, salvarez, steve, confalonieri}@lsi.upc.edu
[2] Centro de Estudios de Ingeniera de Sistemas, Ciudad Universitaria José Antonio Echeverra, Ciudad de La Habana, Cuba, email: sandra@ceis.cujae.edu.cu
[3] Y'All, Netherlands, email: patrick@yall.nl

By combining the three elements, it becomes possible to monitor the behaviour of a set of actors by keeping track of the fulfilment of the agreements between them rather than the analysis of the exact individual messages exchanged. This paper focuses mainly in the description of the contractual language, although some parts of the service communication and behaviour monitoring will be also described. The content is organized as follows. In the next section we present an example introducing the domain problem while in section 3 we describe the conceptual layers of our contracting language. Section 4 analyses the important elements of the contract representation model. In the following section, it is explained how messaging and communication is achieved between the contracting parties. In section 6 we illustrate how the whole framework operates within a realtime environment. We end this paper identifying some previous work and discussing our conclusions on the presented work.

2 Example: Car Insurance Brokerage

In this paper we will use as example the agreements between several parties in a car insurance scenario. In this scenario there are three types of parties: *insurance companies*, *repair companies*, and an extra organization, *Damage Secure*, which acts as a broker between the former, facilitating all businesses involved in dealing with car damage claims for a number of insurance companies. The goal of Damage Secure is to enhance the quality and efficiency of the total damage claims handling process between consumers, damage repair companies and insurance companies.

The example procedure is as follows: Given a claim of a client, the insurance company delegates the task of finding the best repair company to Damage Secure. The insurance company will state certain requirements for the repair job, e.g., the vicinity of the repair company, the preferred method of repair (e.g. original parts or not), price-range, repair time. Damage Secure will select the best available repair company for the job. After the repair company receives the car and consents to the repair, the two will commit to a short term contract which specifies the details of the repairing procedure, including the invoice etc. Then the repair company repairs the car and notifies Damage Secure when is it complete. For every repair job, a surveyors report has to be provided. Damage Secure will forward the invoice to the insurance company. Provided there is no dispute over the quality of the repair (in which case an expert is called to perform a quality assessment), the insurance company pays the repair company the agreed price in the contract.

The focus of interest of this example is to show how the agreements between insurance companies and repair companies are represented and to explain how this is useful to then check at runtime whether those agreements (explicitly expressed as clauses in the contract) are met when the actual repair takes place.

3 Layers/Elements of the Contracting Language

The contracting language separates concerns between different layers of communication:

1. The *Domain Ontology Layer* contains the domain ontology, providing ontological definitions of terms, predicates and actions (e.g. *car*, *workshop*, *repair*), to ensure that the same definitions are used by all parties and to avoid terminological misunderstanding.
2. The *Contract Layer* defines deontic statements about the parties' obligations, permissions and prohibitions in terms of predicates and actions defined in the previous layer (e.g. the workshop is *obliged to* repair the car in 2 days). The definition of such deontic statements is expressed as *clauses* in the contract.
3. The *Message Content Layer* defines the message content, allowing agents to express statements about contracts (e.g. a contract being *active/inactive*, *fulfilled*, *complete/incomplete*), actions related to contracts (e.g *accept*, *sign*, *cancel* a contract) or expressions about actions and events related to the clauses in a contract.
4. The *Message Layer* allows agents to express their attitudes about the content of the message (e.g. an agent *proposes* to sign contract C1, or an agent *requests* cancellation of a contract C2). Taking Speech Act Theory as a basis, these attitudes are expressed by means of a standard set of pre-defined *performatives* which (similarly to FIPA ACL) are included as part of the message envelope.
5. The *Interaction Protocol Layer*, which pre-defines contract handling protocols as sequences of messages. These protocols structure interaction by defining sets of acceptable sequences of messages which would fulfil the goal state of the protocol (e.g. a protocol for agreeing on contract termination).
6. The *Context Layer* describes the interaction context where contractual parties will carry out the obligations, permissions and prohibitions agreed in the contract.

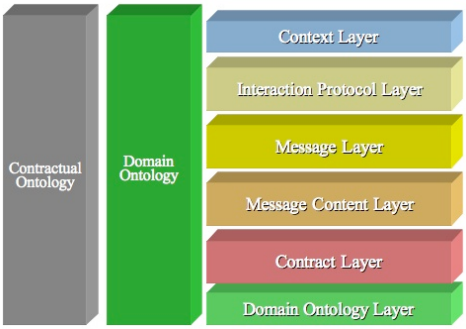


Figure 1. General view of the Contracting Fremework

Apart from these horizontal layers, there is a vertical dimension that appears at all layers: the ontological dimension. This dimension defines all terms that are needed for each layer. In this approach there are basically two types of ontologies:

- Domain ontologies: these define the terms, actions, predicates and relationships needed for communication in a given domain. For instance, in the case of a car insurance application, there will be one or several ontologies defining terms such as *car*, *insurance*, *damage*, actions such as *repair* and predicates such as *repaired*. The domain ontologies are defined in the domain ontology layer, but are also used in other layers such as the contract layer, the message layer or the context layer.
- The *Contractual Ontology*: an ontology that predefines all the terms, predicates and actions that are used by the framework, independently of the application domain. This ontology defines terms such as *contract*, *party*, *obligation*, *action, commitment* or *violation*, actions such as *creating* a contract, *committing* to a contract and predicates such as *fulfilled* or *cancelled*.

More detailed information about the Contracting Language can be found in [1]. We focus only in the main aspects in the next sections.

4 Contract representation

The Contract Data Model is an XML based representation suitable to represent both fully defined contracts or contract templates(contracts partially defined). It is based in the theoretical framework defined in [9]. It is composed by a name, starting and ending dates and three main parts: contextualisation, definitions and clauses. Figure 2 depicts this structure[4].

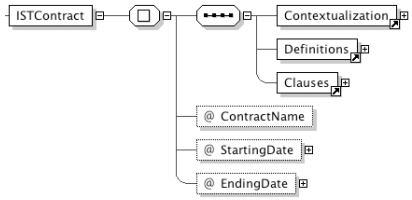


Figure 2. Root elements of the Contract representation

The name of a contract has to be unique inside the context it is being declared in (a contract is uniquely identified by a combination of the context namespace and the contract name). The starting and ending dates of the contract express the valid time period of the contract. Ending date is optional (contracts without end date are allowed by leaving the ending date blank). The rest of the elements are described in the following sections.

4.1 Definitions

The definitions part defines the parties of the contract, the roles they play, some optional grouping of roles and the model of the domain.

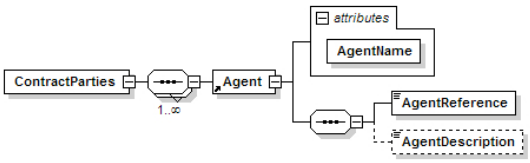


Figure 3. Parties element in contract

[4] The detailed semantics of the figures can be found in the Oxygen XML User Manual, at http://www.oxygenxml.com/doc/oXygenUserGuide-standalone.pdf, pages 91-92

4.1.1 Parties

The contract parties are the list of agents involved in the contract, that is, the set of agents assigned to fulfil one or more clauses of the contract. In the contracting language, the contract parties element has for each agent its name, a reference to this agent and optionally a text description about this agent (see Figure 3).

For instance the following is an extract of the parties definition in the car insurance scenario.

```
<ContractParties>
<Agent AgentName="Feel Safe">
  < AgentReference> feelsafe.com:8080/FS</AgentReference>
  <AgentDescription>Car Insurances</AgentDescription>
 </Agent>
 <Agent AgentName="Damage Secure">
  < AgentReference> damsec.org:8080/DS</AgentReference>
  <AgentDescription>Service Broker</AgentDescription>
 </Agent>
 <Agent AgentName="Fast Fix">
  < AgentReference> fastfix.com:8080/BR</AgentReference>
  <AgentDescription>Bob's Garage</AgentDescription>
 </Agent>
</ContractParties>
```

4.1.2 Role Enactment List

The role enactment list element is used to assign roles to the agents (parties). The definition of roles is not common in existing contracting languages (such as WS-Agreement or WSLA), which tend to use directly some kind of agent identifier to assign responsibilities. In our approach roles are a very powerful mechanism that decouples the definition of responsibilities from the specific agents that will have to fulfill them. This decouplement allows to create *contract templates* which fully specify, e.g., the obligations of a repair company or a insurance company in an archetypal repair scenario without specifying the exact agents enacting the roles. Such contract template can be then instantiated several times by only specifying each time the exact parties and the role-enactment relations.

Figure 4. Role enactment list element in contract

Figure 4 shows the structure of a Role-enactment list. In the car insurance scenario, the assigned roles could be described as follows.

```
<RoleEnactmentList>
<RoleEnactmentElement
    AgentName="Feel Safe" RoleName="Insurance Company"/>
  <RoleEnactmentElement
    AgentName="Damage Sescure" RoleName="Broker"/>
  <RoleEnactmentElement
    AgentName="Fast Fix"   RoleName="Repair Company"  />
</RoleEnactmentList>
```

4.1.3 Group List

The group list element is used to group agents that have different roles into a group that might share some responsibilities. It can be useful when there are clauses that should affect a subgroup of the agents in the contract. Each group of the list has a name and the list of agents that compose the group (see Figure 5).

There is always a predefined group called *ALL*, which groups all agents in a contract. Also all roles implicitly create a group of all agents that enact the same role. An example in the car repair scenario

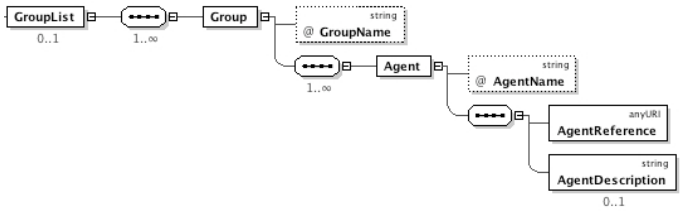


Figure 5. Representation of the Group list element

would be the following: lets suppose that there are some specific regulations applying only to small-medium enterprises (SMEs), and that only two of the parties are SMEs. This could be expressed as follows:

```
<GroupList>
  <Group GroupName="Small-Medium Enterprises">
    <AgentName>Damage Secure</AgentName>
    <AgentName>Fast Fix</AgentName>
  </Group>
</GroupList>
```

4.1.4 World Model

Within the contracting language, it must be ensured that all parties have a shared understanding of the elements in the world that they will refer to in their interactions and also of the characteristics of the world itself. A representation of the different elements composing the knowledge about the domain is depicted in Figure 6.

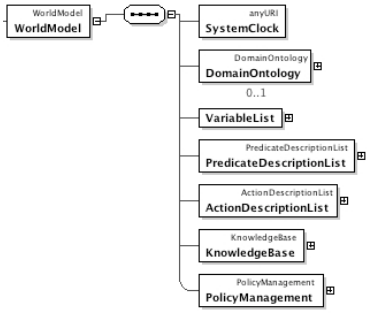


Figure 6. Representation of the world model element

4.2 Contextualisation

In the contracting language, contexts are implicitly created when a contract becomes active (as we explain in section 6). They will obtain elements from the contract, such as the world model, the domain ontology, the descriptions of possible actions and processes within the domain and the set of regulations to be applied within the organization being represented. Moreover, the context of a contract, called *interaction context*, may be as well contained inside another interaction context and therefore inherit all its knowledge representation elements and be constrained by its regulations. In this case, the Contextualisation part of the contract has to specify which is the *parent contract*. If a contract is an instance of a *contract template*, this is also specified in this part.

4.3 Clauses

Clauses express agreements between parties in the form of deontic statements. In order to express the clauses we have adopted a variation of the norm representation defined in [2].

A clause (Figure 7) is structured in two parts: the conditions and the deontic statement. There are three conditions, which have the form of boolean expressions, that have to be evaluated at different stages of the clause life cycle.

- Activating (or triggering) Condition: when this condition holds true, the clause is considered to be activated. This condition can also be referred to as precondition. If the boolean expression of this condition includes a violated() predicate, the clause is considered to be a violation handler.
- Exploration (or maintenance) Condition: this is the invariant of the deontic statement execution, which means that when the clause is active and the statement is being enforced, the exploration condition has to hold always true. If this does not happen, a violated() predicate will be raised. This condition can include temporal operators *before*() and *after*() with allow to express temporal constraints and deadlines.
- End (or achievement) Condition: the clause is considered to be inactive and successfully fulfilled if and only if the exploration condition has always held true and the end condition holds true.

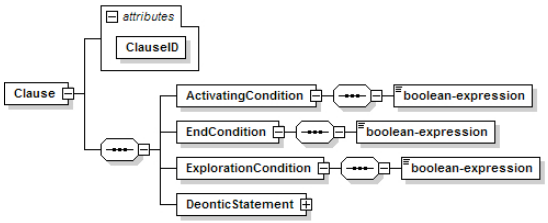


Figure 7. Representation of the clause element

The DeonticStatement is the central element of the clause. There are three main fields in its structure: the deontic modality, the roles involved and the object of the norm (Figure 8):

- Modality: this field indicates the deontic modality of the statement, which can be either Obligation, Prohibition, or Permission.
- Who: contains the set of roles and groups of roles that will have to fulfil the statement.
- What: this represents the object of the norm. This object can be an action or a state. If it is an action, this action will have to be executed in order to fulfil the norm. Otherwise, if it is a state, the responsible actor(s), defined in the Who attribute, will have to ensure that this state is accomplished as long as the exploration condition holds true.

Clauses can be used in two ways:

- as *standard clauses*: they define what ought/ought not to be done. For instance, a clause stating that a buyer should pay for a given item within some time period.
- as *violation handling clauses*: they define what to do if standard clauses are violated) (i.e. The exploration condition does not hold).

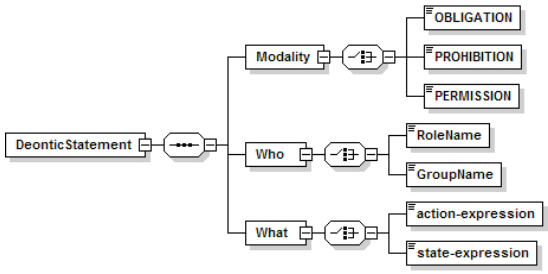


Figure 8. Representation of the deontic statement element

In the car insurance scenario, an example of a standard clause would be "The repair company is obliged to notify Damage Secure before april 10 if the report of the repair is ready"'. Such clause can be expressed in our language as follows:

```
<Clause ClauseID="NotifyRepairCompleted">
  <ActivatingCondition>
    <BooleanExpression>
      exists(RepairReport, isRepaired(car12f3pw, R1))
   </BooleanExpression>
  </ActivatingCondition>
  <EndCondition>
    <BooleanExpression>
      isSentRepairCompleted(car12f3pw,
                    RepairReport, "Damage Secure", R1)
</BooleanExpression>
  </EndCondition>
  <ExplorationCondition>
    <BooleanExpression>
       Before(2008-04-10T15:30:30+01:00)
    </BooleanExpression>
  </ExplorationCondition>
  <DeonticStatement>
    <Modality>
      <OBLIGATION/>
    </Modality>
    <Who>
      <One id="R1" enacting="Repair Company"/>
    </Who>
    <What>
      <ActionExpression>
        sendRepairCompleted(car12f3pw,
                    RepairReport, "Damage Secure", R1)
      </ActionExpression>
    </What>
  </DeonticStatement>
</Clause>
```

This clause only activates when a report stating that the car is repaired exists, and deactivates once the report is sent. The *ExplorationCondition* specifies in this case that the obligation should be met before a given deadline. The *DeonticStatement* specifies that this is an obligation related to one agent, R1[5], enacting the *Repair Company* role, and it consists of one action (sending the repair report).

4.4 Contractual Ontology (IST-Contract)

As mentioned in Section 3, the Contractual Ontology is the ontology that predefines all the terms, predicates and actions that are used by the framework, independently of the application domain. The purpose of this is for a generic, non-application dependent information to exist within the framework and be shared amongst the agents.

This ontology defines objects that represent primal entities existing within the framework, actions committed by an actor and predicates which declare states and conditions of the system. Some of the primal concepts that exist in the contractual ontology are:

[5] It is important to note here that we show a version of the clause that is valid for both contract templates and contracts, where R1 is a variable, the value of which is set by unification in the activating condition (i.e. the Repair Company that created the Repair Report). If one is not interested in reusing this clause in several contracts, R1 could be directly substituted here by the specific Repair Company.

- Contract An agreement between several parties
- Obligation An obligation corresponding to an agent
- Party A person, agent or entity
- Action An action taken by one of the parties
- Predicate A logical predicate with zero or more arguments which is true or false
- List The classical notion of list
- Penalty The penalty for a party when violating of an obligation

The table below shows two examples of pre-defined actions and predicates to express statements about contracts are shown. The first is the action of committing to a contract and the second is the action of creating a contract. Attributes represent both inputs and outputs. Wherever a predicate appears in the precondition and does not in the postcondition, it is assumed that it remains the same after the action is completed.

action	*attributes*	*precond.*	*postcond.*
commit	P: Instance of Party C: Instance of Contract	C is valid	committed(P, C) initiated(C) $\forall$ O Instance of Obligation in C: obliged(P, C, O) happened(P, commit(P, C))
create	P: Instance of Party C: Instance of Contract	$\neg$(exists(C))	exists(C) happened(P, create(P, C))

Other actions, defined in a similar way are: terminate, withdraw, cancel, get, update and more. Predicates are of the form of:

- committed(P, C), where P is instance of a party and C is instance of a Contract

Other predicates, defined in a similar way are: happened, allcommitted, violated, obliged, initiated, active, cancelled, and more.

5 Contracting Messages and Protocols

In order to increase the expressivity of the communication between services to introduce some intentional stance, the contracting language also defines a set of performatives to be used by the parties. We have extended FIPA ACL [6] performative set in a way that can be used not only by Web services but also by agents. To properly use those performatives, our contracting language also specifies the message structure, a content language and a set of contracting protocols

5.1 Message Structure

The message structure is a XML variation of FIPA's message structure [6], where the message body contains the usual FIPA proposed attributes, i.e. sender, receiver, performative, language, content, etc.

5.2 Message Content

The content of the communication message describes what the purpose of the communication is and expresses knowledge existing in the level of the world representation.

One important feature of the contracting language is that not only full contracts can appear as content of the message but statements about contracts and contractual actions too (see 4.4)

The language used to express the content of the messages between the actors is based on a subset of FIPA-SL [5] namely FIPA-SL2, adapted to an RDF representation.[6] The reason for this is that it remains an adequately expressive set as it allows first order predicate, modal logic operators, quantifiers (*forall*, *exists*) and reference operators (*iota*, *any*, *all*), which are needed in order to give the expressivity needed for flexible contract-related communication.

5.3 Performatives

The full set of FIPA-ACL [6] performatives is adopted (e.g. *query*, *inform*, etc.). However, FIPA standards do not include cases in which an agent might propose an action to be performed by more than one agent (e.g. to propose that many agents commit on a contract). For this reason, FIPA-ACL alone is not sufficient, as its speech acts cannot be used to form, maintain and dissolve joint intention (mainly to commit) in order to support advanced social activity (i.e., teamwork).

We have extended FIPA-ACL performatives with extra performatives based in joint intention theory:

- **suggest**: The action of submitting a suggestion for the sender and the receiver to perform a certain action.
  <i, suggest (j, <i, act>, <j, act>)>
  where i is the sender and j is the receiver.
- **consent-suggestion**: The action of showing consent to a suggestion for the sender and the receiver to perform a certain action.
  <i, consent-suggestion (j, <i, act>, <j, act>)>
  Agent i informs j that, it consents for agent i and agent j to perform action act giving the conditions on the agreement.
- **dismiss-suggestion**: The action of dismissing a suggestion for the sender and the receiver to perform a certain action.
  <i, dismiss-suggestion (j, <i, act>, <j, act>, ψ)>
  Agent i informs j that, because of proposition ψ, i does not have the intention for i and j to perform action act.

5.4 Protocols

The architecture defined in [9] identifies agent behaviours which should be well defined and designed, in order for the contract lifecycle to be smoothly executed. These include the phases of Contract Creation, Contract Fulfilment, Contract Modification and Update, Contract Violation, Contract Cancelling By Agreement and more. Such a communication can be achieved through communication protocols which define significant part of every agent's behaviour.

A set of contract handling protocols has been created to support those behaviours. Figure 9 depicts an example of a protocol expressing the creation of a contract between two agents.

Protocol handlers have been developed for both agents and agentified Web services, easing the implementation of the communication to the designer.

6 Runtime Contract Execution

Once a contract is created, the contract handling mechanisms create an interaction context following the specification of the contract. Such context acts as an Electronic Institution providing a safe environment for contractual interactions. Context includes facilitator agents such as Notaries, Ontology Service, Contract Storers and Managers. Notaries are trusted third parties that can be used as testimonies of the execution of a contract. The Ontology service can be queried by the agents: it receives a reference to an ontology definition

[6] RDF has been chosen here instead of XML because it is easier to integrate with semantic representations such as OWL.

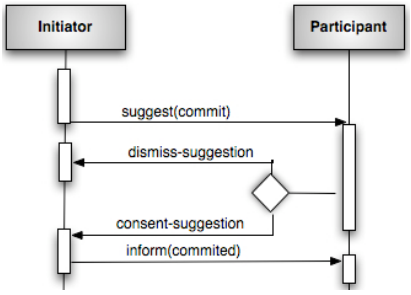


Figure 9. Simple Contract Create protocol

and returns the definition itself. Contract stores are trusted repositories for contracts, which keep a version of any contract agreed and any update made to it. Contract Managers are the ones that monitor contract execution, by keeping an updated list of all clauses in the contract and their status (active, inactive, violated, etc.). The status for each clause is handled by tracking the Activating, Exploratory and End conditions. Contractual obligations are converted into *critical states* (see [9]) which are checked for and acted upon, and which can also be used for verification. Violations are detected by the Activation conditions in the violation handling clauses and in such case the manager performs corrective actions or enforces compensations for agreements violated by parties.

7 Previous Work

As explained in section 1, existing contract representations in service-oriented Architectures are either too abstract, human-oriented (such as OASIS ebXML [3] or too low-level, based on Service Level Agreements (SLAs). We will concentrate in the latter ones, as these are the only ones usable for monitoring computational behaviours. WS-Agreement [4] was one of the first XML-based languages for agreements (terms and meta information about the agreement) which also included a definition of a protocol for establishing these agreements. Although WS-Agreement is widely used, it is not possible to describe multi-party contractual relationships, only one-to-one contracts. It also lacks the definition of metrics in order to support flexible monitoring implementations. Most of this has been solved in the Web Service Level Agreement (WSLA) [8] framework, a more expressive language targeted at defining and monitoring agreements for Web services, covering the definition of the involved parties, the service guarantees and the service definition. WSLA allows the specification of third parties and also monitoring service performance by means of metrics. However, it lacks the description of the application execution context and mechanisms for the execution of activities such as negotiating the contract. Both WS-Agreement and WSLA also lack formal semantics about behaviours and agreements (making it difficult to reason about them or to verify certain properties). Finally, Rule Based Service Level Agreements (RBSLA) [11, 10] focuses on sophisticated knowledge representation concepts for service level management (SLM) of IT services. RBSLA is an extension of RuleML. The rules are based on the logic components of Derivation, Event Condition, Event Calculus, Courteous Logic and Description Logic. RBSLA's flexibility is based on its declarative nature, allowing a more compact and intuitive representation. Monitorisation of agreements is based in Event-condition-action rules with proper operational semantics. However no notion of responsibilities or obligations are defined. The language we propose is thus far more expressive than RBSLA, allowing to represent the parties responsibilities.

More formal approaches in contracts include the work by Sergot in [12], where the role of deontic logic in legal knowledge representation is addressed, or the work by Weigang et al. in [7], where an agent societies model and techniques that achieve the described objectives, landmarks and contracts, are described.

Up to our knowledge, none of the existing languages covers all the layers of communication included in our approach, from the domain and contract layers to the interaction protocol and context.

8 Conclusions

This paper presents ongoing work in the definition of a contracting language which can be used in both service-oriented architectures and in agent-mediated systems. This covers all the levels of communication and bases its expressivity in the introduction of speech acts in the parties communication and the use of deontic notions such as obligations, permissions and prohibitions in the contract clauses.

The semantics of the language are based in the formalisation in [2]: deontic expressions (in clauses) are reduced into LTL expressions, and critical states are treated as landmark patterns. Ongoing work is to extend the formalisation to also map the intentional statements in the message layer (such as joint commitments) into LTL.

ACKNOWLEDGEMENTS

This work has been funded mainly by the FP6-034418 CONTRACT project. Javier Vázquez-Salceda's work has been also partially funded by the Ramón y Cajal program of the Spanish Ministry of Education and Science. All the authors would like to thank the CONTRACT project partners for their inputs to this work.

REFERENCES

[1] *IST Contract project WP3 technical report*, 2007. Available at http://www.ist-contract.org.
[2] Huib Aldewereld, *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols*, PhD thesis, Utrecht University, 2007.
[3] OASIS ebXML Joint Committee, 2008. http://www.ebxml.org.
[4] A. Andrieux et al, *Web Services Agreement (WS-Agreement) Specification*, World-Wide-Web Consortium (W3C), 2005. http://www.ogf.org/documents/GFD.107.pdf.
[5] Foundation for Intelligent Physical Agents, *FIPA SL Content Language Specification*.
[6] Foundation for Intelligent Physical Agents, *FIPA ACL Message Structure Specification*, 2000.
[7] Weigand. H., V. Dignum, J-J Meyer, and F. Dignum, *Specification by Refinement and Agreement: Designing Agent Interaction Using Landmarks and Contracts*, Engineering Societies in the Agents World III, LNAI 2577, Springer Verlag, pp.257-269, Berlin, 2004.
[8] H. Ludwig, *Web Service Level Agreement (WSLA) Language Specification*, IBM, 2003. http://www.research.ibm.com/wsla.
[9] S. Miles, N. Oren, M. Luck, S. Modgil, N. Faci, C. Holt, and G. Vickers, *Modelling and Administration of Contract-Based Systems*, Symposium on Behaviour Regulation in MAS, Aberdeen, Scotland, 2008.
[10] A. Paschke, *RBSLA: A declarative Rule-based Service Level Agreement Language based in RuleML*, Int. Conf. on Computational Intelligence for Modelling, Control and Automation and Int. Conf. on Intelligent Agents, Web Technologies and Internet Commerce Vol-2, 2006.
[11] A. Paschke, J. Dietrich, and K. Kuhla, *A logic based SLA Management Framework*, In Semantic Web and Policy WS (SWPW) at 4th Semantic Web Conf., Galway, Ireland, 2005.
[12] M.J. Sergot, *The Representation of Law in Computer Programs*, Bench-Capon (ed.) Knowledge-Based Systems and Legal Applications, Academic Press, 1991.

Argumentation for Normative Reasoning

Nir Oren and **Michael Luck**[1] and **Timothy J. Norman**[2]

Abstract. An agent's behaviour is governed by multiple factors, including its beliefs/desires/intentions, its reasoning processes and societal influences acting upon it, such as norms. In this paper we propose an extensible argumentation inspired reasoning procedure, and show how it may be used to perform normative reasoning. . The language used by our procedure is built around defeasible, non-monotonic rules called argument schemes. The evaluation of the interactions between argument schemes and predicates is performed using a novel argumentation based technique. We show how issues such as normative conflict, priorities over norms, and the effects of norms may be represented using the framework, and how the agent may use these to reason effectively.

1 Introduction

An increasingly popular way of declaratively controlling agent behaviour is through the use of norms. Most commonly, a set of obligations and prohibitions are imposed upon an agent, constraining its behaviour accordingly. Different approaches define these norms in different ways. In the simplest case, an obligation can be seen as a hard constraint, with the system entering an undefined state if the norm is violated. More flexible systems treat norms as soft constraints, but as flexibility increases, difficult questions arise in areas including normative reasoning, verification, and semantics, as well as how norms interact with each other.

In particular, additional complications arise when normative conflicts occur, with an agent having to select which norms to honour, and which to ignore. Ultimately, the agent's actions are governed by its preferences, knowledge (including beliefs, desires and intentions), the norms affecting it, and the state of the environment. Several efforts to address this issue have been proposed, which take these various factors into account in different ways.

One strategy espoused by some (including the philosopher John Pollock [8]) follows the approach that humans appear to use when faced with multiple choices; namely to engage in an *internal dialogue* and act based on its outcome. For example, if I am to decide whether to play a game or write a paper, I would weigh up the pros and cons of each of these actions, in the context of my obligations, e.g. when the paper is due, and how much I like my job. This weighting process may create additional reasons to pick one action over another, and may cause other reasons to no longer be applicable.

Pollock's work was intended to apply to any form of practical reasoning, and did not pay particular attention to dealing with norms. His internal dialogue based representation of practical reasoning, instantiated via an argumentation procedure, is highly appropriate for reasoning about norms because norms typically constrain desired behaviour, leading to the need to resolve (internal) conflicts so as to allow an agent to determine whether to comply with its norms. Our reasoning framework, while simpler than Pollock's, contains the features necessary to reason about normative issues. It is based on the emerging AIF standard [3], and provides support for normative concepts via the idea of argument schemes. Argument schemes represent defeasible, possibly non-deductive, rules of inference, and are intended to capture common patterns of argument. They may be general, or domain-specific.

In this paper, therefore, we use argument schemes to represent reasoning rules. We present a number of argument schemes that can be used to reason about normative concepts. By representing its knowledge using these argument schemes, and using results from argumentation theory, an agent is able to infer, from the interactions between argument schemes, how to act on the basis of its norms, and whether any of its norms should be ignored. Our approach is able to naturally deal with normative conflict and, due to its non-monotonic nature, is easily able to handle cases where an agent is presented with additional information. Apart from the formal introduction of normative argument schemes, our main contribution thus revolves around the framework's ability to aid an agent in resolving normative conflict. Thus, we begin the next section by introducing argument schemes, after which we show how an agent may reason about their interactions. After providing an example showing the framework in action, we conclude the paper by examining related research and proposing further extensions to the work presented here.

2 Argument Schemes and Information

As mentioned in the introduction, agents using our framework reason about the world using inference rules called argument schemes. We begin this section by informally describing these argument schemes. Later, in Section 4, we show how these argument schemes may be used for normative reasoning.

An argument scheme represents a (possibly non-deductive) rule of argument. Argument schemes consist of three components: a set of premises, a set of conclusions, and a set of other argument schemes which may be *undercut* by this scheme (an undercut represents a reason for not being allowed to use the scheme, and is described in more detail later). Premises and conclusions refer to concrete elements of the environment, form the basis of our language, and are represented using Prolog-like predicates. As per Prolog convention, we assume that the first letter of a variable is capitalised, while a constant's first letter is lowercase. Unbound predicates are predicates containing variables; when we refer to predicates, we mean predicates containing no unbound variables. We call the set of all unbound predicates UPS, while the set of predicates is labelled PS.

We define a function $subst(A, B)$, which returns predicate A unified according to the mapping B, where $A \in UPS$, and B is a mapping between variables and unbound predicates. Predicates are used

[1] King's College London, UK, email: nir.oren,michael.luck@kcl.ac.uk
[2] University of Aberdeen, Scotland, email:t.j.norman@abdn.ac.uk

as the basis for knowledge representation within an agent's knowledge base, and are used to represent both contested and uncontested facts. These facts are used as inputs to argument schemes, and may also form as a result of the application of such a scheme. AIF refers to such predicates as *information*, and we will adopt this terminology.

As stated previously, an argument scheme represents a rule of inference (and usually refers to a repeatedly used form of argument). Argument schemes operate on specific types of information, and thus make use of unbound predicates; they are applicable when specific conditions hold, namely when all their premises are present. The application of an argument scheme results in some conclusions being drawn. Argument schemes may also influence the application of other schemes. Pollock gives the following example of an argument scheme undercutting another argument scheme (i.e. preventing the other scheme coming into force). Here, the application of the second argument scheme nullifies (undercuts) the first scheme's application.

1. If an object looks a certain colour, it is that colour.
2. If a light of a certain colour shines on an object, that object will take on the light's colour (even if the object is not that colour).

Strong parallels exist between argument schemes and default reasoning, particularly when dealing with issues such as burden of proof and critical questions [10]. Premises to an argument scheme may thus be required, or be default (i.e. only the explicit presence of the negation of a default may cause a scheme to not be applied). Since our language does not explicitly cater for negation, two types of conclusions may exist for a given argument scheme, namely those conclusions supported by the scheme, and those attacked by the scheme. Formally then, we may define an argument scheme as follows:

Definition 1 ***(Argument Scheme)*** *An argument scheme is a structure of the form*

$$AS = \langle ASDefaults, ASPremises, ASSupports, ASAttacks, ASUndercuts \rangle$$

where $ASDefaults, ASPremises, ASSupports, ASAttacks \in 2^{UPS}$. *Furthermore, a predicate may only appear in one of* $ASDefaults$, $ASPremises$, $ASSupports$ *and* $ASAttacks$. *If a variable* v *appears in an unbound predicate within* $ASSupports$ *or* $ASAttacks$, *it must also appear within* $ASDefaults$ *or* $ASPremises$. $ASUndercuts$ *is a set of undercutting bindings.*

$ASDefaults$ represents the set of defaults which must not hold for an argument scheme to be applied, while $ASPremises$ represents those pieces of information that must hold. $ASSupports$ consists of those conclusions supported by the argument scheme, and $ASAttacks$ stores the conclusions that are attacked by the scheme.

Definition 2 ***(Undercutting bindings)*** *An undercutting binding is a pair* $\langle AS, UBMapping \rangle$ *where* AS *is an argument scheme, and* $UBMapping$ *is a mapping from the variables found in* $(ASPremises \cup ASSupports \cup ASAttacks)$ *to* UPS.

As an example, consider the argument scheme "if A implies C, and B implies C, then, by accrual, C is to be strongly believed". Formally, this argument scheme could be represented as follows:

$$\langle \{\}, \{A, B, implies(A, C), implies(B, C)\}, \{C, strongly(C)\}, \{\}, \{\} \rangle$$

Such an argument scheme implicitly assumes that the implication may accrue. Another argument scheme, together with its associated undercutting bindings may be defined to prevent this scheme from being applied in situations where accrual may not occur[3]:

$$\langle \{\}, \{notAccrue(X, Y)\}, \{\}, \{\}, \{\langle \langle \{A, B, implies(A, C), implies(B, C)\}, \{C, strongly(C)\}, \{\}, \{\} \rangle, \{\{A, X\}, \{B, Y\}, \{C, C\}\}\}\rangle$$

Note that there is no specialised representation for negation in our framework. The following schemes are thus present in most systems:

$$< \{\}, \{A\}, \{\}, \{not(A)\}, \{\} >$$
$$< \{\}, \{not(A)\}, \{\}, \{A\}, \{\} >$$

An argument scheme may be used, together with some information, to generate new information. The application of an argument scheme in this way results in an instantiated argument scheme, and is achieved via a process of unification.

Definition 3 ***(Instantiated Argument Scheme)*** *An instantiated argument scheme is a tuple* $IAS =< AS, Mapping >$ *where* AS *is an argument scheme, and* $Mapping$ *is a mapping such that,*

$$\forall U \in ASPremises, subst(U, Mapping) \in KB$$
$$\forall U \in ASDefaults, subst(U, Mapping) \notin KB$$
$$\forall U \in ASSupports \cup ASAttacks, subst(U, Mapping) \in PS$$

3 Evaluating Arguments

The process of argument not only generates arguments, but also links information in a specific way, with some arguments supporting others, and other arguments and pieces of information attacking each other in various ways. Given a set of arguments, an agent must be able to determine which of these are, in some sense, admissible. That is, which arguments may be deemed to "hold" given the interactions within the set. The most influential work in this area is probably that of Dung [4]. Here, arguments are treated as abstract entities, with no attention paid to their content. The only attribute associated with arguments is that they require the ability to attack other arguments. Given a set of arguments, and a binary attack relation, Dung neatly classifies the sets of arguments that a rational reasoner deems to be admissible at the end of an argument.

Various extensions to Dung's work have been proposed (e.g. [2, 1]), and in this paper, we make use of Oren's work on abstract evidential reasoning [6], which we now describe. An evidential argument system stores the arguments and records the way they may interact with each other. Two types of interactions may exist: attacks, and supports. Oren's work investigated which sets of arguments may be considered "admissible"; that is, which sets make sense to some sort of rational reasoner, given the interactions between arguments. In his work, an argument is only considered admissible if there is an unbroken chain of support from some sort of universally accepted "evidence argument" to the argument. A chain may be broken if it is successfully attacked by some other supported argument.

Definition 4 ***(Evidential Argument System)*** *An evidential argument system is a tuple* (A, R_a, R_e) *where* A *is a set of arguments,* R_a *a relation of the form* $(2^A \backslash \{\eta\}) \times A$ *and* R_e *a relation of type* $2^A \times A$.

Additionally, $\nexists z \in 2^A, y \in A$ *such that* $zR_a y$ *and* $zR_e y$.

[3] It is also possible to represent this by introducing "virtual" information and using the $ASDefault$ property.

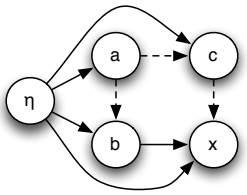


Figure 1. An evidential argument system. Solid arrows represent support, while dashed arrows represent attacks between arguments.

The R_e and R_a relations respectively encode evidential support and attacks from a set of arguments, to an argument[4]. η represents a special argument, representing unquestionable support from the environment, and can also be used to represent defaults. As an example, consider the following arguments:

- a It was dark. When it is dark, a witness's statement could be wrong.
- b Witness b made the statement that the bird flew. A witness' statement can be viewed as evidence.
- c Witness c made the statement that the bird did not fly. A witness' statement can be viewed as evidence.
- x We know that birds can normally fly, and thus given some evidence, we may claim that the bird flew.

This would result in the following evidential argument system, as illustrated graphically in Figure 1.

$$< \{a, b, c, x\},$$
$$\{(\{a\}, c), (\{a\}, b), (\{c\}, x)\},$$
$$\{(\{\eta\}, a), (\{\eta\}, b), (\{\eta\}, c), (\{\eta\}, x), (\{b\}, x)\} >$$

Within an evidential argument framework, it is necessary for an argument to be supported by some other argument for it to ultimately be admissible. This immediately raises bootstrapping concerns as some initial argument has to be supported for any other argument to eventually be supported. We overcome this problem with the special η argument:

Definition 5 ***(Evidential Support)*** *An argument a is supported by a set S iff*

1. *$SR_e a$ where $S = \{\eta\}$ or*
2. *$\exists S' \subset S$ such that $S' R_e a$ and $\forall x \in S'$, x is supported by $S \backslash \{x\}$*

S is a minimum support for a if there is no $S' \subset S$ such that a is supported by S'.

Assume, in the example above, that no direct link exists between η and x. Then x is supported by the set $\{\eta, b\}$. Given the notion of evidential support, we may define an *evidence-supported* attack:

Definition 6 ***(Evidence-Supported Attack)*** *A set S carries out an evidence-supported attack on an argument a if*

- *$X R_a a$ where $X \subseteq S$, and,*
- *All elements $x \in X$ are supported by S.*

An evidence-supported attack by a set S is minimal iff there is no $S' \subset S$ such that S' carries out an evidence-supported attack on a.

[4] Attacks and supports from sets of arguments allow us to represent conjunctive arguments [5].

We will usually write *s-attack* when referring to an evidence-supported attack. A s-attack represents an attack backed up by evidence or facts. Within the example, Arguments a and c are both supported by η; a thus s-attacks b, and c s-attacks x.

Support for an argument is clearly one requirement for acceptability, but it is not enough. Following Dung, an argument should be acceptable (with respect to some set) if it is defended from attack by that set. The question arises as to whether all attacks should be defended against, or only s-attacks. Since the framework focuses on s-attacks, we choose the latter option, allowing an argument to be defended from attack by either having the attack itself attacked, or by having any means of support for the argument attacked by the defending set.

Definition 7 ***(Acceptability)*** *An argument a is acceptable with respect to a set S iff*

1. *S is a support for a.*
2. *Given a minimal s-attack $X \subseteq 2^A$ against a, $\exists Y \subseteq S$ such that $Y R_a x$ where $x \in X$ so that $X \backslash \{x\}$ is no longer a s-attack on a.*

An argument is thus acceptable with respect to a set of arguments S if any argument that s-attacks it is itself attacked (either directly, or by being rendered unsupported) by a member of S. The set S must also support the acceptable argument. In the example, the x is acceptable with respect to the set $\{\eta, a\}$. However, acceptability here is lacking in one respect in that it does not examine whether elements of S might themselves be attacked by other arguments (which would prevent S from supporting a). The concept of admissibility overcomes this issue, but to define it, we need two further notions.

Definition 8 ***(Conflict free and Self Supporting Sets)*** *A set of arguments S is conflict free iff $\forall y \in S, \nexists X \subseteq S$ such that $X R_a y$.*
A set of arguments S is self supporting iff $\forall x \in S$, S supports x.

Admissibility may now be defined as follows, so that in the example, $\{\eta, a\}$ is an admissible set.

Definition 9 ***(Admissible Set of Arguments)*** *A set of arguments S is said to be admissible iff*

1. *All elements of S are acceptable with respect to S.*
2. *The set S is conflict free.*

We are now in a position to define what sets of arguments a rational reasoner may find consistent. Dung named these sets "extensions". Multiple types of extension exist, representing, among others, sets of arguments that credulous and sceptical reasoners may find consistent. For example, given the arguments x ="The man is guilty because of reasons a, b, c", and y ="the man is innocent because of reasons d, e, f", where $a \ldots f$ are independent unrelated arguments, a sceptical reasoner would deem x, y inadmissible, as it has no way of choosing between them. A credulous reasoner would instead say that there are two scenarios, one where x is true, and one where y is true, and would not be able to choose between them.

Definition 10 ***(Extensions)*** *An admissible set S is an evidential preferred extension if it is maximal with respect to set inclusion. That is, there is no admissible set S' such that $S \subset S'$.*

An evidential grounded extension of an evidential argument framework EA containing the set of arguments A is the least fixed point of F_{EA}. Where

$$F_{EA} : 2^A \rightarrow 2^A$$
$$F_{EA}(S) = \{a | a \text{ is acceptable with respect to } S\}$$

Given a set of predicates P
Given a set of instantiated argument schemes I

1 For any $i = \langle\langle iASDefs, iASPrems, iASSupps, iASAtts, iASUcuts\rangle, iMap\rangle \in I$,
2 let $defaults(i) = \{subst(U, iMap) | U \in iASDefs\}$
3 let $premises(i) = \{subst(U, iMap) | U \in iASPrems\}$
4 let $supports(i) = \{subst(U, iMap) | U \in iASSupps\}$
5 let $attacks(i) = \{subst(U, iMap) | U \in iASAtts\}$
6 let $undercuts(i) = \{subst(U, iMap) | U \in iASUcuts\}$
7
8 Let the set of arguments $A = P \cup I \cup \{\eta\}$
9
10 $\forall p \in P$, $R_e = R_e \cup (\{\eta\}, p)$
11
12 if $\exists i \in I$, $p \subset P$ such that $p = premises(i)$
13 $R_e = R_e \cup (p, i)$
14 if $\exists i \in I$, $p \subset P$ such that $p = defaults(i)$
15 $R_a = R_a \cup (p, i)$
16 $\forall i \in I, A = A \cup supports(i)$
17 $\forall i \in I, \forall x \in supports(i), R_e = R_e \cup (i, x)$
18 $\forall i \in I, \forall x \in attacks(i)$, if $x \in A$, $R_a = R_a \cup (i, x)$
19 $\forall i \in I$, if $\exists j \in I, x \in undercuts(i)$
20 where $subst(x, j)$, $R_a = R_a \cup (i, j)$

Figure 2. The algorithm used to convert a set of argument schemes and predicates to an evidential argument system.

An argument framework may have multiple evidential preferred extensions, each representing a set of arguments that a credulous reasoner would find consistent. Only one grounded extension exists, representing the arguments a sceptical reasoner should agree with.

The evidential argument system above treats arguments as abstract entities. We must therefore map between our system, which uses predicates and argument schemes, and the abstract arguments found in an evidential argument system, as shown in the algorithm of Figure 2. This mapping allows us to determine which predicates and argument schemes, are, in a sense consistent, by evaluating the appropriate extension over the resultant evidential argument system.

Within the algorithm, lines 1 to 7 define syntactic shortcuts used in the rest of the algorithm, with line 8 defining the initial set of arguments, made up of the predicates, η, and the instantiated argument schemes. While predicates and instantiated argument schemes have different types, we do not differentiate between them at the abstract level in which evidential argumentation systems operate. The remainder of the algorithm populates the attacks and support relations according to intuitive rules; lines 12 and 13, for example, link premises to argument schemes, while lines 14 and 15 cause the presence of a default to attack an argument scheme. The supported conclusions of an instantiated argument scheme, as specified in lines 16 and 17, must be added to the set of arguments, and the appropriate support relations must also be instantiated. Line 18 instantiates attacks between an argument scheme and its conclusions, while the final line creates attacks between an instantiated argument scheme and the schemes it undercuts.

Running the algorithm results in a system containing more predicates and instantiated argument schemes than a rational reasoner would believe are justified. By computing the evidential argument system's grounded or preferred extension, we may determine what information and argument schemes are in fact consistent.

4 Argument Schemes for Normative Reasoning

We are now in a position to describe a number of argument schemes that an agent may use when reasoning about norms. We only consider obligations and permissions, and provide a very simple representation of these norms. Our first argument scheme deals with violations. We represent obligations using the one place predicate $obliged(G)$. If G does not hold, we assume that the obligation is violated. This leads to the following argument scheme:

$$< \{\}, \{obliged(G), not(G)\}, \{violated(obliged(G))\}, \{\}, \{\} >$$

When reasoning about obligations, an agent must reason about the state of the world if an obligation is honoured. We assume that an agent honours obligations by default, leading to the following argument scheme:

$$< \{not(ignored(obliged(G))), not(violated(obliged(G)))\}, \{obliged(G)\}, \{G\}, \{\}, \{\} >$$

Obligations may be conditional; for example, I may have an obligation to pay someone money if I buy something from them. We model this conditional by using the Modus Ponens argument scheme:

$$< \{\}, \{A, implies(A, B)\}, \{B\}, \{\}, \{\} >$$

(Our use of Modus Ponens, together with the introduction of the *violation(...)* predicate allows us to easily represent contrary to duty obligations.)

Now, permissions undercut obligations. That is, permissions explicitly allow us to ignore an obligation, by preventing the obligation argument scheme from coming into force.

$$< \{\}, \{permission(G)\}, \{\}, \{\}, \{obliged(not(G)\} >$$

$$< \{\}, \{permission(not(G))\}, \{\}, \{\}, \{obliged(G)\} >$$

Finally, we assume that an agent values some obligations more than others. This is captured, in the agent's internal reasoning, via a "higher priority" argument scheme. This argument scheme undercuts the effects of the application of a norm, i.e. an agent reasoning that one norm is higher priority than another will not attempt to honour the lower priority norm. The defeasible nature of argument means that if the higher priority norm is itself successfully attacked (i.e. is not admissible), the lower priority norm will be reinstated.

$$< \{\}, \{obliged(A), higherPriority(obliged(A), obliged(B)\}, \{\}, \{\}, \{(\langle\{not(violated(obliged(G)))\}, \{obliged(G)\}, \{G\}, \{\}, \{\} >, \{A, G\})\}\rangle$$

5 Argument Schemes and Normative Conflict

We have now presented a procedure for an agent to determine which portions of an argument system are admissible, and proposed a number of general purpose and norm related argument schemes. However, we have not yet described how an agent may make use of these schemes in deciding which action to take.

The agent needs to perform inference, determining, from its knowledge base and argument schemes, which predicates hold. One further complication appears as the predicates may be influenced by those norms that the agent is willing to violate. As specified earlier, we assume that an agent has a knowledge base KB containing those

```
01 Given a set of predicates KB
02 Given a set of argument schemes ASKB
03 i = 0
04 CS_0 = (A, R_a, R_e)
05 A = KB, R_a = R_e = {}
06
07 ∀p ∈ KB,  R_e = R_e ∪ ({η}, p)
08 repeat
09     i++
10     CS_i = CS_{i-1}
11     ∀AS = ⟨ASDefaults, ASPremises,
12           ASSupports, ASAttacks, ASUndercuts⟩ ∈ ASKB
13       if subst(ASPremises, M)
14            where M ⊆ A and CS_i = (A, R_a, R_e),
15         CS_i = CS_i ∪ ⟨AS, M⟩ ∪ subst(ASSupports, M)
16 until CS_i = CS_{i-1}
17 return CS_i
```

Figure 3. The algorithm used by the agent to infer a knowledge base given an initial knowledge base KB and some initial argument schemes $ASKB$.

predicates that it believes hold in the environment, and that it is aware of a set of argument schemes, stored in $ASKB$, so that we can create a new knowledge base CS consisting of the agent's original knowledge base, together with any inferences it may draw from its argument schemes. We may then calculate what may be inferred from CS, add it to CS, and repeat the process until no more inferences may be drawn. This is shown formally in Figure 3.

Note that the final CS_i will contain many instantiated argument schemes, and predicates, that may not be admissible. To determine what is admissible, we run the algorithm shown in Figure 2 over CS_i, starting at line 12, as we already have an argument framework. We may then compute the preferred extension to determine which predicates, and instantiated argument schemes, are actually admissible[5]. The presence of more than one preferred extension indicates that a normative conflict may exist. In this situation, the agent must explicitly add predicates of the form $not(honoured(O))$ (and possibly add additional predicates to reflect additional actions), where O is an obligation found in the agent's knowledge base, and rerun the reasoning algorithm, until only a single preferred extension exists.

By attempting to honour, or not honour different obligations, suppositional reasoning may take place. If the agent associates utility gains and losses with various predicates, it may determine what norms to ignore and honour in such a way as to maximise its utility.

6 Example

We illustrate the framework using a simple example. Before examining the agent's reasoning process in detail, we describe the scenario informally: A janitor (the agent) has an obligation to clean the floor, but needs a mop to do so. The mop is behind an alarmed door saying "authorised personnel only". Of course, being a janitor, he is authorised. However, the door is also alarmed, and the janitor is prohibited from setting off the alarm. However, since it is part of his job, cleaning the floor is more important than not setting off the alarm. Clearly, the agent should decide to open the door, even though it means setting off the alarm. We assume that the janitor agent has the following predicates in its knowledge base:

$implies(dirtyFloor, obliged(cleanFloor))$, $dirtyFloor$,
$obliged(not(openCupboard))$, $permission(openCupboard)$,
$implies(openCupboard, setOffAlarm)$, $obliged(not(setOffAlarm))$
$higherPriority(obliged(cleanFloor), obliged(not(setOffAlarm)))$

Apart from the argument schemes described above, we assume that the agent has access to a planner, allowing it to determine what needs to be done to achieve a goal. This is obviously a simplification, but is sufficient for the purposes of illustration.

In this case, the argument system has a single extension, containing all the predicates found in the agent's knowledge base (as no conflicts exist between them), as well as the predicates

$obliged(cleanFloor)$, $cleanFloor$, $openCupboard$, $setOffAlarm$

Figure 4 shows the resultant argument system. Here, if the "higher priority" argument scheme was not admissible, two preferred extensions would have existed. The agent would then have had to decide which norm to violate to yield a single preferred extension.

7 Discussion and Future Work

The reasoning mechanism proposed in this paper is powerful, allowing for reasoning about a wide variety of situations to take place. However, this power comes at a cost; argument schemes must be defined for any situation that must be reasoned about. We proposed a number of argument schemes that may be used by the agent to reason about norms. This work is, however, preliminary and additional normative argument schemes dealing with issues such as power should also be present. Furthermore, our representation of norms is simple; we have ignored issues such as norm activation, maintenance and discharge (to model such issues requires the introduction of reasoning about temporal artifacts). Finding and representing additional norm-related argument schemes forms the backbone of our future work. We also intend to enhance our model, allowing for additional features such as argument schemes containing an arbitrary number of arguments.

Apart from argument schemes and predicates such as "higher priority", we have no way of comparing the strength of attack or support. This concept is very useful, especially when dealing with uncertain concepts. Previous frameworks, such as the one described in [7], provide a powerful mechanism for assigning strength to arguments, but have difficulty handling loops. In the future, we aim investigate how argument strength may be integrated into the model.

Like AIF, the model described in Section 2 does not specify a structure for arguments, instead examining the interactions between predicates and argument schemes. Translating between our model and AIF is trivial, as concepts such as predicates map directly to I-Nodes, while argument schemes become R and C nodes. Unlike AIF, however, we are able to perform reasoning on our resulting structures. Recent work has suggested a more complex structure for argument schemes within AIF [9].

Our argument scheme structure, together with the presence of attack and support links within our abstract framework, enables us to easily model the most popular model of argument, namely the Toulmin model. Existing frameworks have long had difficulty supporting the richness of this model.

[5] Note that this is a rather brute force approach to computing what holds, a dialogue game based approach (as described in work such as [6]) may be more elegant and computationally efficient.

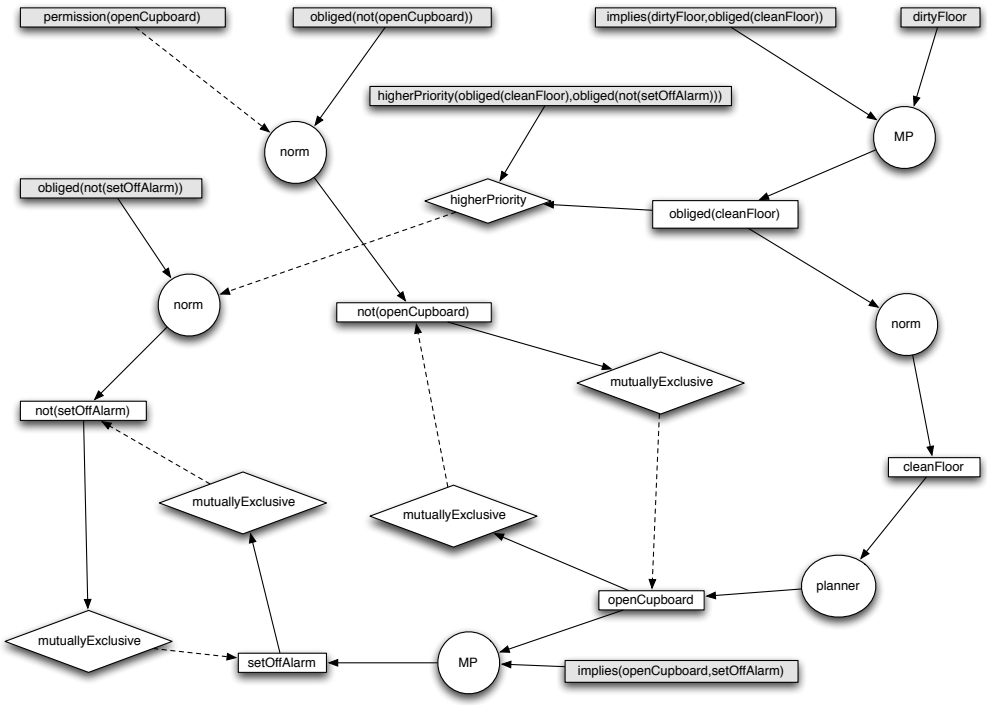


Figure 4. The evidential argument system representing the example; the shaded nodes represent those supported by η. Circular/oval nodes represent argument schemes intended to support certain conclusions, while diamond nodes represent attacking argument schemes. These are treated identically in the framework, but are differentiated for visual clarity. Solid arrows indicate support, while dashed arrows represent attacks.

8 Conclusions

In this paper we showed how an agent may use argumentation schemes and predicates to model its environment and normative state. By transforming this model into an evidential argument system, and computing the extension of this resulting argument system, the agent could perform different types of normative reasoning. This reasoning included detecting conflicts between norms, using additional knowledge from its knowledge base to overcome this conflict, and checking whether it has indeed fulfilled (or decided to ignore) all norms that affect it. Our framework was inspired by the way humans appear to reason when dealing with norms, and is able to easily handle normative conflict. Our approach is easily extensible, by adding extra domain specific argument schemes, the agent can cope with additional knowledge.

Acknowledgement: This work was undertaken as part of the CONTRACT project which is co-funded by the European Commission under the 6th Framework Programme for RTD with project number FP6-034418. Notwithstanding this fact, this paper and its content reflects only the authors' views. The European Commission is not responsible for its contents, nor liable for the possible effects of any use of the information contained therein.

REFERENCES

[1] Trevor Bench-Capon, 'Value based argumentation frameworks', in *Proceedings of the 9th International Workshop on Nonmonotonic Reasoning*, pp. 444–453, Toulouse, France, (2002).

[2] Claudette Cayrol and Marie-Christine Lagasquie-Schiex, 'On the acceptability of arguments in bipolar argumentation frameworks', in *Pro. of the Eighth European Conference on Symbolic and Quantitative Approaches to Reasoning With Uncertainty*, volume 3571 of *LNAI*, pp. 378–389. Springer-Verlag, (2005).

[3] Carlos Chesñevar, Jarred McGinnis, Sanjay Modgil, Iyad Rahwan, Chris Reed, Guillermo Simari, Matthew South, Gerard Vreeswijk, and Steven Willmott, 'Towards an argument interchange format', *Knowl. Eng. Rev.*, **21**(4), 293–316, (2006).

[4] Phan Minh Dung, 'On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games', *Artificial Intelligence*, **77**(2), 321–357, (1995).

[5] Søren Holbech Nielsen and Simon Parsons, 'A generalization of Dung's abstract framework for argumentation: Arguing with sets of attacking arguments', in *Proceedings of the Third International Workshop on Argumentation in Multi-Agent Systems*, pp. 7–19, Hakodate, Japan, (2006).

[6] Nir Oren, *An Argumentation Framework Supporting Evidential Reasoning with Applications to Contract Monitoring*, Phd thesis, University of Aberdeen, Aberdeen, Scotland, 2007.

[7] Nir Oren, Timothy J. Norman, and Alun Preece, 'Subjective logic and arguing with evidence', *Artificial Intelligence Journal*, **171**(10–15), 838–854, (2007).

[8] John L. Pollock, *Cognitive Carpentry*, Bradford/MIT Press, 1995.

[9] Iyad Rahwan, Fouad Zablith, and Chris Reed, 'Laying the foundations for a world wide argument web', *Artif. Intell.*, **171**(10-15), 897–921, (2007).

[10] Douglas N. Walton, *Argumentation Schemes for Presumptive Reasoning*, Erlbaum, 1996.