Dynamic Verification of Trust in Distributed Open Systems

Categories:
Multiagent Systems: Distributed AI / Trust / Verification (Model Checking)

Abstract

In open and distributed systems, agents must engage in interactions of which they have no previous experience. Deontic models are widely used to describe aspects of permission, obligation, and trust anticipated by such agents, but no practical mechanism has been developed for testing deontic trust specifications against models of multi-agent interactions. This paper describes a way of doing this; an implementation of it via model checking; and some preliminary results on a realistic example.

1 Introduction

In large-scale open distributed systems, trust remains a fundamental challenge. Despite much research, the notion of trust remains vague and there is no consensus on what exactly trust is in systems such as multiagent systems (MAS). This is because trust may be addressed at different levels. At the low system level, trust is associated with network security, such as authentication (determining the identity of the user entity), access permissions (deciding who has access to what), content integrity (determining whether the content has been modified), content privacy (ensuring that only authorised entities can access the content), etc. At higher levels, the focus is on trusting entities — these may be human users, software agents, services, directories, etc. — to perform actions as requested, provide correct information, not to misuse information, execute protocols correctly, etc.

Available research has mainly focused on analysing, improving, and developing strategies that address trust issues at various system levels. In this paper we focus not on the strategies, but on the possibility of specifying and verifying such strategies. We therefore inherit the general definition of trust; trust is defined as the problem of who to interact with, when to interact with them, and how to interact with them. We then show how the specification and verification methods of [Osman *et al.*, 2005] may be used to specify and verify trust models at various system levels. We use the Lightweight Coordination Calculus (LCC) of [Robertson, 2004] for modelling global interaction models. Local trust constraints, on the agents level, are modelled via a simple trust policy language introduced in Section 3.2. These models may then be fed to a lightweight dynamic model checker [Osman *et al.*, 2005] for verifying interesting trust properties, which go beyond liveness and safety properties verified by traditional verification techniques (Section 4). The result is a powerful, yet simple, verification mechanism. The verifier itself is lightweight, delegating the complexity of managing the search space to the underlying XSB tabled Prolog system [Sagonas *et al.*, 1994].

We open with a motivating example in Section 2. Section 3 provides an overview of our system model and the languages used for specification. The verification process is introduced in Section 4, before concluding with our results in Section 5.

2 Motivating Example: an Auction System

Section 3 presents our 2-layered architectural approach for distributed open systems. Similar to web service architectures, the basic idea is that a global interaction model is used to specify the rules of the interaction, irrespective of the agents engaged in this interaction. Then each agent, based on its local constraints, tries to find the most suitable interaction protocol along with the most suitable group of agents to interact with. We call the agents' local constraints the deontic constraints, since they specify the agents permissions, prohibitions, and obligations.

Now let us consider the case where an agent is interested in engaging in an auction scenario for either selling or buying a specific item. Trust issues automatically arise on two levels. These may be summarised by the following two questions:

- Which interaction protocol should the agent engage in?
- In such an interaction, which agents does it engage with?

For example, before selling its item, the auctioneer will have to pick the appropriate interaction protocol, where appropriateness is measured by the satisfiability of certain properties. Traditional properties to check for are usually liveness and safety properties. For example, the auctioneer may decide that the interaction protocol is trusted only if it is deadlock free (trust issue TI_1 of Figure 1). A much more interesting set of properties may be obtained when tackling domain specific issues. For example, a more challenging trust issue to verify is whether the interaction protocol enforces truth-telling by the bidders or not (trust issue TI_2 of Figure 1).

For a given interaction protocol, each agent will then have to select the appropriate agents for such an interaction. The goal is to achieve a set of agents that trust each other. For

TI_1: Is the interaction protocol deadlock free?

TI_2: In such an interaction, can the bidders be better off if they bid either a lower or a higher value than their true valuation?

TI_3: If from previous experience the agent knows that DVDs from auctioneer *A* are not original, then *A* is not trusted in delivering good quality DVDs.

TI_4: If the auctioneer agent A is not trusted in delivering good quality DVDs, then it is not trusted in delivering good quality CDs.

TI_5: Agent A is trusted to take the role of the auctioneer only if it has decent ratings and holds a good reputation, with more importance given to the most recent ratings.

Figure 1: The auction scenario – some trust issues

$$\begin{tikzcd}
\arrow[r] & S_0 \arrow[loop right, "message(a(auctioneer,A),\ a(bidder,B_i),\ invite(I,R))"] \arrow[d, "message(a(auctioneer,A),\ a(bidder,B_i),\ invite(I,R))"] \\
& S_1 \arrow[loop right, "message(a(bidder,B_i),\ a(auctioneer,A),\ bid(V_i))"] \arrow[d, "message(a(bidder,B_i),\ a(auctioneer,A),\ bid(V_i))"] \\
& S_2 \arrow[d, "message(a(auctioneer,A),\ a(bidder,B_i),\ won(I,V))"] \\
& S_3 \arrow[d, "message(a(bidder,B_i),\ a(auctioneer,A),\ payment(P))"] \\
& S_4
\end{tikzcd}$$

where *message(A,B,M)* represents the transmission of message *M* from *A* to *B*

Figure 2: The auction scenario – the interaction's state-space

example, one bidder may trust auctioneer *A* in selling anything except DVDs — possibly, due to previous experience, it now knows that these DVDs are not original. It may also use socio-cognitive models of trust to learn that if the DVDs are not original, then most probably the CDs will not be too (trust issues TI_3 and TI_4 of Figure 1). Another widely used trust mechanism is the use of ratings and reputations. The agents should be capable of collecting (or having access to) each other's rating. It is then up to each agent to aggregate these ratings as they see fit. For example, a bidding agent might decide not to trust new auctioneers with no selling history. An average rating is then required, possibly giving more importance to the latest ratings (trust issue TI_5 of Figure 1).

The trust issues of Figure 1 cover a wide sample of the various trust mechanism in the literature [Ramchurn *et al.*, 2004]: from socio-cognitive models (TI_4 of Figure 1), to evolutionary and learning models (TI_3 of Figure 1), reputation mechanism (TI_5 of Figure 1), trustworthy interaction mechanisms ((TI_2 of Figure 1)), etc. While we do not specify how trust is learned, we focus on the agent's individual aggregation mechanisms and their specification, which is essential for verifying trust. For example, while we do not focus on how the agent obtains the ratings of another (TI_5 of Figure 1), we do require the specification of how these ratings are aggregated.

The main goal of this paper is to show how agents may answer these questions by using a dynamic model checker. In our running example, the agent's constraints used (or the trust constraints) are those of Figure 1. The interaction protocol verified is presented by the state-space graph of Figure 2.

The interaction of Figure 2 is that of a Vickrey auction. The interaction starts at state s_0 when the auctioneer *A* sends an invite to a set of bidders for bidding on item *I* with a reserve price *R*. The interaction remains at state s_0 until invites are sent to all bidders. Then the bidders send their sealed bids back to the auctioneer. This is represented by state s_1 of Figure 2. When all bids are collected, the interaction moves to the new state s_2. The auctioneer informs the winner of the price *V* to be paid, moving the interaction to state s_3. Finally, the winning bidder sends its payment *P*, and the interaction is completed at state s_4.

Before we present the verification mechanism used (Section 4), Section 3 introduces our system model's architecture and the languages used to specify such a system.

3 System Modelling

We view MAS systems as a collection of autonomous agents. The system is open and distributed. Various agents may join or leave the system at any time. Interactions become the backbone that holds the system together. Agents group themselves into different, and possibly multiple, interactions. Figure 3 provides such an example. A collection of agents are grouped into three different interactions (or scenarios): two auction scenarios and a trip planning scenario.

Due to the dynamic nature of the system, we believe interaction groups should be created dynamically and automatically by the agents. We also believe everything should be distributed. This implies that there should be no higher layer for coordination, control, synchronisation, etc. It is the agents' responsibility to group themselves into different scenarios. As a result, we split the MAS model into two layers: the interaction layer and the agents layer.

The interaction model specifies the rules and constraints on the interaction. This indicates how exactly the interaction may be carried out. The agents' models specify the rules and constraints on the agents. These are the agents' permissions, prohibitions, and obligations; we therefore call this model the deontic model (Figure 3). Note that for one scenario there is one global interaction model and several local deontic models. While agents need to share the interaction model in order to know the rules of the interaction they're engaged in, each agent will have its own local constraints in its deontic model.

In what follows, we introduce the languages used in specifying these models.

3.1 The Interaction Model

We choose the Lightweight Coordination Calculus (LCC) [Robertson, 2004] for modelling the interaction's state-space graph, since it is the only executable process calculus for MAS that we are aware of[1]. Having an executable process calculus for modelling the interaction's state-space graph is very useful for verifying the executable models of interaction. Figure 4 presents the syntax of LCC.

[1]Note that we are aware of other coordination systems, but none of these are based directly on a process calculus

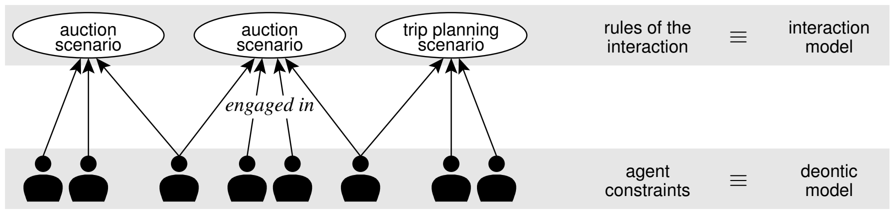


Figure 3: The MAS model – a 2-layered architecture model

$$
\begin{array}{rcl}
Interaction & := & \{Clause, \ldots\} \\
Clause & := & Agent :: ADef \\
Agent & := & a(Role, Id) \\
ADef & := & null \leftarrow C \mid Agent \leftarrow C \mid Message \leftarrow C \mid \\
 & & ADef\ then\ ADef \mid ADef\ or\ ADef \mid \\
 & & ADef\ par\ ADef \\
Message & := & M \Rightarrow Agent \mid M \Leftarrow Agent \\
C & := & Term \mid C \wedge C \mid C \vee C \\
Role & := & Term \\
M & := & Term
\end{array}
$$

$null$ denotes an event which does not involve message passing.
$Term$ is a structured term in Prolog syntax.
Id is either a variable or a unique agent identifier.

Figure 4: LCC syntax

Agents, in LCC, are defined by their roles and identifiers. An interaction is defined by a set of clauses. A clause gives each agent role a definition that specifies its acceptable behaviour. An agent can either do nothing (usually used for internal computations), take a different role, or send/receive messages ($M \Rightarrow A$, $M \Leftarrow A$). Agent definitions are constructed using the sequential ($then$), choice (or), parallel composition (par), and conditional ($\leftarrow$) operators. The conditional operator is used for linking constraints to message passing actions.

Example Revisited

Figure 5 presents the specification of the state-space graph of Figure 2 via LCC. The auctioneer *A*, knowing the item *I*, the reserve price *R*, and the set of bidders *Bs*, recursively sends an invite to all bidders in set *Bs*. It then takes the role of *auctioneer2* to collect bids, send the winner a message, collect payment, and deliver the item won. On the other side, each bidder agent receives an invite from the auctioneer and sends its bid based on its valuation. The winner receives a *win* message and then sends its payment *P*.

3.2 Deontic Models

In this paper, we focus on the *trust* constraints imposed by the agents. We propose a trust policy language for the specification of trust rules. The language is similar to other logic-based policy languages which are built on deontic concepts, such as ASL [Jajodia *et al.*, 1997], RDL [Hayton *et al.*, 1998], and Rei [Kagal *et al.*, 2003]. The syntax of our language is presented by Figure 6.

The syntax states that trust rules might either hold in general or under certain conditions: $TrustSpecs\ [\leftarrow Condition]$. The interaction's trustworthiness is modelled by

```
a(auctioneer(I, R, Bs), A)  ::
    ( invite(I, R) ⇒ a(bidder, B) ← Bs = [B|T] then
      a(auctioneer(I, R, T), A) )
    or
    a(auctioneer(Bs, []), A) ← Bs = [].

a(auctioneer2(Bs, Vs), A)  ::
    append([B, V], Vs, Vn) ← bid(B, V) ⇐ a(bidder, B) then
    ( a(auctioneer(Bs, Vn), A) ← not(all_bid(Bs, Vn))
      or
      ( win(B1, V2) ⇒ a(bidder, B1)
          ← all_bid(Bs, Vn) and
              highest(Vn, B1, _) and second_highest(Vn, _, V2) then
        deliver(I, B1) ← payment(P) ⇐ a(bidder, B1) ) ).

a(bidder, B)  ::
    invite(I, R) ⇐ a(auctioneer(_, _, _), A) then
    bid(B, V) ⇒ a(auctioneer(_, _), A ← valuation(I, V) then
    win(Bi, Vi) ⇐ a(auctioneer(_, _), A) then
    payment(P) ⇒ a(auctioneer(_, _), A) ← Bi = B and payment(P).
```

Figure 5: LCC specification for the interaction of Figure 2

$$
\begin{array}{rcl}
TrustRule & := & TrustSpecs\ [\leftarrow Condition] \\
TrustSpecs & := & trust(interaction(IP), Sign) \mid \\
 & & trust(Agent, Sign) \mid \\
 & & trust(Agent, Sign, Action) \\
Agent & := & a(Role, Id) \\
Sign & := & + \mid - \\
Action & := & \text{MPA} \mid \text{N-MPA} \mid TrustSpecs \\
\text{MPA} & := & Message \Rightarrow Agent \mid \\
 & & Message \Leftarrow Agent \\
Condition & := & Condition \wedge Condition \mid \\
 & & Condition \vee Condition \mid \\
 & & Temporal \mid Term \\
Role, \text{N-MPA}, Message & := & Term
\end{array}
$$

where, $[X]$ denotes zero or one occurrence of X,
IP is an interaction protocol specified in LCC,
Id is either a variable or a unique agent identifier,
$Temporal$ is a temporal property whose syntax is specified in [Osman *et al.*, 2005], and
$Term$ is either a variable or a structured term in Prolog syntax.

Figure 6: Syntax of our trust policy language

$trust(interaction(IP), Sign)$, where IP is the LCC specification of the interaction protocol in question (e.g. the interaction protocol of Figure 5). The agent's trustworthiness is modelled by $trust(Agent, Sign)$, where $+$ and $-$ values of $Sign$ are used to model trust and distrust, respectively. Only if the agent is trustworthy, it can engage in an interaction. Trusting or distrusting agents to perform specific actions is modelled by $trust(Agent, Sign, Action)$. Actions could either be message passing actions (MPA) — such as sending

($Message \Rightarrow Agent$) or receiving ($Message \Leftarrow Agent$) messages — or non-message passing actions (N-MPA) — such as performing computations. We also allow actions to take the form of another trust rule (see $TrustSpecs$ in the $Action$ definition). This supports the delegation of trust, since it permits the specification of whether an agent's trust itself is to be trusted or not.

Example Revisited

Trust issues TI_3, TI_4, and TI_5 of Figure 1 address trust at the agent level. In what follows, we present the specification of the more complex trust rule, TI_5, in our trust policy language. The rule is presented by Figure 8(a). It specifies that agent A is trusted as an auctioneer only if it has a selling history of at least 50 items and an average rating above 70%, going up to 90% for the latest 20 transactions. The mechanism presented here distrusts new entrants and focuses on the agent's latest ratings rather than the overall one.

Trust issues TI_1 and TI_2 of Figure 1 address trust at the interaction level. The conditions for trusting an interaction protocol are specified via a temporal language. Since the 'deadlock free' property of TI_1 is a straightforward property to model via a temporal language, we show how the more challenging property of 'enforcing truth-telling by the bidders' (TI_2) may be specified:

To prove the protocol enforces truth-telling by the bidders, we prove that, for the interaction protocol of Figure 5, the bidders cannot do any better than bidding their true valuation *V*: the maximum value they are willing to pay. For this, we study the two cases: (1) if the competing agent bids a higher value C_h, and (2) if the competing agent bids a lower value C_l. In Figure 7, the grey circle represents the bidder and its valuation *V*. The other two circles represent the two cases of the competing agent, which may bid either a higher or a lower value: C_h and C_l, respectively. For each of these two cases, the bidder may either bid its true valuation, a value between its true valuation and that of its competitor, or a value beyond that of the competitor (Figure 7). The trust rule of Figure 8(b) studies all 6 cases, respectively. For each bid, the winner and the price won at are computed through the temporal property *[bid(Bidder,Bid1), bid(Competitor,Bid2)] <win(Winner,Price)> tt*[2]. The bidder should, naturally, be expected to lose to its competitor in cases 1, 2, and 6. It should win in cases 3, 4, and 5, where case 4 would be the only case where the bidder bids its true valuation and wins the item for the price *Y*. The trust rule above requires that the bidder is not better off when winning in cases 3 and 5, where the item is won for prices *X* and *Z*, respectively. This is expressed by the conditions *X<Y* and *Z<Y*.

The *InteractionProtocol* of the trust rule presented by Figure 8(b) is in fact the LCC protocol of Figure 5. This relatively complex LCC structure[3] is passed entirely as a parameter to the trust rule. The interaction protocol is then said to be trusted if the condition — the collection of temporal properties — is satisfied. In this example, the condition verifies, via the six temporal properties of Figure 8(b), the possible occurrence of transmitting the $bid(\_,\_)$ and $win(\_,\_)$ messages in the LCC interaction protocol of Figure 5.

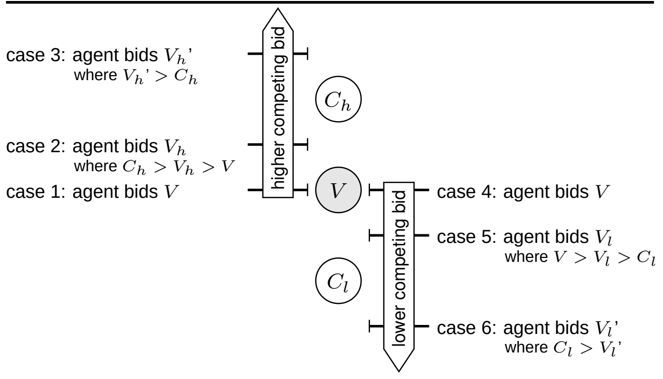


Figure 7: The bidding strategies – 6 cases

trust(a(auctioneer,A), +) ←
rating_count(a(auctioneer,A), Total) and Total > 50 and
rating_average(a(auctioneer,A), Average) and Average > 0.7 and
rating_latest(a(auctioneer,A), 20, Latest) and Latest > 0.9.

(a) Trust rule TI_5

trust(a(interaction(InteractionProtocol)), +) ←
V_l'<C_l and C_l<V_l and V_l<V and V<V_h and V_h<C_h and C_h<V_h' and
[bid(Bidder,V), bid(Competitor,C_h)] <win(Competitor,_)> tt and
[bid(Bidder,V_h), bid(Competitor,C_h)] <win(Competitor,_)> tt and
[bid(Bidder,V_h'), bid(Competitor,C_h)] <win(Bidder,X)> tt and
[bid(Bidder,V), bid(Competitor,C_l)] <win(Bidder,Y)> tt and
[bid(Bidder,V_l), bid(Competitor,C_l)] <win(Bidder,Z)> tt and
[bid(Bidder,V_l'), bid(Competitor,C_l)] <win(Competitor,_)> tt and
X<Y and Z<Y.

(b) Trust rule TI_2

Figure 8: Specification of trust rules

It is worth noting that the trust rule is restricted to the auctions domain, yet independent of the specific auction protocol it is verified upon. For example, the six cases of Figure 7 are comprehensive, even for auction systems consisting of more than two agents. This is because verifying the utility of one agent should take into consideration only the competing agent's bid — all other agents' bids are irrelevant. Furthermore, the verification of such a trust rule will terminate with correct results, whether positive or negative, for any auction protocol that requires agents to place their bids before a message is transmitted informing who the winner is. It will probably fail when verified against more complex auction protocols, such as those selling multiple items and having several winners. However, the verification will be success-

[2]The temporal property *[X,Y]<Z>tt* specifies that if messages *X* and *Y* are sent, then message *Z* will eventually be received.

[3]For a comparison between the LCC process calculus and traditional process calculi, such as Milner's CCS [Milner, 1989] and Hoar's CSP [Hoare, 1985], the interested reader may refer to [Osman *et al.*, 2005]. From this comparison, we may assert that the complexity of LCC is similar to that of traditional process calculi, such as CCS and CSP.

ful in the most common auctions, such as the English, Dutch, sealed first-price, and sealed second-price auctions. Allowing agents to automatically verify such properties (which are relatively independent of the specific interaction protocol), aids the agents in making their protocol selection when faced with new and unexplored protocols.

4 The Dynamic Model Checker

We believe it is solely the agents' responsibility to answer the question of how, when, and who to interact with. Ideally, in a highly dynamic system, this should be done at runtime by deciding which combination of interaction and deontic (trust) models is currently suitable. But is this feasible?

We choose *model checking* from amongst other verification techniques because it provides a fully automatic verification process which could be carried out by the agents during interaction time. We show how *interaction time verification* is made possible with the use of a remarkably efficient (see Figure 11) and lightweight model checker. The concerned agent can then feed the model checker with the system model: the combination of the global interaction model and agents' local trust rules. The model checker should then verify whether the trust rules are consistent amongst themselves as well as consistent with respect to the interaction model. It should be capable of detecting non-suitable (distrusted) agents and/or interaction protocols.

4.1 Model Checking: an Overview

The model checking problem can be defined as follows: *Given a finite transition system S and a temporal formula ϕ, does S satisfy ϕ?* The model checking process is divided into three stages: modelling, specification, and verification. The system to be verified must first be modelled in the language of the model checker S. The properties (ϕ) to which the system model is verified upon should be specified using the model checker's temporal logic. Both the system model and the properties specification are fed to the model checker for verification. The model checker is, essentially, an algorithm that decides whether a model S satisfies a formula ϕ. Figure 9 illustrates the model checking process.

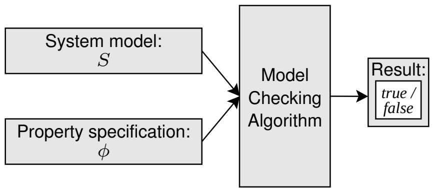


Figure 9: The model checking process

4.2 Model Checking: the Implementation

In traditional model checking, the system model represents a state-space graph. The requirements on a given state-space are then specified through temporal properties in the property specification section. In our case, the system model is a combination of the shared interaction's state-space and the agents' local trust constraints. Trust constraints based on temporal properties, such as those constraining the interaction (e.g. TI_2 of Figure 8(b)), are modelled in the property specification section. Other trust rules constraining the agents (e.g. TI_5 of Figure 8(b)) are kept on the deontic level of the system model. Figure 10 provides an overview of our trust verifying model checker. The sample input data is that of the auction system scenario. The interaction model example of Figure 10 is a copy of that of Figure 5 — the LCC specification of Figure 2. The deontic model example, or the trust constraints on agents, is that of Figure 8(a). The property specification is the same as the trust constraint on the interaction of Figure 8(b).

The model checker should then verify that the interaction model is trusted and that the agents are trusted to engage in the given interaction. To verify this, the concerned agent feeds the model checker with the appropriate interaction model along with the trust rules that are split between the deontic model and the property specification. The verification process is explained in the following section.

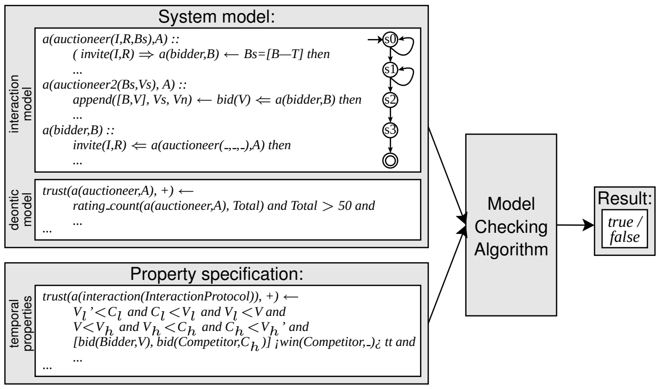
The Verification Process

In this section, we present an overview of the model checking algorithm [Osman *et al.*, 2005] — the black box of Figure 10. The verification process is carried out as follows. A local model checker partially constructs the state-space one step at a time until a solution is reached. Verification starts at the initial state s_0 and tries to verify that the property ϕ is satisfied at s_0[4]. If it succeeds, the verifier terminates and the property is said to be satisfied. Otherwise, the verifier attempts to make a transition(s) to the next state(s) in the state-space[5]. If the transition(s) violates any of the trust (deontic) rules, then the verification process terminates and the property is not satisfied. Otherwise, the satisfaction of the property ϕ is verified with respect to the new state(s), and the whole process is repeated all over again.

The result is a remarkably compact model checker built on top of the XSB system [Sagonas *et al.*, 1994], a tabled Prolog

[4]Satisfaction is verified by applying the modal μ-calculus proof rules presented in [Osman *et al.*, 2005].

[5]Transitions are made based on the LCC transition rules presented in [Osman *et al.*, 2005].



```
a(auctioneer(I,R,Bs),A) ::
      ( invite(I,R) ⇒ a(bidder,B) ← Bs=[B—T] then
      ...
a(auctioneer2(Bs,Vs), A) ::
      append([B,V], Vs, Vn) ← bid(V) ⇐ a(bidder,B) then
      ...
a(bidder,B) ::
      invite(I,R) ⇐ a(auctioneer(_,_,_),A) then
      ...
```

```
trust(a(auctioneer,A), +) ←
      rating_count(a(auctioneer,A), Total) and Total > 50 and
      ...
...
```

```
trust(a(interaction(InteractionProtocol)), +) ←
      V_l'<C_l and C_l<V_l and V_l<V and
      V<V_h and V_h<C_h and C_h<V_h' and
      [bid(Bidder,V), bid(Competitor,C_h)] ¡win(Competitor,_)¿ tt and
      ...
...
```

Figure 10: The auction scenario – the model checker's input

system. This compactness is achieved by placing the burden of searching the state-space on the underlying XSB system.

5 Conclusion and Results

Trust is the key to the success of large-scale distributed systems. Most of the work carried on in the field of trust has focused on the strategies for ensuring trusted interactions. In this paper, we have presented a mechanism for specifying and verifying such strategies. Our mechanism allows the involved agents to dynamically and automatically invoke the model checker at run-time, when the conditions for verification are met, and verify that certain trust requirements will not be broken for a given scenario with a given set of collaborating agents.

The complexity of verifying multi-agent systems relies on the complexity of the different roles in the interaction, rather than the number of agents involved in such an interaction. As presented by Section 3.1, interaction protocols are defined in terms of agents' roles instead of the individual agents that might be engaged in the interaction. For example, there are two roles for our auction scenario: the role of the auctioneer and that of the bidder. All bidders then share the same protocol specification (Figure 5), irrespective of their different local deontic constraints. The complexity of verifying the five trust issues of Figure 1 for the auction scenario of Figure 2 (or Figure 5) depends on the complexity of the auctioneer's and bidder's role definition. In our scenario, if the trust issues are verified for a set of one auctioneer agent and two bidder agents, then the results will hold for a set of one auctioneer agent and n bidder agents, where $n > 2$. The trick is to know the minimum number of agents required for verifying certain properties of a given interaction protocol. We assume such information is provided with the interaction protocol.

In our example, we set the scene to incorporate one auctioneer and two bidders. The dynamic model checker is then invoked for verifying the five trust issues of Figure 1 against the auction scenario of Figure 5. The results are presented by Figure 11.

	TI_1	TI_2	TI_3	TI_4	TI_5
CPU time (sec)	0.017	0.204	0.020	0.021	0.010
Memory usage (MB)	1.327	14.052	1.356	1.376	0.917

Figure 11: Preliminary results

The novelty of our work is in introducing *interaction time verification* and applying it to the field of trust in multi-agent systems. This is made possible by using a relatively compact, yet efficient, dynamic model checker which allows the agents to invoke the model checker at run-time for verifying various trust issues. The model checker is compact enough (implemented in 150 lines of Prolog code) and efficient enough (as shown by the results of Figure 11) for allowing agents to perform the verification at interaction time.

References

[Hayton *et al.*, 1998] Richard Hayton, Jean Bacon, , and Ken Moody. Access control in an open distributed environment. In *Symposium on Security and Privacy*, pages 3–14, Oakland, CA, 1998. IEEE Computer Society Press.

[Hoare, 1985] Charles Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[Jajodia *et al.*, 1997] Sushil Jajodia, Pierangela Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy (SP '97)*, page 31, Washington, DC, USA, 1997. IEEE Computer Society.

[Kagal *et al.*, 2003] Lalana Kagal, Tim Finin, and Anupam Joshi. A policy language for a pervasive computing environment. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*. IEEE Computer Society, June 2003.

[Milner, 1989] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Osman *et al.*, 2005] Nardine Osman, David Robertson, and Christopher Walton. Run-time model checking of interaction and deontic models for multi-agent systems. In *Proceedings of the Third European Workshop on Multi-Agent Systems*, 2005.

[Ramchurn *et al.*, 2004] Sarvapali D. Ramchurn, Dong Hunyh, and Nicholas R. Jennings. Trust in multi-agent systems. *Knowledge Engineering Review*, 2004.

[Robertson, 2004] David Robertson. A lightweight coordination calculus for agent social norms. In *Proceedings of Declarative Agent Languages and Technologies workshop*, 2004.

[Sagonas *et al.*, 1994] Konstantinos Sagonas, Terrance Swift, and David S. Warren. XSB as an efficient deductive database engine. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data (SIGMOD '94)*, 1994.