

Run-Time Model Checking of Interaction and Deontic Models for Multi-Agent Systems

Nardine Osman
School of Informatics
The University of Edinburgh
Edinburgh EH8 9LE, UK
N.Osman@sms.ed.ac.uk

David Robertson
School of Informatics
The University of Edinburgh
Edinburgh EH8 9LE, UK
dr@inf.ed.ac.uk

Christopher Walton
School of Informatics
The University of Edinburgh
Edinburgh EH8 9LE, UK
cdw@inf.ed.ac.uk

ABSTRACT

This paper is concerned with the problem of obtaining predictable interactions between groups of agents in open environments when individual agents do not expose their BDI logic. The most popular approaches to this in practise have been to model interaction protocols and to model the deontic constraints imposed by individual agents. Both of these approaches are appropriate and necessary but their combination creates the practical problem of ensuring that interaction protocols come into contact with agents that possess compatible deontic constraints. This is essentially an issue of property checking dynamically at run-time. We show how model checking can be applied to this problem.

Categories and Subject Descriptors

I.2.11 [Distributed Artificial Intelligence]: Multiagent systems; F.3.1 [Specifying and Verifying and Reasoning about Programs]: Specification techniques; D.2.4 [Software / Program Verification]: Model checking

General Terms

Verification

Keywords

dynamic model checking, interaction models, deontic models

1. INTRODUCTION

One of the most fundamental challenges of multi-agent system engineering is to enable predictable, reliable interaction amongst groups of agents without requiring a deep standardisation of the way in which they are engineered and while preserving as much as possible of their autonomy in individual reasoning. It is not plausible that agents built independently and with no agreement on forms of interaction could be predictably reliable, so researchers have searched

for ways of standardising some aspects of coordination in the hope that this small amount of standardisation would provide predictability sufficient for important tasks. Two contrasting approaches to this problem have emerged:

- The use of explicit **models of interactions** (in a generic process or state-machine language) along with mechanisms by which agents can locate, reason about and participate in models of interaction that they judge appropriate. These are built to describe forms of interaction (thus are detached from individual agents) and typically are accessed when an agent anticipates it wants to initiate or join that type of interaction.
- The specification of constraints imposed by individual agents on the interactions they will allow: **deontic constraints**. These are built locally for an individual agent and typically accessed when a specific interaction is anticipated with an individual agent.

These two approaches are compatible, in the sense that they attack different aspects of a similar problem. They have not, however, been combined. Hitherto, this has not been a major problem, because neither approach had significant user communities. Those now are developing so that it is of practical importance to have a way of answering the question “Given an interaction model and an agent with given deontic constraints wishing to participate in that model, could that combination work?” This is a difficult question because both interaction models and deontic constraints are sophisticated logical objects, so the inference involved is not (say) simple term matching or subsumption. It also is difficult because in practise the question must be answered automatically and in real time, so inference mechanisms must be self contained and efficient. We show how a lightweight dynamic model checker can tackle this problem by allowing agents to invoke it at interaction time for verifying both the interaction and deontic models.

2. MOTIVATING EXAMPLE

Let us consider the travel agency scenario [1], where a customer agent, interested in making flight and hotel reservations, contacts a broker to find the most suitable travel agent for this scenario. The travel agent will have to deal with airline web services, query a hotel directory, and confirm customer’s payments via a credit card web service.

The interaction model will specify the global rules of interaction. For example, it states that the interaction starts

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS’06 May 8–12 2006, Hakodate, Hokkaido, Japan.
Copyright 2006 ACM 1-59593-303-4/06/0005 ...\$5.00.

when the customer agent sends it vacation details (dates / destination) to the travel agent. Then, the travel agent sends a query to the involved airline web services with the given vacation details, and so on. The interaction model may be thought of as a state-space graph that specifies the allowed order of message passing actions between agents.

On the other hand, the deontic model specifies local individual constraints imposed on the various agents involved in the interaction. For example, a constraint on the broker agent is to not accept a travel agent that has no access to the selected hotel directory. Several constraints may be imposed on the customer agent. For example, it might not have access to a *MasterCard*, or it may be forced to encrypt its messages in a specific technology, say OpenPGP only. Similarly, other agents would have other sets of constraints.

The system is modelled through the combination of interaction and deontic models (Figure 1). Note that unlike the global interaction model which constructs a unique state-space graph for coordinating interactions, the deontic model is a collection of the involved agents' local constraints. These constraints could address a diverse collection of issues. They are usually used to tackle issues such as access control, authorisation, authentication, trust, security, etc.

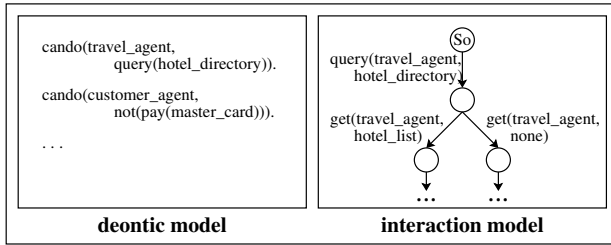


Figure 1: A conceptual model of the system model

Now let us consider the following complication. The customer agent contacts the broker agent requesting an appropriate travel agent. With the broker being responsible for finding suitable agents for a given interaction protocol, it should also be capable of verifying, at interaction time, that the protocol it has instantiated with agents is likely to work. This, however, relies on the correctness of the interaction protocol as well as the compatibility of the chosen agents. For example, trying to ally the customer agent with a travel agent that does not have access to a hotel directory will result in a scenario failure, regardless of whether the interaction protocol itself is error free or not. This requires a verifier that can handle both interaction and deontic constraints, and is capable of operating automatically at run-time.

3. DESIGN & IMPLEMENTATION PLANS

Our goal is to achieve a verifier which could be used by agents at interaction time for verifying MAS through the verification of the interaction and deontic rules. Figure 2 illustrates our design plan.

Because open systems consist of autonomous agents, it is necessary for agents to be capable of automatically verifying, at run-time, dynamic protocols affected by dynamic deontic rules. 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

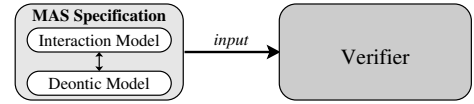


Figure 2: Design plan

during interaction time.

The Model Checking Problem. 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 . In our case, the system model is a combination of interaction and deontic models. Figure 3 illustrates our model checking process.

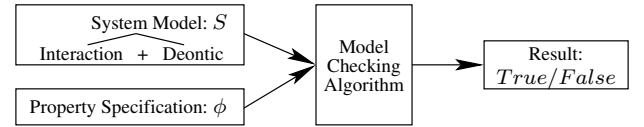


Figure 3: The model checking process

We choose the lightweight coordination calculus (LCC) [4] for specifying our interaction model, some policy language for specifying the deontic model, and the μ -calculus temporal logic for property specifications. The model checking technique implemented is a logic-based local model checking technique. The LCC language provides us with an executable specification that allows interaction models to be portable, and hence shared and global. This also supports dynamic model checking, where agents can automatically retrieve the protocol state and feed it to the model checker for verification. LCC also links cleanly to traditional process calculi used in model checking. The combination of μ -calculus and a logic-based local model checking technique results in an extremely compact size model checker, which actually throws the burden of the state-space search on the underlying tabled Prolog system. While [2] explores these issues in detail, this paper presents a higher-level description of the dynamic model checker irrespective of the specification details.

4. DYNAMIC MODEL CHECKING

At interaction time — when the conditions for verification are met and the related deontic models are obtained — the agent, interested in the satisfaction of property ϕ with respect to a given scenario, feeds the lightweight model checker with the following:

- the system model S (e.g. Figure 1), which consists of:
 - a global interaction model
 - a set of local deontic models
- the property ϕ to be verified

4.1 Example revisited

Figure 4 provides a section of the file that is fed to the model checker by the broker agent for verifying the travel agency scenario it is about to instantiate. The interaction model is a shared global model that specifies the rules of the interaction. The interaction model section presented in Figure 4 deals with a section of the interaction between the travel agent and the customer agent. More specifically, it shows that when the customer agent sends its chosen flight, the travel agent reacts by querying the hotel directory and sending a list of available hotels for the customer to choose from. The interaction model, written in LCC, is essentially a specification of the interaction's state-space graph.

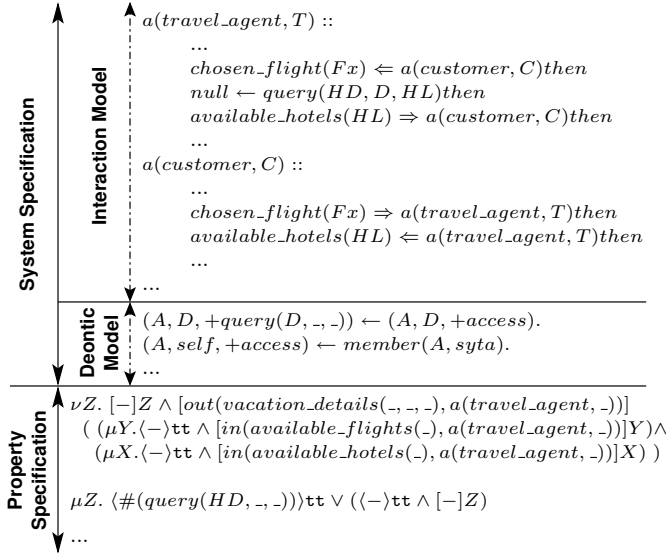


Figure 4: System model fed to the model checker

The deontic rules are constraints on the various agents engaged in the interaction. Unlike the interaction model, these do not impose constraints on the message passing action, but on any other action an agent can take. The first rule in the sample presented by Figure 4 specifies that an agent can query a directory only if it has access to it. This rule would belong to the broker agent who is interested in picking the right travel agent. The second rule specifies that an agent might access the directory only if it was a member of the student youth travel association (*stya*). This rule would belong to the hotel directory for specifying the appropriate authorisations.

Finally, the system model is to be verified against a given set of properties. The 'property specification' section of Figure 4 provides a sample of the properties that could be verified for our travel agency scenario. The first specifies that if the customer agent requests a vacation package, it will always eventually receive a list of flights and hotels. The second property specifies that in every run of the interaction, it is always the case that the hotel directory is eventually queried.

Now when the broker needs to verify that the interaction protocol will work properly for the selected set of agents, it will feed the file presented by Figure 4 to the dynamic model checker. The verification process is outlined below.

4.2 The verification process

The interaction model, specified with the LCC process calculus, is essentially a specification of the interaction's state-space graph. While global model checking is based on generating the whole state-space, local model checking partially constructs the state-space one step at a time until a solution is reached. Verification, in the case of our local model checker, starts at the initial state s_0 and tries to verify that the property ϕ is satisfied at s_0 ¹. 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-space². If the transition(s) violates any of the 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.

This method raises two questions: (1) how do we ensure termination? and (2) how do we ensure correctness? Inspired by the XMC model checker [3], we build our model checker on top of XSB [5], a tabled Prolog system. Caching in tabled Prolog ensures termination, avoids redundant subcomputations, and computes the well-founded model of normal logic programs. Our model checker, unlike XMC, accepts LCC interaction models in parallel with deontic models.

5. CONCLUSION

This paper presents a verification technique for predicting and preventing failure — due to errors within the interaction protocol, conflicts between the interaction protocol and agents' rules and requirements, as well as clash of interests between agents — as early as possible in MAS scenarios. This is made possible by allowing the agents to invoke the model checker at runtime, when the conditions for verification are met, and verify that certain properties hold for the given combination of a shared interaction model and agents' local deontic models.

6. REFERENCES

- [1] H. He, H. Haas, and D. Orchard. Web services architecture usage scenarios. *W3C Working Group Note*, Feb 2003. <http://www.w3.org/TR/2004/NOTE-ws-arch-scenarios-20040211/>.
- [2] N. Osman, D. Robertson, and C. 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.
- [3] C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, Y. Dong, X. Du, A. Roychoudhury, and V. N. Venkatakrishnan. XMC: a logic-programming-based verification toolset. In *Computer Aided Verification*, Lecture Notes in Computer Science, 2000.
- [4] D. Robertson. A lightweight coordination calculus for agent social norms. In *Proceedings of Declarative Agent Languages and Technologies workshop*, 2004.
- [5] K. Sagonas, T. Swift, and D. 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.

¹Satisfaction is verified by applying the modal μ -calculus proof rules presented in [2].

²Transitions are made based on the LCC transition rules presented in [2].