MODEL CHECKING KNOWLEDGE AND COMMITMENTS IN MULTI-AGENT SYSTEMS USING ACTORS AND UPPAAL

Christian Nigro, Libero Nigro, Paolo F. Sciammarella

Software Engineering Laboratory
University of Calabria, DIMES - 87036 Rende (CS) – Italy
Email: christian.nigro21@gmail.com, l.nigro@unical.it, p.sciammarella@dimes.unical.it

KEYWORDS
Multi-agent systems, knowledge and commitments, NetBill protocol, actors, model checking, UPPAAL.

ABSTRACT

This paper proposes a method for modelling and analysis of knowledge and commitments in multi-agent systems. The approach is based on an actors model and its reduction onto UPPAAL. A key factor of the approach is the possibility of exploiting the same UPPAAL model for exhaustive verification or, when state explosion problems forbid model checking, for quantitative evaluation of system properties through statistical model checking. The article describes the method, shows its application to modelling the NetBill protocol, proposes a translation into the terms of the timed automata language of UPPAAL and demonstrates the analysis of the NetBill protocol together with some experimental results.

INTRODUCTION

Nowadays more and more distributed software systems are developed using the multi-agent systems (MAS) paradigm (Wooldridge, 2009). MAS enable the construction of highly modular and scalable systems, where the resultant behavior at the population level (macroscopic or society level) emerges (in a non-intuitive way) from the individual behavior of agents and their interactions (microscopic level). Fundamental abilities of agents contributing to the expressive power of MAS include autonomy, goal-directed mission, sociality (i.e., communications), pro-activity and reactivity.

As the applications of MAS continue to grow the need arises to support modelling and verification of an agent-based system in order to ensure correctness of a development.

This paper focuses on modelling and analysis of knowledge and commitments in MAS (Al-Saqqar et al., 2015)(Singh, 2000) which naturally occur, e.g., in business web-based protocols and applications. Commitments express the willingness for agents to do something (e.g., paying for an accepted offer of a good over the Internet). Commitments have a status which can switch from creation, to fulfillment, to discharge, to cancellation. Knowledge refers to the epistemic relation of agent awareness about the status of a commitment.

Reasoning on knowledge and commitments in MAS is often pursued through a particular temporal logic language like CTLKC$^+$ (Al-Saqqar et al., 2015)(Sultan, 2015) whose models can be verified through reduction to an existing model checking tool. In (Al-Saqqar et al., 2015) the use of the NuSMV model checker (Lomuscio et al., 2007) is demonstrated with the overall approach which favours model scalability.

The work described in this paper was triggered by the CTLKC$^+$ work, and the desire to offer timed automata modelling and particularly to simplifying the query language in order for it to become more easy to use and understand by final modellers. In the proposed approach agents are modelled by actors (Cicirelli & Nigro, 2016)(Nigro & Sciammarella, 2017a) and the analysis activities are carried out in the popular UPPAAL toolbox (Behrmann et al., 2004)(David et al., 2015). The presented reduction of actors to UPPAAL is novel and extends previous authors' work (Nigro & Sciammarella, 2017a-b)(Nigro et al., 2018). The new contribution refers to the fact that a reduced actor model in UPPAAL now enables both exhaustive model checking (through the construction of the model state-graph) and statistical model checking (based on simulations) which can be necessary when exhaustive verification is forbidden by state explosion problems.

The paper is structured as follows. First some background is provided concerning basic concepts of knowledge and commitments in MAS and of the adopted actors. Then the NetBill protocol (Sirbu, 1997)(Al-Saqqar et al., 2015) is modelled using actors. After that, the developed reduction of actors onto UPPAAL is detailed through the transformation of the NetBill model. Then some experimental work is reported using the achieved UPPAAL model. Finally, conclusions are presented with an indication of further work.

BACKGROUND CONCEPTS

Commitment issues

Knowledge and commitments (to do something) in MAS are issues that in the literature are often handled independently. This is unfortunate (Al-Saqqar et al., 2015) because, as occurs in business settings, agents need to reason on their social commitments *and* their knowledge, especially when they are engaged in conversations.

The following example, quoted from (Al-Saqqar et al., 2015), testifies the importance of expressing the interactions between knowledge and commitments:

"*Suppose that we asked a member from our team to buy a book for us last month. He made the online order and committed to pay. The credit card debit succeeded,*

Proceedings 32nd European Conference on Modelling and Simulation ©ECMS Lars Nolle, Alexandra Burger, Christoph Tholen, Jens Werner, Jens Wellhausen (Editors)
ISBN: 978-0-9932440-6-3/ ISBN: 978-0-9932440-7-0 (CD)

meaning that the agent (our team member) knows that he fulfilled his commitment to pay. The publisher company committed to send the requested book to our address. Unfortunately, the book has never arrived. The publisher claimed they had send it out, but the shipping company they dealt with could not find it in their records. As a result, we asked them to send it again. However, knowing that the book is delivered (i.e., fulfilling the commitment of delivering the book) will help avoiding such situations."

Commitments are naturally tied to communications (Singh, 2000)(Sultan, 2015). Two main approaches can be distinguished. In the *mentalistic* approach communications are in terms of privately defined information of agents, that is agent mental states as beliefs, desires and intentions. In such approaches the assumption is made that "*each agent can read each other mind*". However (Sultan, 2015), mental based models do not favour verification. A different approach considers communications as based on publicly available information. In a social commitment, a *debtor* agent (sender) agrees with a *creditor* (receiver) agent about a debtor engagement to bringing a certain property. Social commitments provide an objective semantics to messages and naturally accommodate for agent heterogeneity.

A crucial issue when modelling social commitments is agent *uncertainty*. Uncertainty, that is non-deterministic behavior, concretely affects agent evolution and makes it challenging to analyze agent behavior. Non determinism specializes into probabilistic behavior when agent commitments are studied quantitatively. In this case, non deterministic courses of actions in the agents can be labelled by chosen probability values.

In (Al-Saqqar et al., 2015)(Sultan, 2015), social commitments between a debtor and a creditor are associated with the existence of *shared variables* of the debtor and creditor agents. Shared variables, in turn, mean suitable communication channels among the agents exist through which the value of a shared variable is updated following a social commitment operation.

Formally, in CTLCK$^+$, a social commitment is denoted by $C_{i\to j}\ \varphi$ whose meaning is: agent i (the debtor) commits toward agent j (the creditor) about φ which is the content of the commitment. Of course, during its lifecyle, a commitment has to be known to the involved agents. The relation $K_i\ \varphi$ formally represents the fact that agent i knows φ in some state. A commitment status should be known to both its involved agents as they proceed in a conversation.

In (Al-Saqqar et al., 2015) the CTLKC$^+$ temporal logic is mapped on an action-based logic (ARCTL) and then on to the modelling language of the NuSMV symbolic model checker (Lomuscio et al., 2007), which is based on Binary Decision Diagrams (BDD) and Boolean functions (Bf). The overall approach is proved to be efficient in space and time and favours model scalability.

The analysis of a probabilistic version of CTLKC$^+$ is described in (Sultan, 2015) along with a reduction to PRISM modelling language and toolbox (Hinton et al., 2006).

Actor issues

The contribution of this paper is to propose a different approach to modelling and verification of knowledge and social commitments in MAS, which rests on a lightweight actor model which can effectively be reduced to UPPAAL timed automata (Alur & Dill, 1994) for model checking. When the model size forbids exhaustive verification by state explosion problems, the UPPAAL model can be analyzed quantitatively, by inferring probability measures about required properties by using the UPPAAL Statistical Model Checker (SMC).

Adopted actors (see (Cicirelli & Nigro, 2016)(Nigro & Sciammarella, 2017a-b)(Nigro et al., 2018) for more information) encapsulate a *data status* and publish a particular *message interface*. Message-passing is asynchronous. Only through a received message, an actor can update its local variables. Actors are thread-less agents: they are at rest until a message arrives. Messages can be sent to known actors (*acquaintances*). For pro-activity, behavior an agent can also send a message to itself.

The programming/modelling style of actors is easily supported by Java. Each admitted message is processed in a *message-server* method (in Java the method is provided of the @Msgsrv annotation) having the same name, and which can have parameters. The basic send operation follows the syntax:

```
target_actor.send( message_name[, args] );
```

and exploits computational reflection for posting a corresponding (untimed) message object on an underlying message pending set (see below). A message can be timed by specifying an *after* clause (relative time) in the send operation:

```
target_actor.send( after, message_name[, args ] );
```

It is meant that the message has to be consigned to its target actor as soon as after time units are elapsed since the send time. When the after clause is missing, it evaluates to 0. Current absolute model time is returned by the now() primitive.

The message servers of an actor class can be programmed according to a finite state machine.

A collection of actors (*theatre*) runs on a computing node (JVM instance) and is (transparently) regulated by a *control machine* which governs the set of sent (timed and untimed) messages. The control machine is responsible of managing a time notion and follows a *control loop*. At each iteration, a message is chosen from the pending set and dispatched to its destination actor. A library of control machines was developed (see (Cicirelli & Nigro, 2016)).

A theatre system consists in general of a collection of theatres which coordinate each other, e.g., to guarantee a common time notion.

A fundamental semantic aspect is the *macro-step semantics of messages*. In a same theatre, actors are

activated by messages one at a time. It is not possible to dispatch the next message until the last dispatched one was completely processed (the corresponding message-server method is terminated). This way, a control machine induces a cooperative concurrency schema among local actors which depends on message interleaving. Message servers of distinct theatres can be executed in true parallelism.

As a final remark, it is worth noting that a concrete implementation of the actor model can guarantee that messages sent by an actor A to an actor B, at the same time, will be received in the sending order. Such an order does not exist among messages sent by different senders toward a same receiver. When messages have different timestamps, their dispatch follows the timestamp order. However, during modelling and analysis, for generality concerns, message delivery at a same time will be assumed not deterministic.

ACTOR-BASED NETBILL PROTOCOL

The NetBill protocol (Sirbu, 1997)(Al-Saqqar,2015) is played by a customer agent (*cus*) and a merchant agent (*mer*). The message conversation relates to buying and selling an encrypted software good over the Internet.

In the following, the NetBill protocol is modelled using actors by adapting the customer and merchant models reported in (Al-Saqqar et al., 2015). Two commitments are involved in the protocol: $C_{cus \to mer}\ Pay$ and $C_{mer \to cus}\ Deliver$. The Pay commitment is raised by the customer toward the merchant to testify the customer intention, after having accepted the offer from the merchant, to proceed with the payment of the good. The $Deliver$ commitment is played by the merchant toward the customer, to express its willingness to deliver to it the required (and paid!) good. However, uncertainty in the agent behaviors can change the course of expected actions following the initial intention to commit, as shown in the models of customer and merchant in the Figg. 1 and 2.

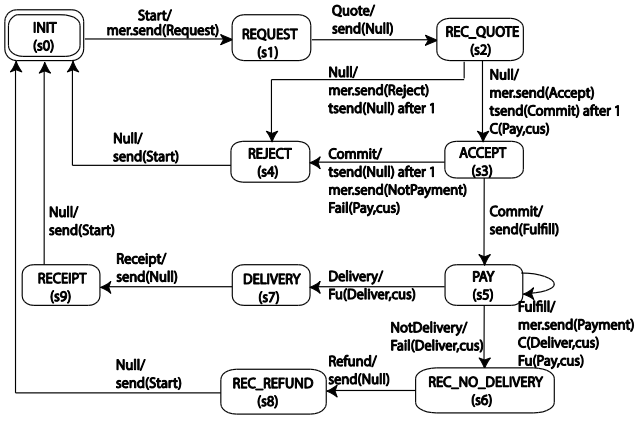

Fig. 1 - The Customer model

The customer starts by sending a request for a quote to the merchant. After receiving the quote the customer can, non deterministically, proceed by accepting the quote (state s3) or rejecting it (state s4) after which the model is restarted from the initial state (s0).

In the case the quote is accepted, the customer sends a message Accept to the merchant and a message Commit to itself, then it moves to state s3. Accepting the quote logically issues the Pay commitment.

In the work described in (Al-Saqqar et al., 2015) a social commitment implies a communication between the customer and the merchant in order to reflect the status of the commitment in the shared variables of the two agents. Such a communication can be naturally captured by an explicit message exchange in our actor modelling. However, the Accept message itself serves the purpose of transmitting to the merchant the customer intention to (possibly) proceed with the payment, then it plays also the role of communicating that the Pay commitment was issued. In addition, the customer in the state s2 sends to itself (pro-activity) the Commit message as a *reminder* to proceed with payment. On receiving the Commit message, the customer can, again non deterministically, choose to fulfil the Pay commitment or to send a NotPayment message to the merchant thus putting the Pay commitment into the failure state.

An important aspect of the actor model is concerned with message delivery which, as anticipated in the previous section, is supposed to be non deterministic. For example, the sequence of untimed messages Accept then NotPayment, without other provisions, can be actually received by the merchant in the order first NotPayment then Accept, thus incurring into a malfunction. To ensure the right order, the Commit message is sent as a timed message with a delay of 1 time unit. In the case the customer decides to proceed with the Pay fulfillment (state s5), it sends to itself the Fulfill message whose reception causes the customer to effectively send a Payment message to the merchant.

The status of a commitment c is assumed to be changed by the functions C(c,req), Fu(…) and Fail(…) which respectively set it, for the requestor agent req, to WillingToDo, Fulfilled and Invalid. Knowledge about commitments will be checked during model analysis.

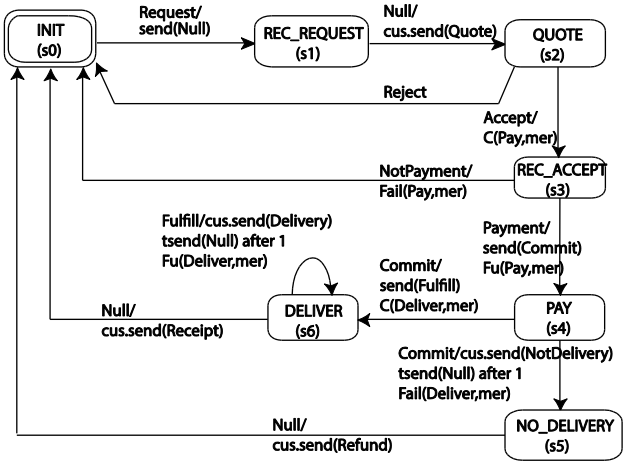

Fig. 2 - The Merchant model

By reciprocity, when the customer receives a Delivery message from the merchant, it calls Fu(Deliver,cus) to update its information about the *Deliver* commitment. Similarly, it calls Fail(Deliver,cus) when a NotDelivery is received, in which case a Refund is expected from the merchant. The Null message in the Fig. 1 is used pro-actively to ensure a proper state sequencing.

The merchant model in Fig. 2 follows the same design guidelines seen in Fig. 1. For example, on receiving an Accept message, it calls C(Pay,mer) to optimistically reflect the customer intention to proceed with payment. Similarly, the arrival of a Payment message in state s3, requires the merchant to update its status of the *Pay* commitment to Fulfilled. Other details should be self-explanatory.

A REDUCTION OF ACTORS TO UPPAAL

The rationale of proposed reduction of actors to UPPAAL is to express both actors and messages as template timed automata. The challenging is to correctly reproduce the asynchronous character of messages. Although UPPAAL SMC supports dynamic automata (Nigro & Sciammarella, 2017a-b), in this paper a certain number of statically created message instances are instead used and dynamically activated and then, after being dispatched, reset for them to be re-used. All of this opens to the possibility of making exhaustive model checking, provided the right number of message instances is prepared. The same model can also be used by simulations using the Statistical Model Checker of UPPAAL (David et al., 2015).

First, actors have to be uniquely identified:

```
const int CUS=2; //number of customers
const int MER=2; //number of merchants
const int N=CUS+MER; //number of total agents
typedef int[0,CUS-1] cus_id; //customer ids
typedef int[CUS,N-1] mer_id; //merchant ids
typedef int[0,N-1] aid; //type of agent ids
```

For simplicity, customers and merchants are supposed to operate in pairs. A customer id, e.g.,$cus \in [0, CUS-1]$, is paired with the merchant id: $mer = cus + CUS$.

Similarly, exchanged messages are classified:

```
const int Null=0, Start=1, Quote=2, Delivery=3, Receipt=4,
NotDelivery=5, Refund=6, Request=7, Reject=8, Accept=9,
Payment=10, NotPayment=11, Commit=12, Fulfill=13;
const int MSG=14; //number of different messages
typedef int[0,MSG-1] msg_id; //type of message ids
```

Then the array of broadcast channels msgsrv[][] is introduced which is used to dispatch a message to a given actor:

```
broadcast chan msgsrv[aid][msg_id];
```

Message automata ids corresponding to untimed and timed messages are dimensioned according to the needs of the models of customer and merchant (see Fig. 1 and Fig. 2):

```
const int UM=CUS; //number of untimed messages
typedef int[0,UM-1] umid;
const int TM=CUS; //number of timed messages
typedef int[0,TM-1] tmid;
bool freeUM[UM], freeTM[TM]; //initialized to all true
```

The following channel arrays serve to activate message instances:

```
broadcast chan send[umid][aid][msg_id],
               tsend[tmid][aid][msg_id];
```

Functions nM() and nTM() respectively return the id of the next free untimed or timed message instance. Obviously, when freeUM[.] or freeTM[.] are insufficiently dimensioned, an error will occur during model checking.

Fig. 3 and Fig. 4 show the automata for an untimed and a timed message. A sent message is first scheduled then dispatched.

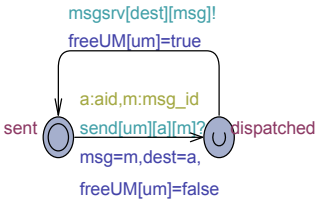

Fig. 3 - Message automaton

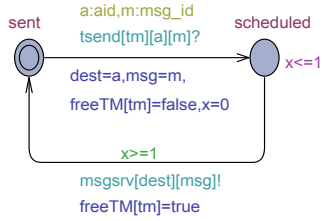

Fig. 4 - TimedMessage autom.

Message/TimedMessage automata have respectively the only parameter: const umid um, or const tmid tm.

The following global declarations introduce commitments and knowledge support:

```
const int PAY=0, DELIVER=1; //admitted commitments
typedef int[PAY,DELIVER] cid; //commitment ids
const int Invalid=0, WillingToDo=1, Fulfilled=2;
typedef int[Invalid,Fulfilled] status;
status k[cid][aid]; //agent knowledge of comm. states
```

Commitment states in agent knowledge are updated by the functions C(cid,aid), Fu(…) and Fail(…) (not shown for brevity) which receive the commitment id and the requestor agent. Function K(aid,cid) returns the status of a commitment as known to a given agent.

The Customer automaton shown in Fig. 5 has the only parameter: const cus_id cus. The associated mer id is implicitly estabilished as a local constant at initial time.

In the Receive location, the next arrived message is received through one msgsrv channel. In the Select location, the identity of the received message is checked and the corresponding response path selected, after which the Receive location is re-entered. It is worth noting that a processing path for responding to a message is achieved, in general, by committed locations, which have greater priority than urgent locations (see e.g. Fig. 3). All of this is a key to ensure the macro-step semantics of message processing. For the rest, the UPPAAL automaton in Fig. 5 closely corresponds to its design in Fig. 1. Fig. 6 shows the Merchant automaton.

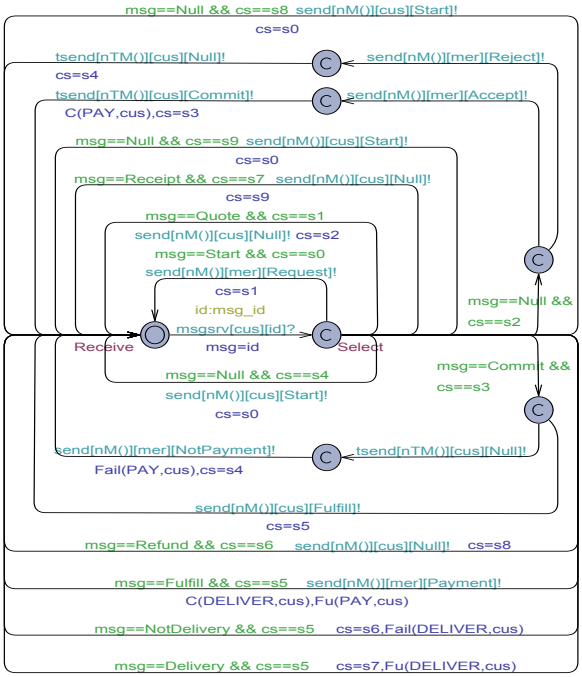

Fig. 5 - The Customer automaton

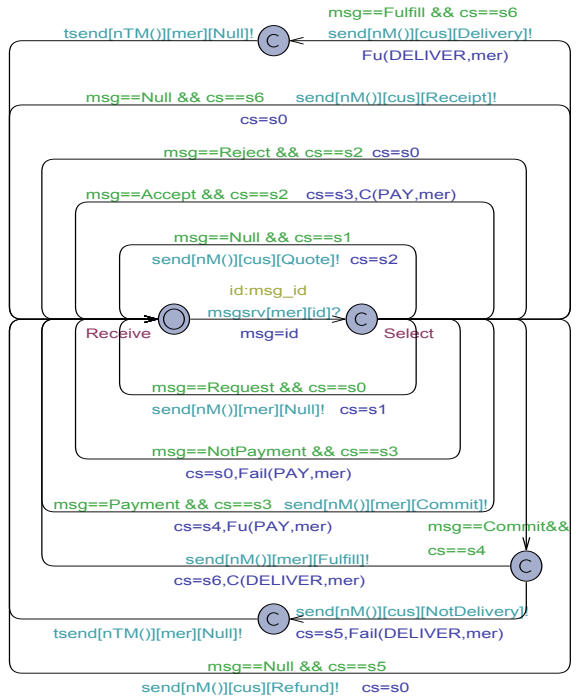

Fig. 6 - The Merchant automaton

The reduced NetBill protocol model is put into operation by a Main automaton (see Fig. 7) which initializes global data structures and sends a Start message to all the customers.

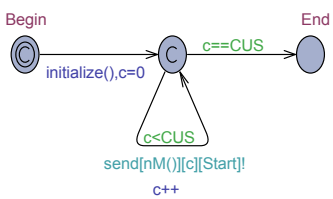

Fig. 7 - The bootstrapper Main automaton

Finally, the following is the system configuration line:

```
system Main,Customer,Merchant,Message,TimedMessage;
```

MODEL CHECKING THE NETBILL PROTOCOL

Both *safety* (a bad state is never reached) and *liveness* (a good state is eventually reached) properties were checked on the reduced actor-based NetBill protocol in UPPAAL.

Exhaustive model checking is centred on the TCTL-based query language offered by UPPAAL (Behrmann et al., 2004) which does not allow formulas to be nested, as instead is permited in CTL-like logics (see, e.g., (Al-Saqqar et al., 2015) and NuSMV queries). All of this contribute to simplicity and readability, without compromising correctness checking.

Qualitative analysis by model checking

Up to 4 agents (two pairs <cus,mer>) were easily model checked (also on a laptot with 4GB RAM). A greater number of agents causes state explosions. However, correct protocol behavior can be verified also on a single pair <cus,mer>.

Safety properties

The query

```
A[] ! deadlock
```

was found satisfied, thus guaranteeing there are no deadlock states in the model.

Another safety property is concerned with the situation: the customer fulfils its payment commitment, i.e., it sends the payment to merchant, but the merchant does not know that. In the following, the cus id 0 is considered, which works with mer id CUS. One way for checking the property is through the *leads-to* operator --> (cs is a local variable holding the current state of the customer):

```
Customer(0).cs==s5 -->
    K(0,PAY)==Fulfilled && K(CUS,PAY)==Fulfilled
```

which means: when the customer is in state s5, it inevitably will be followed by a successor state in which both the customer and the merchant know that the PAY commitment is fulfilled. The query was found satisfied. A similar query (satisfied) was prepared for checking the fulfillment of the DELIVER commitment (from mer to cus):

```
Merchant(CUS).cs==s6 --> K(CUS,DELIVER)==Fulfilled
    && K(0,DELIVER)==Fulfilled
```

Liveness properties

One such a property is the following: in the case the merchant commits to deliver the goods to the customer, it will eventually deliver them. The property was checked with the query:

```
K(CUS,DELIVER)==WillingToDo -->
    K(CUS,DELIVER)==Fulfilled
```

or the query:

```
Merchant(CUS).cs==s6 --> K(CUS,DELIVER)==Fulfilled
```

which are both satisfied. The following queries check that for both the customer and the merchant, it inevitably follows that the initial state s0 will be re-entered (now is a decoration clock measuring the global model time):

```
A<> now>0 && Customer(0).cs==s0
A<> now>0 && Merchant(CUS).cs==s0
```

Both queries are satisfied. As another example, would the customer or the merchant be in the state s1, it will definitely move to the initial state (queries are satisfied):

```
Customer(0).cs==s1 --> Customer(0).cs==s0
Merchant(CUS).cs==s1 --> Merchant(CUS).cs==s0
```

Quantitative analysis by statistical model checking

An important benefit of the developed reduction of actors to UPPAAL, is concerned with the possibility of exploiting the statistical model checker (SMC) for quantitative property checking. This, in turn, will help in the cases where state explosion problems do not allow for an exhaustive verification.

UPPAAL SMC (David et al., 2015)(Nigro et al., 2018) does not build the model state-graph. Instead it is based on simulations which imply a linear demand of memory. SMC predicts a number of simulation runs and infer from them (using Monte Carlo-like techniques or sequential hypothesis testing) a probability measure for a checked property. It comes equipped with a powerful query language for statistical analysis which is based on the Metric Interval Temporal Logic (MITL).

During the analysis using UPPAAL SMC of the NetBill protocol, non deterministic behavior is turned into a probabilistic one. For example, in state s3 of Fig. 1 where che customer can either choose to fulfil the payment toward the merchant, or send it the NotPayment message, the two alternatives have an equal probability to occur during SMC analysis. Would it be useful for the modelled system, the modeler can also label the two transitions with a probability weight. In the following, uncertainty by non determinism is implicitly replaced by probabilities without introducing specific probability weights. In addition, default statistical options of the toolbox are used (e.g., the uncertainty error of a confidence interval is $\varepsilon = 0.05$).

Functional behavior of the NetBill model was preliminarily checked with such queries like the following:

```
simulate 1 [<=10] { Merchant(CUS).cs }
```

which monitors, in 10 time units, the evolution of the local state variable cs of the Merchant model. An observed behavior is reported in Fig. 8 which confirm cs always regularly returns to the s0=0 value.

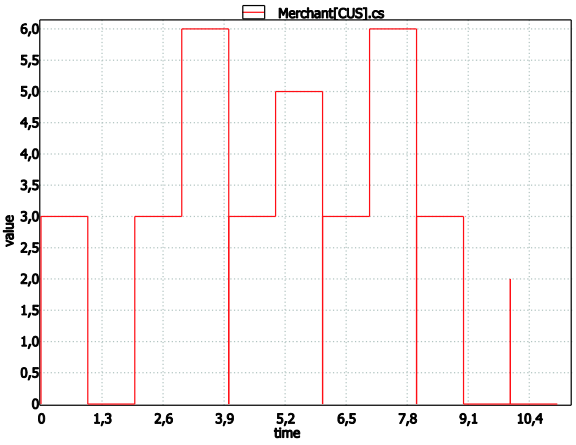

Fig. 8 - Evolution of Merchant(CUS).cs variable vs. time

In a different way, the liveness property, e.g., of Customer(0), that its local state variable cs will eventually assume again the s0 value, the following query estimates the probability of the event: "being the customer in a state different from s0 at an instant in [0,100], in at most 2 time units it will come back to its home state s0":

```
Pr(<>[0,100] (Customer(0).cs!=s0 &&
    (<>[0,2] Customer(0).cs==s0) ))
```

Uppaal SMC, using 738 runs, suggests a confidence interval of [0.95,1] with a confidence degree of 95%, to indicate that the event has a very high probability of occurrence.

The following query asks about the probability that, at any instant in [0,100], would the customer know that the PAY commitment is fulfilled, in 0 time the merchant will know it too.

```
Pr(<>[0,100] (K(0,PAY)==Fulfilled &&
    (<>[0,0] K(CUS,PAY)==Fulfilled) ))
```

Also in this case the probability has a confidence interval of [0.95,1] with confidence 95%.

The query:

```
Pr[<=1000] (<>Customer(0).cs==s5 &&
    K(0,PAY)==Fulfilled)
```

asks to estimate the probability that when the customer is in the state s5, it knows the PAY commitment is fulfilled. Using 36 runs, Uppaal SMC says the probability has a confidence interval of [0.902606,1] with confidence 95%. The same event was monitored (see Fig. 9) with the query:

```
simulate 1 [<=100] {Customer(0).cs==s5 &&
    K(0,PAY)==Fulfilled}
```

Obviously, as one can see from Fig. 9, not always the monitored condition is true because there are cases when the customer from s3 goes to state s4 rejecting the offer, instead to switching to s5 where it is guaranteed that the payment will be honored.

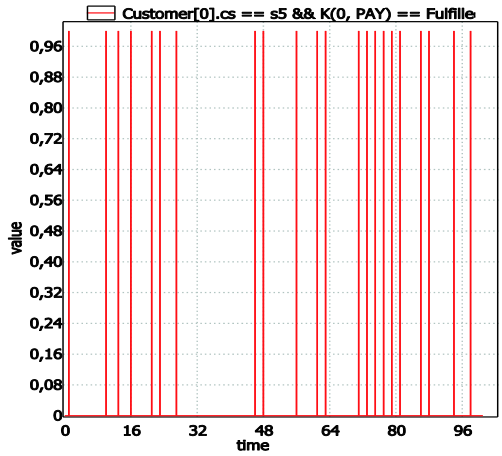

Fig. 9 – "Customer(0) in s5 knows PAY is fulfilled?" vs time

In the light of the reported qualitative/quantitative experimental results, the NetBill protocol was found to be correct.

Experiments were carried out on a Linux machine, Intel Xeon CPU E5-1603@2.80GHz, 32GB, using UPPAAL 4.1.19 64bit.

CONCLUSIONS

This paper proposes an approach to modelling and verification of knowledge and commitments in multi-agent systems (MAS) intended for business web-based protocols and applications.

The method is novel and it is based on lightweight actors (Cicirelli & Nigro, 2016)(Nigro & Sciammarella, 2017a-b)(Nigro et al., 2018) together with a reduction on top of UPPAAL timed automata (Behrmann et al., 2004)(David et al., 2015). This way a MAS model can be studied by exhaustive verification and/or through the statistical model checker which rests on simulations.

With respect to the inspiring work of (Al-Saqqar et al., 2015)(Sultan, 2015), the possibility of switching from model checking to statistical model checking on a same model, possibly enhanced by probabilistic weights at branch points, is a key contribution. In addition, the use of simpler and more readable queries than those CTL-like with nested formulas, should be pointed out.

Another benefit of the proposed approach stems from the adoption of an actor modelling language which has a clear link to implementation. Actually, the used actors are embedded in Java and can effectively be exploited for programming time-dependent distributed systems.

Prosecution of the research aims to:

- improving the proposed reduction of actors on to UPPAAL;
- continuing experimenting with modelling general MAS and analysing their properties;
- specializing the approach to modelling and qualitative/quantitative analysis of distributed, possibly probabilistic, timed actors (preliminary results are described in (Nigro & Sciammarella, 2017a)).

REFERENCES

Alur, R., Dill, D.L. (1994). A theory of timed automata. *Theoretical Computer Science,* **126**:183–235.

Al-Saqqar, F., Bentahar, J., Sultan, K., Wan, W., Asl, E.K. (2015). Model checking temporal knowledge and commitments in multi-agent systems using reduction. *Simulation Modeling Practice and Theory,* **51**, 45-68.

Behrmann, G., A. David, K.G. Larsen (2004). A tutorial on UPPAAL. In: *Formal Methods for the Design of Real-Time Systems*, Lecture Notes in Computer Science, Vol. 3185, Springer-Verlag, pp. 200-236.

Cicirelli, F., L. Nigro (2016). Control centric framework for model continuity in time-dependent multi-agent systems. *Concurrency and Computation: Practice and Experience*, **28**(12):3333-3356, Wiley.

David, A., K.G. Larsen, A. Legay, M. Mikucionis, D.B. Poulsen (2015). UPPAAL SMS Tutorial. *Int. J. on Software Tools for Technology Transfer*, Springer, **17**:1-19, 06.01.2015, DOI 10.1007/s10009-014-0361-y.

Hinton, A., Kwiatkowska, M.Z., Norman, G. and Parker, D. (2006). PRISM: a tool for automatic verification of probabilistic systems. In *Proc. of 12th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, Springer-Verlag LNCS Vol. 3920, pp.441–444.

Lomuscio, A., Pecheur, C., Raimondi, F. (2007). Automatic verification of knowledge and time with NuSMV. In *Proc. of the 20th International Joint Conference on Artifical Intelligence, IJCAI'07*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 1384–1389.

Nigro, L., Sciammarella, P.F. (2017a). Statistical model checking of distributed real-time actor systems. In *Proc. of 21st IEEE/ACM Int. Symposium on Distributed Simulation and Real-Time Applications (DS-RT'17),* October 18-20.

Nigro, L., Sciammarella, P.F. (2017b). Modelling and analysis of distributed asynchronous actor systems using Theatre. *Advances in Intelligent Systems and Computing 661*, DOI 10.1007/978-3-319-67618-0_14, Springer.

Nigro, C., Nigro, L., Sciammarella, P.F. (2018). Modelling and analysis of multi-agent systems using Uppaal SMC. *Int. J. of Simulation and Process Modelling,* **13**(1):73-87.

Singh, M.P. (2000). A social semantics for agent communication languages. In *Issues in Agent Communication*, pages 31–45. Springer-Verlag.

Sirbu, M.A. (1997). Credits and debits on the Internet. *IEEE Spectr*. **34**(2):23-29.

Sultan, K.I. (2015). *Modeling and verifying probabilistic social commitments in multi-agent systems.* PhD thesis, Concordia University (CA), https://spectrum.library.concordia.ca/979616/

Wooldridge, M. (2009) *An Introduction to Multi-Agent Systems*, 2nd ed., John Wiley & Sons, Chichester, West Sussex, PO19 8SQ, UK.