A Layered Approach for Modelling Agent Conversations

Mariusz Nowostawski
Martin Purvis
Stephen Cranefield
Department of Information Science
University of Otago
PO Box 56, Dunedin, New Zealand
Phone: (64 3) 479 8318, Fax: (64 3) 479 8311
{mnowostawski,mpurvis,scranefield}@infoscience.otago.ac.nz

ABSTRACT

Although the notion of conversations has been discussed for some time as a way in which to provide an abstract representation of extended agent message exchange, there is still no consensus established concerning how to use these abstractions effectively. This paper describes a layered approach based on coloured Petri Nets that can be used for modelling complex, concurrent conversations among agents in a multi-agent system. The approach can be used both to define simple conversation protocols and to define more complex conversation protocols composed of a number of simpler conversations. With this method it is possible (a) to capture the concurrent characteristics of a conversation, (b) to capture the state of a conversation at runtime, and (c) to reuse conversation structures for the processing of multiple concurrent messages. A prototype implementation of such a system with some examples is described.

Keywords

Agent communication languages, conversations, conversation protocols, Petri Nets, conversation monitoring and visualising

1. INTRODUCTION

Speech act theory [1, 20] has been used as a fundamental mechanism for the modelling of, analysis, and design of agent communication, but it is still difficult for an agent that receives a message to understand the sender agent's intended meaning. Several authors [21, 8] have indicated that the basic problem lies in having agent communication specified in terms of the mental states of the sending agent, and that the way forward is to work on the specification of expanded external protocols that characterise agent conversations. However, beyond the initial proposal to use finite state machines, there has been little agreement on how such protocols should be specified and how they should be used in modelling and guiding agent conversations.

We might consider some of the features that a useful agent conversation formalism would be expected to have. For example, we might like such an approach to be able to

- define dialogue patterns (protocols)
- define conversation models
- capture different roles acting in an interaction
- capture the concurrent characteristics of a conversation
- capture the state of a complex conversation during runtime
- direct mapping to the conversation implementation
- graphical representation supporting visualisation, monitoring and debugging
- promote reuse of the conversation structures for independent concurrent instances of dialogues

In this paper we present a layered architecture that employs a formalism based on coloured Petri Nets that can be used for the modelling of complex, concurrent conversations agents. Although coloured Petri Nets have previously been proposed for agent conversation modelling [3, 4], the approach presented here involves some additional features that take advantage of some of the unique properties offered by coloured Petri Nets and thereby is able to provide most of the desired features listed above. A prototype implementation of a system supporting predefined conversation protocols as Petri Net models developed for the New Zealand Distributed Information Systems research platform is briefly described at the end of the paper.

We employ a three-layer architecture for modelling conversations, which expands on previous approaches that employs two abstractions of *protocols* and *conversations*, by adding an additional notion, that of *policy* (or strategy). The policy layer guides the participating agents during the course of a conversation and can be used to deal with conversational components that are directed to be *about* the current conversation in progress or that can serve to reroute the current conversation in a new direction. This new additional level helps to keep conversation-specific logic close to the conversation models, which improves encapsulation and helps conversation debugging and the verification process.

In Section 2 we briefly introduce formalisms currently being used for conversation modelling, and in Section 3 we propose a modelling architecture and terminology to describe modelling entities.

In Section 4 we introduce the concepts of coloured Petri Nets, which we further expand in Section 5 to present a modelling framework based on coloured Petri Nets. In Section 6 some simple and more complex examples are presented, and finally in Section 7 a brief summary and comparison with other formalisms is presented.

2. RELATED WORK

There is reasonably extensive work being done in the field of agent conversation modelling. The suitability of Petri Nets as a compact and uniform model has been already proposed [4, **?**]. For reference and comparison we have chosen some other traditional solutions based on the Deterministic Finite Automata (DFAs) [2,24,14], Enhanced Dooley Graphs [18], and extended UML [16, 17] formalisms.

Deterministic Finite Automata represent the simplest and most straightforward modelling formalism used for conversations. They are suitable for specifying all the states the conversation must go through, and can be used to check the validity of the designed model. This approach however suffers from two major shortcomings: it does not cope well with representing concurrency, and it fails to associate clearly the message information with the individual participants in the conversation. Enhanced Dooley Graphs modelling is a technique based on extensions to Dooley graphs [6] proposed in [18]. It provides not only state information but also information about the individual participants of the conversation, which is a considerable improvement over purely DFA-based solutions. Recently a number of possible approaches based on the Unified Modelling Language (UML) [15] have been investigated, and a number of possible modelling formalisms have been proposed [16, 17]. In contrast with a purely graph-based solution, they consist of a number of interleaving and complimentary techniques and modelling formalisms, based on state charts, sequence diagrams, and activity diagrams.

3. AGENT INTERACTIONS MODELLING

Modelling agent interactions is not a trivial task. It is not enough to have an appropriate modelling formalism with a graphical representation and formal semantics supporting the analysis process. It is important also to identify key aspects of the agent interactions and to model the interactions on different levels of abstraction and different levels of details. To improve the modelling process and to help make the discussion that follows more clear we have defined (or redefined) three fundamental terms used when discussing agent interactions:

protocol (interaction or conversation protocol) - the template of the communicative acts sequence

conversation - an instance of a conversation, a particular sequence of communicative acts

policy (interaction or conversation policy) - a strategy, guidelines and constraints guiding a conversation.

We will explain those terms in more detail in the following paragraphs[1].

[1]Please note that the terminology used by us does not necessary match the terminology from other publications in the field, where protocol, policy and conversation are very often used interchangeably. In particular the notion of conversation policy from [14] is equivalent to conversation protocol in our terminology.

Firstly the most important term is a **communicative act**. A **communicative act** is a special action type in the speech act theory [1, 20]. It represents a basic building block of the dialogue between agents, and it has a well-defined semantics independent of the content of the action. There is ongoing work to define standard collections of communicative acts, and there are currently two major widely accepted standards, namely Foundation for Intelligent Physical Agents (FIPA) [11] agent communication language (ACL) [10] and Knowledge Query and Manipulation Language (KQML) [9]. Both define a set of communicative acts, together with the outer, transport layer and the inner, content layer of messaging.

From our experiences in building multi-agent systems we have observed that agent conversation modelling can be decomposed into several separate layers. The first, most basic layer, is a *protocol layer*. A **conversation protocol** or **interaction protocol** or just a **protocol** is a template of sequences (in our case subnets) of expected communicative acts organised into roles. This definition is compatible with the definition of a protocol specified by FIPA as: "a common pattern of dialogues used to perform some generally useful task; the protocol is used to facilitate a simplification of the computational machinery needed to support a given dialogue task between agents; simply: a dialogue pattern" [10]. A protocol, apart from identifying roles and sequences of communicative acts, also specifies relations between **roles**. A **role** is an identity of a single sequence of acts executed by a single entity; it is denoted by an identifier, a name of such a sequence.

On top of that layer another layer is constructed: the *conversation layer*. A **conversation** is a particular instance of a protocol or set of protocols; it is an ongoing sequence of messages exchanged between two or more agents. Please note the distinction: a protocol is just a template or pattern, whereas a conversation is an instance of a given template or templates. It is possible, conceptually in a general sense, to have a conversation not be an instance of any particular template. This definition also complies with the one defined by FIPA: "an ongoing sequence of communicative acts exchanged between agents relating to some ongoing topic of discourse" [10]. However, for the sake of argument we have decided to constrain the term only to valid predefined templates and combinations (compositions) of templates, and to consider only conversations as instances of predefined agreed patterns. This excludes conversations being constructed from arbitrary chosen acts not conforming to a formal protocol.

The final, third layer is called the *policy* layer. This is the layer which would, with other approaches, be left to the agent application to coordinate and not be included explicitly in the conversation modelling process. However we feel that it is more appropriate to treat it as closely related to the conversation layer. A conversation policy is a collection of rules and interaction specifications that guide a particular path or trajectory in a conversation space. A *policy* defines the details concerning the conversation is handled by interested parties. Each *protocol* defines a space of possible sequences of communicative acts. Each *conversation* follows one trajectory from this space. A *policy* guides a particular conversation.

For example: imagine one protocol defining two roles, *buyer* and *seller*, and a sequence of acts for the buyer: ask (ask a seller for a particular goods delivery), then $accept$ (accept the price and buy goods) or $reject$ (reject goods, do not buy). And for a seller, the possible answers to the buyer ask action could be: $propose$ (pro-

pose the goods price) then $sell$ when accepted or do nothing when rejected. This simple specification is a *protocol*. Two participants have to follow a *protocol* to form a *conversation*. One possible *policy* (strategy) for the buyer would be to ask for goods from several other agents concurrently, and accept the lowest price and reject all the other proposals. That would dynamically create a relatively complex conversation involving several selling agents and a single buying one. A simpler strategy would be to ask only one selling agent, and $accept$ or $reject$ the proposal given by this agent, then start over a new conversation by issuing yet another ask to another potential seller for a proposal if the first iteration was finished without making a deal.

Policies may be implemented simply by set of rules, or, in more complex cases, they may have their own complex protocols that exist and change state in parallel with the immediate context of an ongoing conversation. Under these more complex circumstances, there might be a "policy-level interaction protocol" (another protocol, but at the policy level). It is under these conditions that we can benefit from having another modelling layer at the policy level, above that of the ordinary conversational modelling layer. The two layers can be joined together by representing them both as a coloured Petri Net (see Section 5).

Suppose, for example, we have a conventional conversation protocol involving two players playing a game of chess in a chess tournament. There are possible rules for legal moves and legal responding moves by the opposing players, which would be described by this conversation protocol. But existing above that level of abstraction is another level of discourse that can take place during the game. Suppose one of the players has a question concerning the official rules of the game and wants to have a ruling made by one of the tournament judges. Or suppose one of the players at some point wants to take time out from the game and halt play so that he or she can drink water or attend to some personal needs. These kinds of 'interrupt' or 'exception' are common to many kinds of interactions and can take place at almost any time. The discourse involved in these interrupts are usually "off-topic" from the context of the immediate conversation, and in fact they are often *about* the conversation that is taking place (such as the chess player who may accuse his opponent of breaking the conversation protocol rules associated with playing the game of chess). Since they are likely to be "off-topic" and can occur at any moment, it can be tedious to include these kinds of conversational strands in the given (domain-specific) conversation protocol. To do so would "clutter" the visual simplicity of the original conversation protocol and would lessen the value in providing a easy-to-comprehend visual modelling representation of the interaction. On the other hand, to leave out the possibility of representing such events is to ignore the possibility of their occurrence and consequently means that there is a failure to model the world adequately so that its essentially contingent nature is recognised. Our solution is to model these kinds of interactions that can guide, interrupt, or redirect existing conversations by representing them as another, parallel modelling layer above that of the existing conversation layer. This idea was suggested in [7] for specific types of converation, but we have generalised the notion and incorporated it into a Petri Net representation.

Thus a *conversation* is a combination of *protocols* being instantiated and manipulated by a particular *policy*. In Section 5 we will discuss how Petri Nets can be used on all those layers, to specify protocols, to monitor and analyse conversations, and how one can construct and use policies within conversations.

4. COLOURED PETRI NETS

Petri Nets provide an appropriate mathematical formalism for describing distributed, concurrent systems in the same way that finite-state automata are an appropriate tool for describing sequential systems. Petri Nets are ordinarily described in terms of a graphical representation (see for example Figure 7), but they have a formal mathematical description that is independent of any graphical representation. The basic (elementary) structure of a Petri Net can be formally defined by a 5-tuple (P,T,I,O,M0), where:

P – is a set of *places*,

T – is a set of *transitions*,

I and O – are the input and output functions, described as *arcs*, that map places to transitions and transitions to places, respectively; and

M0 – the *marking*, is a vector that characterises the initial state of the system by indicating the number of tokens in each place in the net.

In addition to the net structure, there are rules that describe how transitions fire in order to produce new states of the system. Although there are a number of transition firing rules associated with different types of Petri Net, they all share the common property of nondeterminism. If the Petri Net arcs have *weights* associated with them, then a Petri Net transition is enabled (and may fire) if:

- for every input place the number of tokens is greater than the weight associated with the connecting arc and
- every output place the sum of the number of tokens already existing in the place and the weight associated with the connecting arc is equal to or less than the capacity of the output place.

When a transition does fire:

- the tokens in each input place are reduced by a number equal to its input arc weight and
- the tokens in each output place are increased by a number equal to its output arc weight.

After a transition fires, the Petri Net has a new marking, characterised by a new distribution of tokens in the various places.

Coloured Petri Nets represent an elaboration of ordinary (*Place-Transition*, PT-net) Petri Nets and are so-named, because the initial extension to PT-nets involved the attempt to distinguish individual tokens of PT-nets by giving them colours [13]. Here, tokens can be designated to have any abstract data type that can exist in a programming language. Consequently CP-net tokens can have an arbitrary degree of complexity, if desired, and their types can be composites of other simpler types.

Informally, a coloured Petri Net has three basic components:

- a net structure of places, transitions, and arcs, which is like that of PT-nets

- ◇ a set of data declarations
- ◇ a set of net inscriptions (arc expressions, guards, and place initialisations)

The reader will find more detailed information in other publications [12, 19].

Petri Nets are in widespread use in many different aspects of software system design, analysis and implementation. The main reason for the great success of Petri Nets is their graphical representation, well-defined semantics and mathematical formalism allowing formal analysis and transformations [12]. Petri Nets are successfully being used in workflow modelling, and many useful workflow patterns have been developed [22, 23]. We believe that as much as Petri Nets are suited for modelling and simulating workflow processes, they can be used for modelling, simulating, analysing, monitoring and debugging conversations between agents in multi-agent systems. There is also substantial work being done in modelling protocols [5].

5. CONVERSATIONS

To be able to benefit from Petri Net modelling it is necessary to precisely define the semantics of different elements we are going to use. In this section we will define all the concepts necessary to design, analyse, deploy, monitor and debug protocols, conversations and policies.

Generally, tokens represent messages, arcs represent message passing and delivery mechanisms, and transitions represent message processing units. Roles are organized into subnets, and roles are separated by horizontal dashed lines. Arcs crossing role boundaries, i.e. arcs which cross dashed lines represent physical message passing actions (the process of sending and receiving a single message in the agent system). The arcs within roles are left up to the implementation and usually, for efficiency purposes, are implemented as method calls. This is how we have implemented it. Places represent message containers or intermediate containers, and usually do not map in the implementation to anything in particular, unless the Petri Net model is mapped directly to a Petri Net implementation, as in our case. Then a place is an abstraction of a message folder, containing processed or being processed messages.

There is always one initiator of a conversation, a role which starts the conversation by issuing the very first message, and this role (and only this role) always has the $Start$ place, which enables the very first transition to fire. All roles have separate dedicated $Terminated$ places, which collect the tokens when no further message processing is scheduled to occur.

A conversation is a whole Petri Net composed of a set of subnets (i.e. protocols), where at least one role has the $Start$ place (initiator) and is connected to an arbitrary number of other conversation participants. A conversation state is a current net marking. A conversation policy may, in straightforward cases, be encoded via arc inscriptions and guards inside roles of existing conversations. In more complicated cases, a conversation policy can be encoded as a parallel Petri Net that lies above the existing conversation protocol and represents exceptional, or "off-topic" conversational elements that may take place at various times during the ordinary conversation. See Figure 1 for an example of such a policy-level Petri Net.

It is natural to compose more complex conversation models out of simpler conversations or sets of protocols by connecting appropriate elements by arcs. It is important to note that complex conversations do not change the semantics of the protocols (subnets).

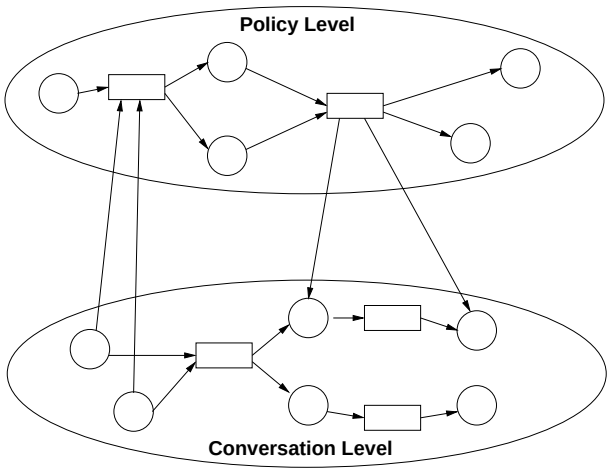

Figure 1: Coloured Petri Net incorporating conversation and policy level diagrams

For consistency, all basic act exchange schemas are defined via protocols, even if only a single communicative action is executed between two agents (single act without a response). That means that all communicative acts defined in an Agent Communication Language (ACL) (such as FIPA ACL [11] or KQML [9]) have at least one protocol defined for them .

6. EXAMPLES

To demonstrate the expressive power of Petri Net based interaction protocols, we have decided to show some of the examples based on the FIPA [11] interaction protocols specifications. FIPA has defined a collection of simple interaction protocols, which can be used in separation or in conjunction with other protocols. We will start with two very simple ones, $inform$ and $request$. The former is a simple communicative act for passing a single statement (proposition) from one agent to another, the latter is a simple request for an execution of an action. Then we will discuss a more complex example with a simple contract net protocol, which in the final example we extend to handle other unspecified cases of interactions and we demonstrate how to compose complex conversations out of simpler interaction protocols.

For an informal outline of the protocol, we have chosen a notation based on FIPA 97 specifications. FIPA [11] used a notation (in its previous specifications) based on Deterministic Finite Automata, represented graphically simply as connected boxes. Boxes with double edges represent communicative actions, which can also be treated as states; white boxes represent actions performed by initiators; shaded boxes represent actions performed by other participants in the protocol. Connections between boxes can be interpreted as transitions. For simplicity we have skipped `not-understood` responses, which can be sent in response to virtually any communicative act.

For Petri Net models we use the notation introduced in Section 5. It is important to distinguish interaction protocols, i.e. individual subnets, from the conversation models. For simplicity and clarity

of the diagrams, only names of places and transitions are presented, and inscriptions, guards and marking are left unspecified.

6.1 FIPA inform protocol

This is one of the simplest interaction protocols specified by FIPA[2]. There are only two participating agents, fixed at the beginning of the interaction, and the protocol basically consists of a single *inform* action being executed by one of the participating agents. Following FIPA conventions this interaction protocol can be represented as a single rectangle, shown on Figure 2.

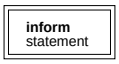

Figure 2: FIPA inform protocol

The same protocol represented as a primitive Petri Net is shown on Figure 3. It is worth noting that the Petri Net model captures the details of even such a simple interaction, and can be directly used to implement an application framework to handle this type of conversation - the developer needs simply to fill out the specific processing code inside transition actions, and clean up the subnet when the conversation is finished, i.e. when the token is placed in the $Terminated$ place. It is possible to reuse the model concurrently for more than a single inform conversation, and collect in $Terminated$ places information from the past activities of the conversation structures.

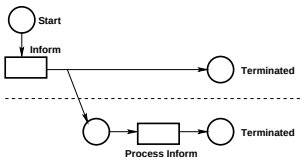

Figure 3: Inform protocol as a Petri Net model

6.2 FIPA request protocol

The FIPA *request protocol* simply allows one agent to request an action to be performed by another agent. The action request can be rejected or accepted, and once accepted can be finished with a success or failure. The schematic representation of this protocol is shown on Figure 4.

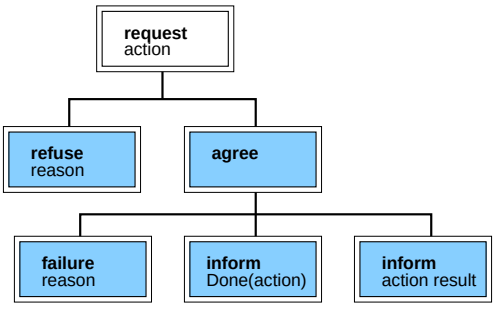

Figure 4: FIPA request protocol

[2]In fact it is not usually referred to as a *protocol*, but following the conventions introduced in Section 3, we will consistently call all interaction patterns as *protocols*, even if they only contain a single communicative act.

The Petri Net based model of a simple request conversation is drawn in Figure 5. As discussed in Section 5 all the arrows (transitions) crossing the roles boundaries represent message exchange between two agents (roles). We can call the upper role from the diagram an *employer* and the other role a *contractor*. The conversation formally specifies where and how the interaction between two interested parties occur, and what communicative acts are allowed in particular stages of the conversation.

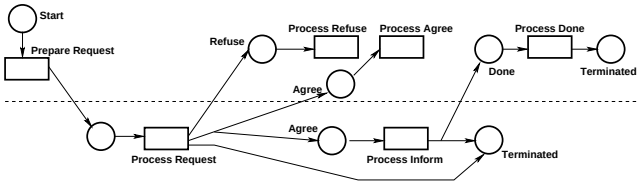

Figure 5: Petri Net request conversation

It is worth noting that in the Petri Net models it is always possible to capture the current state of the conversation via taking the current marking of all the participating subnets. If the request was successfully accomplished in both of the roles (subnets) there will be one token in each $Terminated$ place. In the real application though, those tokens need to be collected by means of some housekeeping mechanisms to free memory and release all irrelevant state information.

6.3 Contract-net protocol

We will use here a modified version of the FIPA contract-net protocol. In our model the manager wishes a task to be performed by one or a group of agents according to some arbitrary function which characterises the task. The manager issues the *call for proposals*, i.e. cfp act, and other interested agents can send $proposals$. In contrast to the original FIPA contract-net protocol, there is no need to do anything if an agent playing a role of a potential contractor is not interested in submitting proposals. That means that our contract-net model from the very beginning relies on the notion of timeout, i.e. some actions need to be performed in the event of a lack of enough proposals or even in the case of a complete lack of proposals.

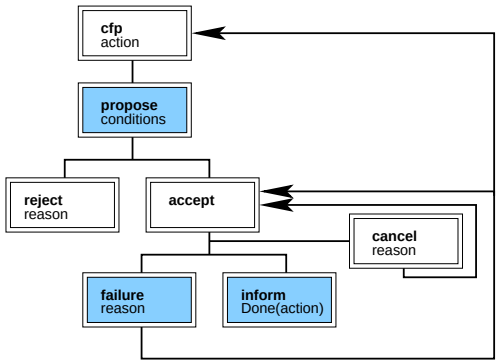

Figure 6: Custom contract-net protocol

The proposals are collected by the manager, and then they are rejected or accepted. The accepted proposals can be cancelled, either, by the manager via a $cancel$ action, or by the contractor via a $failure$ action. In case of cancellation other submitted proposals can be reconsidered, or a completely new call for proposals can be

issued. The schematic representation in the FIPA notation is presented on Figure 6. The Petri Net model is shown on Figure 7. Following FIPA naming conventions we will refer to the contract-net initiator as a *manager* and all other participants as *contractors*. In the Petri Net case, we have drawn an example conversation based on the contract-net protocol between a *manager* and three *contractors*.

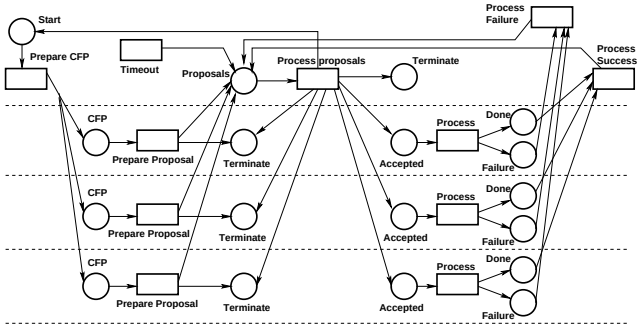

Figure 7: Custom contract-net conversation with three contractors

It is important to note here that the actual behaviour of the manager and contractors is not specified by the example net, and this information is encoded inside arc inscriptions and guards. Consider two potential strategies, i.e. conversation policies, the manager can follow during the course of the conversation:

1. wait for the first two proposals, choose the best one, accept the chosen one and reject the second one, and all other late proposals. If the chosen proposal fails[3], reissue the cfp again and follow this approach all over again until the task is successfully accomplished.

2. wait for the first two proposals, choose the best one, accept the chosen one, but do not reject the second one, keep collecting the incoming proposals instead. Once the chosen one is finished successfully, reject all other proposals. If the chosen one fails, choose the next best, and iterate through the process until successful, or in the case of no more proposals waiting, reissue cfp.

One can build more complex conversations based on the manager and contractor roles. It is even possible to combine two or more protocols into a single conversation model. It is not difficult to imagine a contract-net protocol to work together with $request$ and $inform$ protocols. Imagine a situation when a contractor is not really capable of performing the advertised job, but acts on behalf of one or more agents, which are capable of performing the job. In such a case, setting up an agreement between a contractor broker and a real worker can be achieved via the $request$ and $inform$ protocols, whereas the other part of the conversation, between the broker and manager, is done with the unmodified contract-net protocol.

With the complex interaction schemas it is possible but not desirable to reuse the net models and structures for concurrent unrelated conversations, as the net is already being used in a concurrent fashion by a single concurrent conversation. In such a case the creation of separate structures for each of the conversation instances is suggested for the sake of simplicity. This approach is undertaken by our current implementation. However it is worth mentioning that with coloured Petri Net-based underlying model, it is possible to use a single net structure even in those complex concurrent cases. In that case an appropriate additional matching based on conversation identifiers is necessary inside the arc expressions and transition code.

[3]The proposal fails when the contractor cannot successfully finish the contract, or if the contractor decided to drop the intention of finishing the agreed action.

7. SUMMARY

We have developed a Petri Net simulator which allows during runtime the construction of conversations out of simple predefined protocols, plug-in on-the-fly additional participants, and the specification of policies for different roles in a conversation. There is not yet a fully integrated visualisation module which will enable visualizing and monitoring progress of the conversation. At the moment only marking dumps can be used to debug a conversation. The prototype implementation allows construction of complex concurrently progressing conversations. It also allows multiple instances of different conversations that use the same conversation structures (reuse concurrently the same conversation structures).

	FIPA 97 Protocol	UML sequence diagram	Deterministic State Automaton	Enhanced Dooley Graph	Petri Net
Can model a protocol	√	√	√	√	√
Can model a conversations composed of single protocol	√	√	√	√	√
Can model a conversation composed of many protocols		√	√	√	√
Can be constructed dynamically			√	√	√
Can visualise progress of a conversation			√	√	√
Can distinguish different participants				√	√
Can express a particular policy					√
Concurrency supported					√
Maps directly to the implementation		√	√		√
Reuse of the runtime structures					√

Figure 8: Features summary

In Figure 8 we present a brief summary of the different features and their support by different formalisms. We discuss the comparison below.

Can model a protocol refers to a formalism's ability to model a single interaction pattern, i.e. a protocol. All of the discussed formalisms can cope with this task well, as this is the primary requirements for the formalism to be useful. The protocol needs to

be split into the initiator and other parties, and in some of the formalisms this split is done explicitly promoting reuse (UML, Petri Nets). In some however it results in a complete separation of roles, which means the designer ends up with a set of loosely decoupled *parts* not connected with one another (DFAs). This is undesirable, because the information of individual participants is lost and the conversation cannot be easily monitored as a whole.

Can model a conversation composed of one protocol refers to the ability of expressing a whole conversation model, in the case when the conversation consists of a single protocol. As in the case above, all discussed formalisms cope with this task well, as they are basically used mostly for that purpose. As mentioned before, some of the formalisms will provide only the state information.

Can model a conversation composed of many protocols refers to the ability of expressing a whole conversation model, in the case when the conversation consists of more than a single protocol. Apart from the simplistic notation of FIPA 97 specification, all formalisms could be successfully used to plug more than a single protocol into a complete conversation model.

Can be constructed dynamically refers to the ability of the formalism to have a model built dynamically at runtime, i.e. the model emerges from the progress of the ongoing conversation, and the formalism can capture some of the aspects of the ongoing conversation and can provide some quantitative measurements. Three formalisms based on different graph-based modelling can generally cope with this task quite easily (DFAs, Dooley Graphs, Petri Nets), however extended UML notation provides no support here. For UML models the roles and interactions need to be known in advance, and there is only limited support for capturing the conversation model from the ongoing conversation.

Can visualise progress of a conversation refers to the ability of the formalism to represent graphically the progress of a known conversation. Similarly to the previous point, all graph-based modelling techniques are quite useful here, and provide such an ability. UML also provides tools here to visualise the progress of the ongoing conversation via sequence diagrams, state charts and activity diagrams. Unlike with graph-based solutions where the progress can be directly shown on the conversation model, with UML there is a need for a separate notion of the conversation model, and a separate notion for the processes within the model. This can be useful in some cases, however it may introduce an unnecessary level of complexity in other cases.

Can distinguish between different participants refers to the ability of the model to capture the identity of different participants of a given conversation. This is only supported fully in Enhanced Dooley graphs and Petri Nets models. We consider this to be a very important feature for monitoring and debugging a running multi-agent system.

Can express particular policy refers to the ability of the conversation model to include a particular policy for conversation participants. Only Petri Net models with the expressive power of inscriptions and guards can cope with this task. With other formalisms this is left to the agent application layers and is not accessible on the conversation level as such. We consider this to be a unique and powerful feature of a proposed conversation modelling approach based on Petri Nets. We believe this to be an important aspect of conversations, and the policy logic should be expressed in the conversation model.

Concurrency supported refers to the ability to express concurrency on the conversation model level. Today, only Petri Net models are considered to support this feature fully.

Maps directly to the implementation refers to the ability of generating automatically an implementation supporting protocols and participants for a given conversation model. UML-based, DFAs, and Petri Nets models are all suitable and easily mappable into implementation stubs and skeletons which can be generated automatically from the model.

Reuse of the runtime structures refers to the ability of concurrent reuse of the runtime structures by different instances of a given conversation. Only in the case of Petri Nets is this available to the agent programmer, simply by the propagation of different message instances through the different places of the same conversation Petri Net. In both UML-based and DFAs models it is possible to reuse the structures, but only in a sequential and synchronised fashion.

To conclude, coloured Petri Nets offer a useful abstraction for use in modelling complex concurrent interactions driven by discrete events. As such, they are suited as a modelling formalism for agent interaction protocols and agent conversations. It has been shown in this article that apart from the underlying formal mathematical model, there are other useful features of the proposed approach, namely a consistent modelling formalism on all levels of abstractions: from the protocols and conversation specifications through the implementation and conversation integration with the given agent and agent platform, to the conversation monitoring and debugging. In the proposed modelling approach Petri Net protocols, conversation models and policy models can be used not only for the specification itself, but can also help to deploy a given conversation and ensure the correctness of a particular conversation implementation. The approach can assist an agent developer to construct interaction-based behaviour, and keep track of the progress of concurrently running conversations along with possible changes in policy level interaction protocols. It can also promote reuse of conversation structures, and provide a more compact implementation than alternative solutions.

8. FUTURE WORK

Predefined conversations are enough for some domains and some problems, but more general models for constructing and sharing conversation models and protocols need to be addressed. Further investigation of dynamically constructed conversations out of a collection of simpler predefined conversation protocols has not been fully investigated so far and it is a direction for possible future research. Such dynamically constructed conversations can grow into considerably large and complex structures. Management of such dynamically created and enacted conversation is the issue, which could potentially benefit from an underlying Petri Net representation as outlined in the article.

We will continue the development of software tools supporting the proposed modelling technique, and in particular the development of a graphical representation and visualizer for debugging and monitoring purposes.

9. REFERENCES

[1] J. L. Austin. *How to Do Things With Words*. Oxford University Press: Oxford, England, 1962. J. O. Urmson

editor.

[2] J. M. Bradshaw, S. Dutfield, P. Benoit, and J. D. Woolley. Kaos: Toward an industrial-strength open agent architecture. In J. M. Bradshaw, editor, *Software Agents*. AAAI/MIT Press, 1998.

[3] R. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng. Modeling agent conversations with colored petri nets, May 1999.

[4] S. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng. Using colored petri nets for conversation modeling, 1999. Available online at http://www.csee.umbc.edu/∼jklabrou/publications/ijcai99acl.ps.

[5] M. Diaz. Petri nets based models in the specification and verification of protocols. In *Petri Nets: Applications and Relationships to other Models of Concurrency, Advances in Petri Nets, Part III, Lecture Notes in Computer Science*, volume 255. Springer-Verlag, 1987.

[6] R. A. Dooley. Repartee as a graph. In *Appendix B in Longacre 76*, pages 348–358. 1976.

[7] R. Elio and A. Haddadi. On abstract task models and conversation policies. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 89–98, May 1999.

[8] J. Ferber. *Multi-Agent Systems - An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999.

[9] T. Finin, Y. Labrou, and J. Mayfield. Kqml as an agent communication language In Software Agents, 1997. Available at http://www.cs.umbc.edu/kqml/papers/kqmlacl.pdf.

[10] FIPA. FIPA Spec 2 - 1999. Agent Communication Language. Draft, Version 0.1, Available at FIPA http://www.fipa.org/specifications/index.html, 16 April 1999.

[11] FIPA. FIPA specification. FIPA specifications online at http://www.fipa.org/specifications/index.html, 2000.

[12] K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*, volume 1: Basic Concepts. Springer-Verlag, Berlin, 1992.

[13] J. l. Peterson. *Petri Net Theory and the Modelling Systems*. Prentice-Hall, 1981.

[14] J. B. Mark Greaves, Heather Holmback. What is a conversation policy? Mathematics and Computing Technology The Boeing Company P.O. Box 3707.

[15] Object Management Group. OMG Unified Modeling Language Specification, version 1.3. Available at OMG: ftp://ftp.omg.org/pub/docs/ad/99-06-09.pdf, June 1999.

[16] J. Odell, H. V. D. Parunak, and B. Bauer. Extending UML for agents. In E. Y. Gerd Wagner, Yves Lesperance, editor, *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, pages 3–17, 2000. Available online at http://www.jamesodell.com/ExtendingUML.pdf.

[17] J. Odell, H. V. D. Parunak, and B. Bauer. Representing agent interaction protocols in UML. In P. Ciancarini and M. Wooldridge, editors, *Proceedings of the First International Workshop on Agent-Oriented Software Engineering 2000*. Springer-Verlag, 2000. Available online at http://www.jamesodell.com/Rep _Agent_Protocols.pdf.

[18] H. V. D. Parunak. Visualizing agent conversations: Using Enhanced Dooley graphs for agent design and analysis. In *Proceedings of the Second International conference on Multi-Agent Systems (ICMAS'96)*, 1996. Available online at http://www.erim.org/∼vparunak/dooldesn.pdf.

[19] W. Reisig. Petri nets. an introduction. In *EATCS Monographs on theoretical Computer Science*, volume 4. Springer-Verlag, 1985.

[20] J. R. Searl. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press: Cambridge, England, 1969.

[21] M. P. Singh. Agent communication languages: Rethinking principles. In *IEEE Computer, 0018-9162*, pages 40–47. December 1998.

[22] W. M. P. van der Aalst. Three good reasons for using a petri-net-based workflow management system. In S. Navathe and T. Wakayama, editors, *Proceedings of International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, November 1996.

[23] W. M. P. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computer*, 8(1):21–66, 1998.

[24] T. Wagner, B. Benyo, V. Lesser, and P. Xuan. Investigating interactions between agent conversations and agent control components. In *Agents 99 Workshop on Conversation Policies*. 1999.