Verifiable Protocol Design for Agent Argumentation Dialogues

Sotiris Moschoyiannis[1], Member, IEEE, Paul Krause[1], Daniel Bryant[1], Peter McBurney[2]

[1]Department of Computing, Surrey University, Guildford, GU2 7XH, UK
e-mail: {s.moschoyiannis | d.bryant | p.krause}@surrey.ac.uk
[2] Department of Computer Science, University of Liverpool, L69 3BX, UK, e-mail: p.j.mcburney@csc.liv.ac.uk

***Abstract*— We describe a formal approach to protocol design for dialogues between autonomous agents in a digital ecosystem that involve the exchange of arguments between the participants. We introduce a vector language-based representation of argumentation protocols, which captures the interplay between different agents' moves in a dialogue in a way that (a) determines the legal moves that are available to each participant, in each step, and (b) records the dialogue history. We use UML protocol state machines (PSMs) to model a negotiation dialogue protocol at both the individual participant level (autonomous agent viewpoint) and the dialogue level (overall interaction viewpoint). The underlying vector semantics is used to verify that a given dialogue was played out in compliance with the corresponding protocol.**

***Index Terms*— digital ecosystems, autonomous agents, negotiation, vector semantics, verification, digital economy.**

I. INTRODUCTION

An increasing number of software applications are designed and implemented using autonomous agents that can decide for themselves which goals to adopt and how these goals should be reached. In a digital ecosystem, participating entities need to be proactive in using ecosystem resources and making their own resources available. The autonomous agents need to interact in order to work together in finding proofs, to resolve conflicts of interest, or simply to inform each other about pertinent facts. Such communication requirements result in complex and distributed interactions that go beyond the exchange of single messages. Instead, agents need to exchange sequences of messages which all bear upon the same subject. In other words, they need to engage in *dialogues*.

A rational agent involved in a dialogue can express claims and judgements about the subject, aiming at reaching a decision or informing, persuading, negotiating with other agents. However, in large-scale open multi-agent systems such as those offered by SOA-based applications [1] envisioned in a digital ecosystem, pertinent information may be insufficient or partially incoherent. In such cases agents can be assisted by *argumentation*, a process based on the exchange and the valuation of interacting arguments which support opinions, claims or proposals for action.

The foremost advantage of the argumentation-based approach to dialogue [2],[3] is that agents only accept facts if they can be persuaded they are true or if they follow from things they already believe to be true. This naturally gives dialogue a form of social semantics [4] which is another aspect that features in digital ecosystems.

In previous work [5] we have been concerned with distributed transactions in digital ecosystems and have described a transaction model where each participating organisation has a local agent that coordinates the deployment of the underlying services. In this paper we extend this work to take into account argumentation-based dialogues, e.g., agents negotiating over a scarce resource.

Dialogues are often conducted based on formal dialogue games in which participants "move" by uttering sequences of locutions from a pre-defined set (the content of which depends on the agreed dialogue type [6]). The dialogue is governed by a protocol which specifies the right behaviour of agents involved in the interaction. In an open digital environment dialogues take place amongst different agents, with different objectives and beliefs. Therefore, we need to be able to verify that participating agents are compliant with the behaviour rules that the particular protocol expresses. Agent Communication Languages (ACLs) provide agents with the ability to communicate, but little guidance exists for protocol designers [7].

In this paper we describe a formal approach for designing agent dialogue protocols in which the protocol can be described using standard UML and then be interpreted into a formal framework that enables to check (in a state-based way) whether a dialogue was played out in compliance with the protocol. The expectations about agent behaviour are formalised as sets of *dialogue vectors* (essentially tuples of sequences, one for each agent) that describe the sequences of possible moves, and effectively record the dialogue history. The vector semantics is then used to determine ordering constraints on the agents' moves during a dialogue and prohibit behaviour that is not compliant with the protocol used in the interaction.

The paper is structured as follows. In Section II we give a brief account of protocols for agent argumentation dialogues. In Section III we describe how UML *protocol state machines* can be used to capture the legal sequences of moves for each participant. In Section IV we introduce the formal language of dialogue vectors and in Section V we show how this can be used to check that agents executing a dialogue are compliant with the protocol. The paper finishes with some concluding remarks and ideas for future work.

II. ARGUMENTATION-BASED DIALOGUES

In order to engage in a dialogue, the interested agents have to use the same set of moves and agree on a dialogue

type (e.g. persuasion, negotiation, deliberation, information-seeking, etc.). A *move* is understood here as the utterance of a locution or speech act. A protocol that governs the usage of each locution can then be used to express the right behaviour of the agents involved in the dialogue.

In the dialogue typology of [6] a negotiation dialogue takes place when autonomous agents in a digital ecosystem bargain over the division of a scarce resource in a way acceptable to all, with each individual party aiming to maximise its share. Arguably, negotiation dialogues may also occur in other scenarios such as the now-standard marketing model of products as comprising bundles of features - buyers making trade-offs between competing feature-bundles such as price, after-sales service and product warranties. For instance, an offer(x) move may be used in this case to set the context for the negotiation to follow, where x is a choice (of product-bundle, say) between different alternatives and is returned by the argumentation framework deployed by the particular agent. Further considerations about the argumentation-based reasoning model are beyond the scope of the present paper.

A theoretical framework for argumentation-based agents dialogue and the definition of dialogue protocols has been developed in [8] In this paper, we will be concerned with the design of agent argumentation protocols. We shall use a negotiation dialogue protocol to illustrate the key ideas. Drawing upon [8] which identifies a set of moves for a general model of dialogue, we may consider the following set of moves for negotiation.

M_{neg} = {request(neg), refuse(neg), accept(neg), offer(), accept(), refuse(), challenge(), argue(), withdraw}

We may now also consider a protocol that specifies when a particular move can be made in the course of a negotiation dialogue neg between autonomous agents. The protocol is given in Table 1 in the form of pre- and post-conditions for each move, and these are expressed in terms of other moves.

***m1*: request(neg)**	PRE = no moves by either participant may occur before POST = accept(neg), refuse(neg)
***m2*: refuse(neg)**	PRE = request(neg) POST = no moves by either participant may occur after
***m3*: accept(neg)**	PRE = request(neg) POST = offer(x), withdraw
***m4*: offer(x)**	PRE = accept(neg) POST = accept(x), refuse(x), challenge(x), withdraw
***m5*: accept(x)**	PRE = offer(x) POST = accept(y), withdraw
***m6*: refuse(x)**	PRE = offer(x) POST = offer(y), withdraw
***m7*: challenge(x)**	PRE = offer(x) POST = argue(reasons for x), withdraw
***m8*: argue(for x)**	PRE = challenge(x) POST = accept(x), refuse(x), offer(y), withdraw
***m9*: withdraw**	PRE = 1. request(neg) followed by accept(neg) and 2.Neither agent can have uttered withdraw previously POST = No further moves can occur after withdraw

Table 1. A protocol for agent negotiation dialogues

It can be seen that the next move in a dialogue depends on the reasoning model of the agent (used to select between moves, and their content if necessary), but also on the previous moves played by each participant. We assume that an agent making an offer must be willing to honour that offer if the other agent issues an accept. This means that there is no need for the agent making the offer to confirm its own acceptance. Note that termination is not guaranteed under this protocol; the agents may interact indefinitely with neither leaving the dialogue (and no transaction taking place, in the case of the product-bundles scenario).

For example, an offer(x) move by one agent may be accepted, refused or challenged via a challenge(x) move by the other agent. This incites the agent to give an argument y in favour of x, and it does so via a move argue(y). We do not define here the grammar for the content of a move (*m4*–*m8*). It suffices to understand that these are propositions in a logical language *L* closed under negation, and can be retrieved from a knowledge base containing formulae in *L*.

We note that it would be possible to have different protocols and different locutions for the same negotiation application scenario. What is important however is that our approach to the design of agent argumentation protocols, described in the remainder of the paper, is generic and can address different sets of locutions and different protocols.

III. DESCRIBING DIALOGUE PROTOCOLS

In this section we describe the argumentation-based dialogue protocol from the individual participant's perspective. We use the negotiation dialogue protocol for illustration, but it should be noted that our approach applies equally well to other dialogue types (e.g. persuasion).

We have seen that there is a subset of moves that can be made in a negotiation dialogue. In fact, each participant can only use a subset of these moves. This subset is determined with the first move of the dialogue as follows. If, say, the agent *a1* wishes to enact a negotiation dialogue, and it does so by uttering request(neg), it is then restricted to the following subset of moves,

M_{a1} = {*m4*: offer(x), *m8*: argue(reasons for x)}

We note that there is always the case that the other agent, *a2*, does a move *m2*: refuse(neg) afterwards in which case a (negotiation) dialogue is never started. If participant *a2* accepts the request to engage in a negotiation dialogue, then it is restricted to the following moves,

M_{a2} = {*m5*: accept(x), *m6*: refuse(x), *m7*: challenge(x)}.

The point to be made here is that *a1*, by taking the initiative for entering a negotiation dialogue with *a2*, has effectively restricted *a2* to the set of moves M_{a2} and itself to the set of moves in M_{a1}.

We note that, once a dialogue has been started, either agent can withdraw at any point during the dialogue. We also note that we assume the participating agents have agreed on the dialogue type. The remainder of the paper will thus focus only on the two subsets M_{a1}, M_{a2} of moves as *m1*, *m2*, *m3* and *m9* do not add anything interesting to the behaviours (and so we can simplify the discussion).

The protocol for a negotiation dialogue outlined in Section 2, imposes certain conditions on the orderings between moves. Given the subset of these moves that is available to each participant, each participant is restricted to

particular sequences of dialogue moves - those that can be formed over the corresponding subset.

Agent *a1* has to start with *m4* and only then can (on occasion, as we will see) proceed to do *m8*. This can be captured using the concept of a *protocol state machine* (PSM) in UML2.0 [9]. Figure 1 shows the corresponding PSM using the notation of UML state diagrams.

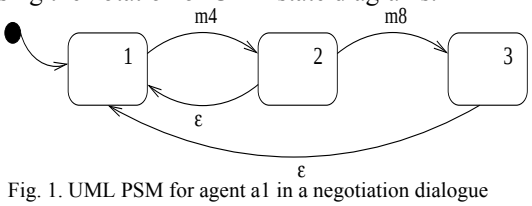

Fig. 1. UML PSM for agent a1 in a negotiation dialogue

Transitions labelled by ε do not correspond to any move and are used as a means of returning to the initial state, where everything is again possible. The semantics of an ε transition is that it is taken once all activity at the source state has completed (incl. exit actions, if any) and execution continues with the target state (incl. entry actions, if any). This bears some relevance to the handling/updating of agent commitment stores, though we will not be concerned with commitment stores in this paper. The state diagram of Figure 1 describes the possible sequences of moves for agent *a1* in the course of a negotiation dialogue.

We now consider the sequences of possible moves for agent *a2* in the course of a negotiation dialogue. Figure 2 shows the UML state diagram for the protocol state machine of agent *a2*. ε transitions are defined as before.

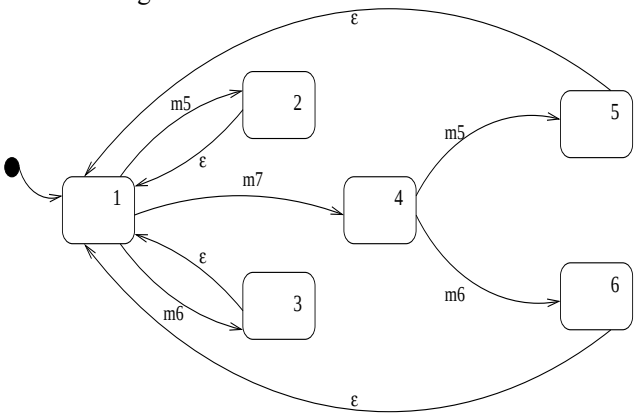

Fig. 2. UML PSM for agent a2 in a negotiation dialogue

It can be seen that the protocol state machine for a participant describes *all possible sequences* of dialogue moves (given the subset of moves corresponding to the agent). But a participating agent cannot do all of these sequences of moves in any single run of a negotiation dialogue. The sequences it can do each time are determined, in part, by the (sequences of) moves the other agent does, during the dialogue. Therefore, in order to be able to verify the correct behaviour of the participating agents, we need to determine the legally-possible moves *available* to each agent, at any given point in the course of a (negotiation) dialogue, i.e. as the dialogue progresses.

It transpires that we need a representation of a (negotiation) dialogue, in terms of the moves (externally visible behaviour) of the participants, that takes into account the behaviours of each participant at any point during a dialogue.

IV. A FORMAL LANGUAGE FOR AGENT DIALOGUES

In this section, we introduce a language-based representation of agent dialogue protocols that enables formal verification in that compliance with the protocol can be checked. We consider a formal language of vectors, essentially a set of *tuples of sequences* of moves (one sequence for each participant), rather than the usual trace semantics found in process algebras which can describe a single sequence of actions. Each agent is allocated a specific coordinate in the vector representation, which effectively records its moves. This representation of behaviour allows us to monitor all participating agents at once; each vector provides a snapshot of the dialogue in which we can see what moves have been played by each agent up to that point. When a dialogue is played out, we end up with a set of such vectors that describes the dialogue history.

This formal framework draws upon early work on vector languages [10] which has been subsequently extended to model interactions in a UML-type framework [11]. In what follows, we describe how we may adapt such a construction to obtain a vector language representation of a dialogue which formalises the interplay between the moves available to each participant.

Let *DA* denote the set of agents, equipped with a *theory* (as described in [8], ch.2), involved in a dialogue. Let *M* denote the set of moves.

Dialogue signature. A *dialogue signature* is a pair $\Sigma = (A, \mu)$ where

$A \subseteq DA$ is a set of *participants*, and

$\mu: A \to \wp(M)$ hence, μ returns the set of moves associated with each participating agent.

We require that there exists a unique $a \in A$ that enacts the dialogue, and also that all moves in a dialogue are a subset of the set of moves available, i.e., $\bigcup_{a \in A} \mu(a) \subseteq M$.

In the course of a dialogue, each agent will do a sequence of moves or utter a sequence of locutions. These will be recorded in the so-called *dialogue vectors*. Note that each move gives rise to a distinct vector.

Dialogue vectors. Let Σ be a dialogue signature. Define V_Σ to be the set of all functions $\underline{v}$: $A \to M^*$ such that for each $a \in A$, $\underline{v}(a) \in \mu(a)^*$. We shall refer to elements of V_Σ as *dialogue vectors*.

By $\mu(a)^*$we denote the set of finite sequences over $\mu(a)$. In mathematical terms, the set V_Σ is the Cartesian product of the sets $\mu(a)^*$, for each a. Effectively, dialogue vectors are n-tuples of sequences where each coordinate corresponds to a participating agent (hence, n is the number of participants) and contains a finite sequence of moves the corresponding agent has played.

Example. Suppose that two agents, $a1, a2 \in A$ engage in a negotiation dialogue, i.e. $M = M_{neg}$, and *a1*, who enacts the dialogue, is allocated the first coordinate and *a2* the second. We have that $\mu(a1) = \{m4, m8\}$ and $\mu(a2) = \{m5, m6, m7\}$. The following are dialogue vectors, and we use Λ to denote the empty sequence.

(Λ, Λ), $(m4, \Lambda)$, $(m4, m7m5)$, $(m4, m5m7)$, $(\Lambda, m5)$

Compare with the following, which are *not* dialogue vectors over the given signature Σ.

($m4$, Λ, Λ), ($m4$, $m4$), ($m4$, $m7$, $m7m5$)

For instance, the dialogue vector ($m4$, $m7m5$) says that agent $a1$ has made $m4$ and agent $a2$ has made a move $m7$ followed by $m5$ (the ordering is given by the usual prefix ordering on sequences). □

We have seen that dialogue vectors are essentially tuples of sequences. This means that we can define operations on vectors in terms of well known operations on sequences.

For $\underline{u}, \underline{v} \in V_\Sigma$, we define

- $\underline{u}.\underline{v}$ to be the unique vector $\underline{w}$ such that $\underline{w}(a) = \underline{u}(a).\ \underline{v}(a)$, for each $a \in A$ (*concatenation*)
- $\underline{u} \leq \underline{v}$ iff $\underline{u}(a) \leq \underline{v}(a)$, for each $a \in A$ (*prefix ordering*).

It is not hard to see that V_Σ is a monoid with binary operation '.' and identity $\underline{\Lambda}_\Sigma$, where $\underline{\Lambda}_\Sigma$ is the empty vector. Furthermore, V_Σ is a partially ordered set (poset) with partial order '$\leq$' and bottom element '$\underline{\Lambda}_\Sigma$'.

Dialogue vectors can be seen to be built up by a series of concatenations with a specific kind of vector $\underline{e}$ which describes a move. This is also a dialogue vector but has the additional constraint that each of its coordinates is either empty sequence or contains a single move. For example, the vector $\underline{e}$ = ($m8$, Λ) represents a move $m8$ by the agent corresponding to the first coordinate. If $m8$ is intended to occur only after both $m4$ and $m7$ have, then this is described in a normal dialogue vector $\underline{v}$ = ($m4m8$, $m7$) which is obtained as $\underline{u}.\underline{e}$ = ($m4$, $m7$).($m8$, Λ) = ($m4m8$, $m7$) = $\underline{v}$.

Thus, every move in the course of a dialogue is recorded in a distinct dialogue vector, and at the appropriate coordinate. This means that once a dialogue has been played out we end up with a set of such vectors, the so-called *dialogue language*. The intuition is that this is the language that was used in the particular dialogue.

Dialogue. A dialogue D is a pair D = (Σ, V) where

- Σ is the signature of D
- V_Σ is the dialogue language of D.

This definition says that a dialogue consists of a signature Σ which identifies the participants, and the moves associated with each, together with a 'language' of vectors formed over Σ. The idea is that the dialogue language indicates possible constraints on the order in which moves can be made and reflects the legally-possible moves that are available to each participating agent, at every stage in the course of a dialogue.

The study of the order-theoretic properties of such vectors in [11],[12] shows that it is possible to express causal dependency, mutual exclusion, concurrency and simultaneity within the corresponding languages. This development draws upon the relation between the algebraic and order-theoretic representation of behaviour established in [10]. For the purpose of the present paper, it suffices to understand that the order-theoretic properties can be exploited in determining what moves (out of the legally-possible ones) have already taken place, and on this evidence what moves are now available to each participant. In short, the ordering relation between different vectors in a dialogue language reflects the orderings between moves from different agents. In this way, we may restrict the sequences of moves each agent can do as the dialogue progresses.

For instance, the moves $m4$ and $m5$ are related by causality ($m5$ can only occur after $m4$ has) in Figure 3(i) while they are mutually exclusive (occurrence of one excludes future occurrence of the other) in Figure 3(ii). In Figure 3(iii) they are concurrent while in Figure 3(iv) they are simultaneous.

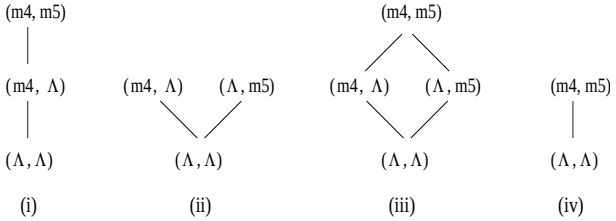

Fig. 3. Order structure and dependencies between agents' moves

In the treatment of argumentation-based dialogues given in [8], moves are played sequentially (and by alternating participants) and thus we will not be concerned with the cases (iii) and (iv) in this paper.

V. PROTOCOL DESIGN AND VERIFICATION

We are now set to show how dialogue languages can be used to restrict from the set of all legally-possible moves (as given by the PSM for each participating agent, in the previous section) to the set of legally-possible moves that are *available at each step* to each agent, as the dialogue progresses. We have seen, and Figure 3 may be instructive in this respect, that such information is found in the order structure of a dialogue language and this is dependent on context - on what other vectors are included in the language.

It becomes apparent that it is particularly important to be able to restrict to an appropriate subset V of V_Σ. In [12] we have shown how vector languages can be obtained from UML design models. In what follows we outline how dialogue languages can be obtained from UML interaction diagrams in particular.

Our starting point is the protocol specified for the dialogue type in question. The protocol for a negotiation dialogue was given in Section 2 in terms of pre- and post-conditions and these were expressed in terms of other moves. The pre- and post-conditions of each locution are used to axiomatise behaviour in the sense that they specify when each locution can be uttered - when a move involving that locution is legal. The resulting protocol is captured in the sequence diagram of Figure 4 which describes the legally-possible moves in an instance of a negotiation dialogue using the UML2.0 notation for interaction diagrams.

We now briefly describe how a negotiation dialogue language can be obtained from the scenario-based specification of the protocol for negotiation. The full details of the translation can be found in [12]. For this paper, it suffices to understand that the sending/receiving of a message in Figure 4 is associated to a move of the sender/receiver. Each move corresponds to (one or more) dialogue vector, which is obtained by considering the current move and the dialogue vector of the previous move.

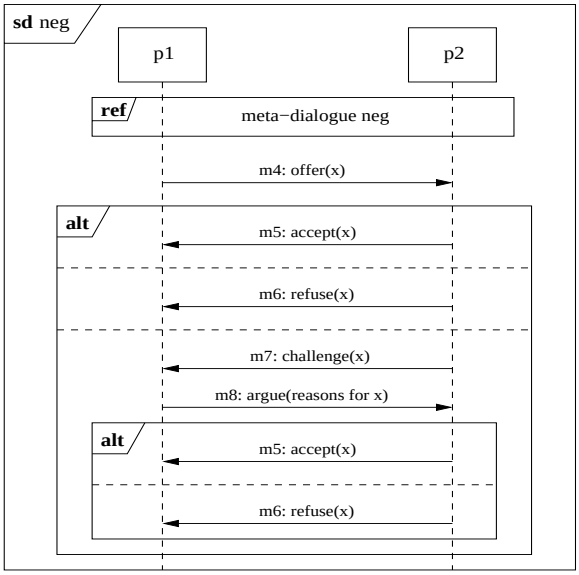

Fig. 4. UML sequence diagram for the negotiation dialogue protocol

We start with the empty vector (Λ, Λ) which reflects the fact that initially (at the beginning) nothing has happened. The first thing that can happen is an utterance of the locution offer(x) denoted by move *m4*, as before (recall Table 1). This is recorded in a dialogue vector $\underline{v} = (m4, \Lambda)$ at the coordinate corresponding to *a1* who made the (first) move, by coordinate-wise concatenation with (Λ, Λ) i.e. $(\Lambda, \Lambda).(m4, \Lambda) = (m4, \Lambda) = \underline{v}$.

In the case of moves belonging to different operands of an alternative fragment, denoted by the keyword **alt** in UML sequence diagrams, the previous move is considered to be the last move before entering the fragment. For example, the dialogue vectors corresponding to *m6* in Figure 4 are obtained based on those of *m4* and not *m5*. This reflects the alternative scenarios described in the diagram. The choice will be resolved based on the reasoning capabilities, as given by the argumentation framework agent *a2* uses (described in [8], ch. 2, 3). Full details on translating UML sequence diagrams into vector languages can be found in [12].

After considering all moves in the dialogue scenario described in Figure 4, we obtain the set of dialogue vectors,

$V = \{ (\Lambda, \Lambda), (m4, \Lambda), (m4, m5), (m4, m6), (m4, m7), (m4m8, m7), (m4m8, m7m5), (m4m8, m7m6) \}$

which is the language for a negotiation dialogue. Its order structure is depicted in Figure 5.

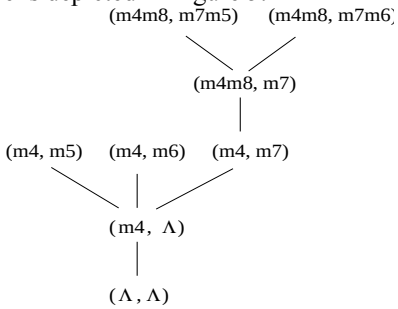

Fig. 5. Order structure of a negotiation dialogue language

Notice for example that $(m4, m5m7)$ is not included in V although it is a valid dialogue vector (recall *Example*, Section IV), because it turns out it does not describe intended behaviour during a negotiation dialogue.

Note that after *a1* has done move *m4*, the agent *a2* has 3 different moves available. If *m5* or *m6* is chosen, the execution of the present scenario is completed and the scenario can now be played again, but only from the beginning. Similarly, for the branch involving $(m4, m7)$. In fact, there are dialogue vectors in V which do not describe earlier behaviour than any other vector in V.

Maximal dialogue vectors. Let V be a dialogue language and $\underline{v} \in V$, then we say $\underline{v}$ is a *maximal* vector in V if there is no other vector $\underline{u} \in V$ such that $\underline{v} \leq \underline{u}$.

Maximal dialogue vectors determine the points in which repetition of the dialogue, about different topics, may occur. This means that it is appropriate to view a dialogue language as describing a *pattern of agent behaviour*, in terms of predefined moves, that can be repeated arbitrary many times.

Languages of vectors give rise to a class of automata, as described in [11], and this can be exploited in obtaining a state-based description of the moves during a dialogue. Dialogue vectors can be seen as states; effectively a dialogue vector represents that state reached after all the moves it describes have taken place. Further, we have seen that dialogue vectors are built up by coordinate-wise concatenation with vectors $\underline{e}$ each of whose coordinates is empty or contains a single move. Thus, the language-based representation of a dialogue can be readily associated with a state machine - by simply considering dialogue vectors as states and defining the transition relation in a way that reflects the fact that the vectors are built up by repeatedly concatenating vectors such as $\underline{e}$ to it.

The dialogue language for negotiation can be represented as a UML protocol state machine, in a fashion similar to the protocol state machines for each participant, presented in Section 3. The PSM for the negotiation dialogue protocol of Figure 4 is shown in Figure 6 using the UML2.0 notation.

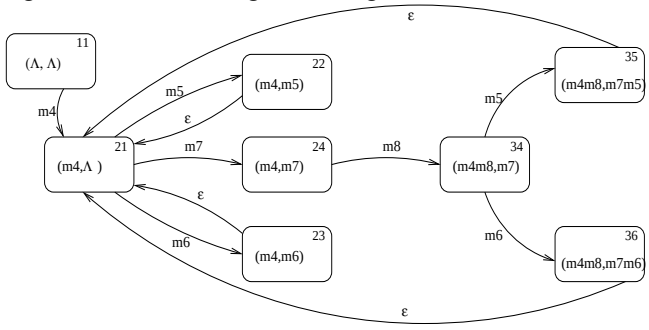

Fig. 6. UML state diagram for available moves in a negotiation dialogue

However, this now refers to the negotiation dialogue as a whole rather than from a participant's viewpoint. It encompasses the moves that are available to each participant at any given point in the dialogue rather than all possible moves a participant can do in a negotiation dialogue. This additional information is precisely what restricts the moves available to each participant, as the dialogue progresses, rather than simply specifying all legally-possible moves in a negotiation dialogue.

For example, it can be seen that when the dialogue is played out, *m8* cannot appear immediately after *m4*; one of *m5*, *m6*, *m7* must appear in between.

Therefore, we have moved from *all possible* sequences

of moves for each agent (Figures 1 and 2, respectively) to the set of moves available to each at every point in the course of a negotiation dialogue (Figure 6).

VI. CONCLUSIONS AND FUTURE WORK

The verification of the correct behaviour of participating agents in a dialogue plays a fundamental role in a digital ecosystem since the desired properties of the interaction are guaranteed only if the agents behave properly with regard to the protocol used. Our formal approach to agents protocol design goes some way to addressing this problem since it makes it possible to check that expectations of a dialogue are fulfilled by the actual agent behaviour, i.e. that moves expected (not) to happen have actually (not) happened, which is something that cannot be assumed *a priori* in an open society of autonomous agents [4] in a digital ecosystem.

The vector language representation of a dialogue we proposed captures (i) the legal sequences of moves of each participating agent, (ii) the dependencies between moves from different agents and (iii) the contribution of each agent to the dialogue history. Given the partial history of the interaction, up to some point, the underlying formalism determines the sequence of legal moves available to each agent at that point in the dialogue.

Due to space limitations we only hinted towards the automata generated by dialogue languages [11]. These look like the abstract state machines proposed in [13] for modelling dialogue protocols, but the difference is that while the ASMs of [13] need to be extended with a finite set (for each agent) to model agent dialogue protocols, in our case this *blackboard* architecture is already embedded in the structure of the automata, via the underlying dialogue vectors.

Existing approaches that opt for a more intuitive definition of agent protocols by means of a graphical notation include AML and AUML [14]. However these approaches lack a formal semantics and thus cannot be used in more rigorous approach to protocol design. A graphical notation based on flow charts together with a logic-based formalism for protocol specification is proposed in [15]. The graphical notation is proprietary and seems to be tailored to medical guidelines while we use standard UML.

We have implemented a system that supports agent interaction protocols for both deliberation and negotiation dialogues, of which an initial prototype version is presented in [16]. The system allows agents to exchange locutions by injecting them into a shared tuple space. A "Dialogue Manager" component is assigned to each participating agent, which not only records utterances and dialogical commitments, but also determines whether a move is appropriate. The dialogue languages described in this paper provide formal support for the Dialogue Manager by way of verifying whether the execution sequence of a particular dialogue is compliant with the protocol. Work is in progress on extending the tool to support a range of dialogue types, whose executions can be verified against the protocol state machine generated by the corresponding dialogue language.

Because of its formality, this approach potentially permits software agents to reason about protocols, and about dialogues within protocols, and thus for the agents themselves (rather than their designers) to select protocols and dialogue paths appropriate for particular objectives. Thus, this framework is particularly suitable for the dynamic nature of digital ecosystems as it may support implementation of service-oriented architectures [1] where the participants interact using protocols selected at run-time.

In addition, this approach would allow agents to verify protocol compliance in situations where the protocol is not decided (or not decided conclusively) before the dialogue begins (e.g. see [17]), but may evolve in the course of the dialogue. By their very nature, digital ecosystems can facilitate innovation and evolving business activities which makes this direction worth pursuing further.

VII. ACKNOWLEDGEMENTS

This work was partially supported by the EU-FP6 funded project OPAALS Contract No. 034824 and a UK-EPSRC PhD Studentship.

VIII. REFERENCES

[1] M. Papazoglou, P. Traverso, S. Dutdar, *et al*. Service-Oriented Computing Research Roadmap. In *Dagstuhl Seminar Proc. 05462, Service-Oriented Computing (SOC)*, pp.1-29, 2006.

[2] S. Parsons, C. Sierra, N. R. Jennings. Agents that Reason and Negotiate by Arguing. *Journal of Logic and Computation*, 8(3): 261-292, 1998.

[3] S. Kraus, K. Sycara, A. Evenchik. Reaching Agreements through Argumentation: a Logical Model and Implementation. *Artificial Intelligence*, 104(1-2):1-69, 1998.

[4] M. Singh. A Social Semantics for Agent Communication Languages. In *Issues in Agent Communication*, LNCS 1916, pp.31-45, Springer, 2000.

[5] A. Razavi, S.Moschoyiannis, P. Krause. A Coordination Model for Distributed Transactions in Digital Ecosystems. I*n Proc. IEEE Digital Ecosystems and Technologies (IEEE-DEST'07)*, 2007.

[6] D. N. Walton, E.C.W. Krabbe. Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning. State University of New York Press, 1995.

[7] P. McBurney, S. Parsons, M. Wooldridge. Desiderata for Agent Argumentation Protocols. In *Proc. Autonomous Agents and Multi-Agent Systems (AAMAS'02)*, pp.402-409, ACM Press, 2002.

[8] L. Amgoud, L. Bodenstaff, M. Caminada, *et al*. Report on Formal Argumentation System for ASPIC Project. Technical Report ULCS-07-005, University of Liverpool, 2006. Available at: http://www.csc.liv.ac.uk/research/techreports/techreports.html

[9] Unified Modelling Language: Superstructure, version 2.0. OMG document formal/05-07-04, 2005. http://www.omg.org/uml

[10] M. W. Shields. Multitraces, Hypertraces and Partial Order Semantics. *Formal Aspects of Computing*, 4 (1992):649-672, 1992.

[11] S. Moschoyiannis, M. W. Shields, P. J. Krause. Modelling Component Behaviour using Concurrent Automata. In *Proc. ETAPS 2005 – FESCA'05*, ENTCS 141(3):199-220, Elsevier,2005.

[12] S. Moschoyiannis, P. Krause, M.W. Shields. A True-concurrent Interpretation of Behavioural Scenarios. In *Proc. ETAPS 2007 – FESCA'07*, ENTCS, Elsevier, 2009. *To appear*

[13] R. Fernandez, U. Endriss. Abstract Models for Dialogue Protocols. *Journal of Logic, Language and Information*, 16(2):121-140, 2007.

[14] M.P. Huget. Agent UML Notation for Multiagent System Design. *IEEE Internet Computing*, 8(4):63-71, Jul-Aug 2004.

[15] F. Chesani, A. Ciampolini, P. Mello, *et al*. Protocol Specification and Verification by Using Computational Logic. In *Proc. WOA'05,*pp.184-192, 2005.

[16] D. Bryant, P. Krause, S. Moschoyiannis. A Tool to Facilitate Agent Deliberation. In *Proc. JELIA'06,* LNAI 4160, pp.465-468, 2006.

[17] J. McGinnis, D. Robertson. Realizing Agent Dialogues with Distributed Protocols. In *Proc. Agent Communication*, LNAI 3396, 2004.