Dialogue Protocols for Multi-Agent Systems

by

Christopher Walton
Abstract:
In this paper we propose a new agent communication language which separates agent dialogue from any specific agent reasoning technology. This language is intended to address a number of perceived shortcomings with the mentalistic model of agent communication on which the FIPA-ACL standard is founded. Our language expresses inter-agent dialogue through the use of agent protocols, and is intended to be independent of the technology used for message delivery. In this paper we specify the syntax of our communication language, together with an operation semantics which defines an implementation of the language. Our language specification is derived from process calculus and thus forms a sound basis for the verification of our agent protocols.

Keywords: Multi-Agent Systems, Dialogue Protocols, Agent Communication Languages
Dialogue Protocols for Multi-Agent Systems

Christopher D. Walton *

Centre for Intelligent Systems and their Applications (CISA),
Appleton Tower, Crichton Street, Edinburgh, UK, EH8 9LE.

Email: cdw@inf.ed.ac.uk Tel: +44(0)-131-650-2718 Fax: +44(0)-131-650-6513

September 30, 2003

Abstract

In this paper we propose a new agent communication language which separates agent dialogue from any specific agent reasoning technology. This language is intended to address a number of perceived shortcomings with the mentalistic model of agent communication on which the FIPA-ACL standard is founded. Our language expresses inter-agent dialogue through the use of agent protocols, and is intended to be independent of the technology used for message delivery. In this paper we specify the syntax of our communication language, together with an operation semantics which defines an implementation of the language. Our language specification is derived from process calculus and thus forms a sound basis for the verification of our agent protocols.

1 Introduction

A Multi-Agent-System (MAS) may be defined as a collection of agents, which are autonomous and rational components, that interact within an environment. An individual agent of a MAS exhibits intelligent behaviour based on interactions with other agents, the environment, and internal reasoning processes. It is this intelligent behaviour that distinguishes a MAS from a conventional distributed or parallel software system. From this definition it is clear that an essential pragmatic consideration in the construction of a MAS must be a specification for the interactions between individual agents, as agents must interact in order to exhibit intelligent behaviour. A popular basis for this interaction is the theory of theory of rational action by Cohen and Levesque [CL90]. The FIPA-ACL specification [FIP99] recognises this theory by providing a formal semantics for the performatives expressed in BDI logic [RG98]. However, there is a growing dissatisfaction with the mentalistic model of agency as a basis for defining inter-operable agents between different agent platforms [Sin98].

Inter-operability requires that agents built by different organisations, and using different software systems, are able to safely communicate with one another in a common language with an agreed semantics. The problem with the BDI model as a basis for inter-operable agents is that although agents can be defined according to a commonly agreed semantics, it is not generally possible to verify that an agent is acting according to these semantics. This stems from the fact that it is not known how to assign mental states systematically to arbitrary programs. For example, we have no way of knowing whether an agent actually believes a particular fact. For the semantics to be verifiable it would be necessary to have access to an agents’ internal mental states which is not typically possible. This problem is known as the semantic verification problem and is detailed in [Woo00].

To understand why semantic verification is a highly-desirable property for an inter-operable agent system it is necessary to view the communication between agents as part of a coherent dialogue between the agents. According to the theory of rational action, the dialogue emerges from a sequence of speech acts performed by an agent to satisfy their intentions. Furthermore, agents should be able to recognise and reason about the other agents intentions based upon these speech acts. For example, according to the FIPA-ACL standard, if an agent receives an inform message then it is entitled to believe that the sender believes the proposition

---

*This work is sponsored by the UK Engineering and Physical Sciences Research Council (EPSRC Grant GR/N15764/01) Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (IRC).
in the message. There is an underlying *sincerity assumption* in this definition which demands that agents always act in accordance with their intentions. This assumption is considered too restrictive in an open environment as it will always be possible for an insincere agent to simulate any required internal state, and we cannot verify the sincerity of an agent as we have no access to its mental states. This problem precludes dialogues which are not fully co-operative, for example, negotiation or persuasion dialogues. In order to avoid the problems associated with the mentalistic model, and thereby express a greater range of dialogue types, a number of alternative semantics for expressing rational agency have been proposed. Two of these approaches are a semantics based on social commitments, and a semantics based on dialogue games. A summary of these approaches, and other semantic models is presented in [MC02].

In this paper we do not adopt a specific semantics of rational agency, or define a fixed model of interaction between agents. Our belief is that in a truly heterogeneous agent system we cannot constrain the agents to any particular model. Instead, we define a model of dialogue which separates the rational process and interactions from the actual dialogue itself. This is accomplished through the adoption of a *dialogue protocol* which exists at a layer between these processes. This approach has been adopted in the Conversation Policy [GHB99] and Electronic Institutions [ERS+01] formalisms. The definition presented in this paper differs in that dialogue protocol specifications can be directly executed.

A dialogue protocol allows the semantics of a dialogue to be independently expressed. This approach has some compelling advantages, for example, we can succinctly express the rules of an auction as a dialogue protocol, while the agents participating in this dialogue are free to select their own auction strategies. Agents can be specified in different languages, using different rational processes, and still participate in the dialogue expressed in the protocol. The only restriction on the agents is that they follow the dialogue protocol which encodes all the necessary information to participate in the dialogue.

It should be noted that dialogue protocols also greatly assist in the design of large MAS as they impose structure on the agents, co-ordinate tasks between agents, and define commitments which agents must satisfy. They also simplify the design of individual agents as they separate the task of defining the co-ordination of the agents from the definition of agent behaviours. This separation also permits the refinement and verification of the agent protocol independently from the design of the individual agents.

In this paper we present our approach to defining a dialogue protocol suitable for constructing large MAS of inter-operable agents. Our method draws from our earlier experiences with defining Electronic Institutions and borrows its theory from the process calculus domain. In section 2 we describe the existing work on Electronic Institutions, highlighting a number of problems with this approach. In section 3 we present an alternative which uses a small language to cleanly express dialogue protocols. In section 4 we present a relational operational-semantics for evaluating our language which can be implemented in an agent system. Lastly, in section 5 we describe our implementation and discuss future work concerning the verification of our dialogue protocols.

## 2 Electronic Institutions

The Electronic Institution (EI) framework [RMN+00] is a popular technique for specifying agent dialogue protocols. The underlying rationale of this framework is that human interactions are never unconstrained, rather they are controlled by such notions as conventions, customs, etiquette, and laws (i.e. social norms). Institutions are a means of representing and enforcing social norms in a bounded space, analogous to a human institution, e.g. a marketplace. The EI framework provides a method for controlling the interaction of agents in a MAS using formally defined institutions.

An institution can be viewed as a space in which a group of agents enter, interact through a dialogue protocol, and then leave. Agents take one (or more) *roles* in the institution. The interactions are articulated through the use of *scenes* in which groups of agents directly interact. Within a scene, all the participating agents follow a single dialogue (modelled as a finite-state system) which guides their interactions. Scenes are composed as a network called a *performative structure*. Agents move through the institution, participating in different scenes, with the possibility of assuming different roles in each. Because the actors are agents and not real humans, some additional terminology is also required. The agents interact through the use of speech acts. Institutions define the acceptable vocabulary (i.e. ontology) through the use of a *dialogic framework*. The actions of an agent in the context of an institution may have consequences that either limit or enlarge
its subsequent acting possibilities. The possible paths for an agent within the performative structure are thus defined by a set of normative rules containing obligations and commitments. We will not go further into the formal definition of institutions here as these details are presented in [ERS+01]. We note that institutions are defined as instances of state-charts [Har87].

![General Practitioner Scene](image)

It is helpful to consider an example of an institution in order to illustrate these concepts. In Figure 1 we present an example scene which would form part of a larger institution. This scene defines an interaction protocol between doctor and patient agents. This is the scene intended to represent a patient visiting a General Practitioner (GP) to obtain a diagnosis of some symptoms. There are two roles in this scene: doctor and patient. Any number of agents satisfying these roles are permitted. For convenience we assume that all agents use the same dialogic framework (i.e. they know how to communicate) and that there are no normative rules. The scene begins with all the agents entering the INITIAL state. A patient agent then sends a request message to a doctor agent indicated by request(P, D), where P is a patient and D is a doctor. This message is intended to represent the patient making an appointment to see a doctor. The patient then enters the WAIT state until an accept(D, P) or reject(D, P) message is received from the doctor. If a reject message is received, then the agent returns to the initial state. If an acceptance is received, the agent enters the ACCEPT state and proceeds to send a message symptoms(P, D) to the doctor. The doctor then performs a diagnosis of the patient in the DIAG state and the result is that the agent is referred refer(D, P) for further diagnosis, or no-referral norefer(D, P) is made and the patient leaves the scene.

The use of the EI framework addresses a large number of issues in the design of MAS. However, our experience in implementing such models has highlighted a number of issues which have yet to be addressed in this framework. There are several ambiguities in the definition [ERS+01], such as how agents move between institutions. However, such issues can readily be addressed and are not of importance here. Rather, there are three pertinent issues which we consider to be real obstacles in the deployment of institution-based MAS:

1. There is no mechanism for disseminating institution protocols.
2. Institution protocol specifications are not directly executable.
3. Institutions need synchronisation to ensure that they operate smoothly.

The dissemination of institution protocols to individual agents appears to be a significant problem. This problem arises when a new agent wishes to participate in an institution. The agent must follow the design of the institution, but this requires that the agent knows the internal details of the institution. The current approach used to design agents which will operate inside an institution is to construct a plan for the agent with respect to a particular institution, and then synthesise an agent description from this plan [VSSQ02]. The assumption is that it will be known in advance which scenes the agent will participate in. This assumption is valid for a number of simple models, but breaks down when used with more complex models. For example, if we consider our medical diagnosis scenario, the patient will not be aware of which scenes are required until a diagnosis is performed by the doctor. Thus, in order to synthesise such an agent it is necessary to consider all possible outcomes of the diagnosis and design the agent to cope with all such possibilities. If we consider all possible medical diagnoses then the problem quickly becomes intractable. An additional problem with the synthesis of agents from plans is that the institution definition cannot change. If the definition changes, then the corresponding plans will also change, and all the agents must be re-synthesised. The synthesis process is not automatic and thus the agents may need substantial reworking even under minor changes.
The second issue concerning the execution of institutions is closely related to the dissemination problem. Institution protocols are non-deterministic, i.e. there can be multiple transitions between states in a scene. In such cases, an agent must make a choice between subsequent states. For example, in the WAIT state of Figure 1, the doctor must choose whether to accept or reject the patient. However, the protocol does not make any attempt to define when the agent should choose one state over another. Thus, in order to implement a dialogue protocol within an agent, it is necessary to manually assign behaviours to all of the choice points within the protocol during agent synthesis. This can be a difficult task if the protocol is large or the intention of the protocol designer is unclear. It can be argued that the assignment of these behaviours belongs to the rational layer, and therefore should be completely separate from the protocol. However, this means that a dialogue protocol cannot be disseminated to an agent and automatically executed without human intervention.

The final issue concerns the synchronisation of agents within an institution. The performative structures and normative rules contain conditions, which restrict the behaviours of the agents. Similarly, the agent protocols rely on agents being aware of the current state of the institution, and where they are expected to interact. However, the enforcement of this synchronisation raises a number of questions when implementing these systems. The enforcement techniques which have been proposed [VSE02] rely on the use of administrative agents, or agent proxies to ensure the smooth running of the institutions. However, this is a central point of failure and is arguably contrary to the underlying principles of self-contained autonomous agents.

3 The MAP Language

In the preceding section we highlighted a number of issues which we have experienced with the EI framework. Nonetheless, it should be noted that EI provide a solution to many other problems encountered in the deployment of MAS. In particular, we consider the underlying concepts of institutions, scenes, and roles to be essential in the deployment of large-scale MAS. For this reason, the protocol language presented in this section retains these concepts, while redefining the core specification to alleviate the perceived shortcomings.

The division of agent dialogues into scenes is a key concept in our protocol language. A scene can be thought of as a bounded space in which a group agents interact on a single task. The use of scenes divides a large protocol into manageable chunks. For example, a negotiation scene may be part of a larger marketplace institution. Scenes also add a measure of security to a protocol, in that agents which are not relevant to the task are excluded from the scene. This can prevent interference with the protocol and limits the number of exceptions and special cases that must be considered in the design of the protocol. Additional security measures can also be introduced into a scene, such as placing entry and exit conditions on the agents, though we do not deal with these here. However, we assume that a scene places barrier conditions on the agents, such that a scene cannot begin until all the agents are present, and the agents cannot leave the scene until the dialogue is complete.

The concept of an agent role is also central to our definition of a dialogue protocol. Agents entering a scene assume a fixed role which persists until the end of the scene. For example, a negotiation scene may involve agents with the roles of buyer and seller. The protocol which the agent follows in a dialogue will typically depend on the role of the agent. For example, an agent acting as a seller will typically attempt to maximise profit and will act accordingly in the negotiation. A role also identifies capabilities which the agent must provide. For example, the buyer must have the capability to make buying decisions and to purchase items. Capabilities are related to the rational processes of the agent and are encapsulated by decision procedures in our definition.

We have previously stated that we have abandoned the graphical state-chart formalism of EI in favour of a language-based approach to dialogue protocols. Our formalism allows the definition of infinite-state dialogues and the mechanical processing of the resulting dialogue protocols. The semantics of our language are derived from the field of process calculus, and in particular the Calculus of Communicating Systems (CCS) [Mil89]. CCS was developed for the formal expression of concurrent and communicating processes, and appears to provide a good fit with the problems of inter-agent communication and synchronisation found in multi-agent dialogues. The use of process calculus also provides a good foundation for our future work on the verification of dialogue protocols. Nonetheless, CCS is a mathematical system and requires heavy sugaring to make it suitable for expressing dialogue protocols. Therefore, we have designed a language of
Multi-Agent Protocols (MAP) which we will now define.

The abstract syntax of MAP is presented in Figure 2. Agents are uniquely identified by a name \( a \), and have a fixed role \( r \) for the duration of the scene. A scene comprises a fixed set of roles \( R \), a set of participating agents \( A \), and a sequence of protocols \( P^{(k)} \). A protocol \( P \) can be considered a procedure where \( a, r, \phi \) are the arguments. The initial protocol for an agent is specified by setting \( \phi = \emptyset \) (i.e. \( k = 0 \)). Protocols are constructed from operations \( op \) which control the flow of the protocol, and actions \( \rho \) which have side-effects and can fail. The interface between the protocol and the rational process of the agent is achieved through the invocation of decision procedures \( p \). Interaction between agents is performed by the exchange of messages \( M \) which contain performatives \( \sigma \).

### Figure 2: MAP Abstract Syntax.

To illustrate the difference between the EI framework and the MAP language, Figure 3 shows a MAP implementation of the General Practitioner (GP) scene from Figure 1. We distinguish between the different types of terms by prefixing variables names with \( \$ \), role names with \( % \), and agent names with \( ! \). The MAP syntax enables us to specify the same dialogue as in the EI specification, with the advantage that we can now define additional properties such as the agent decisions.

We define two agents \(!Patient1\) and \(!Doctor1\) which have roles \( %patient \) and \( %doctor \) respectively. The protocol for the patient is specified separately from the doctor, though the two will interact closely. This differs from the institution model where a single shared protocol is defined for all agents in the scene to follow. The separation of protocols is a natural consequence of abandoning the state-based method of specification. Our language has no explicit representation of the dialogue state as the method of synchronisation between agents is purely through the exchange of messages.

When exchanging messages, through send and receive actions, a unification of terms in the definition \( agent(\alpha, r, \phi) = op \) is performed, where \( \phi \) is matched against the agent name, and \( \phi \) is matched against the agent role. For example, the request for an appointment in line 4 of the protocol will match any agent whose role is \( %doctor \). Similarly, the receipt of the request in line 17 of the protocol will match any agent whose role is \( %patient \), and the name of this agent will be bound to the variable \( \$patient \). We can therefore define broadcast and multi-cast communications. Furthermore, our example will scale when more that two agents are present in the scene.

The semantics of message passing in our language corresponds to reliable, buffered, non-blocking communication. Sending a message will succeed immediately if an agent matches the definition, and the message
1 GeneralPractitionerScene([%patient, %doctor], [{Patient1, !Doctor1},
2 agent(!Patient1, %patient) =
3 request(appointment) => agent(_, %doctor) then
4 waitfor
5 (accept(appointment, $appointment) <= agent($doctor, %doctor) then
6 ($symptoms = getSymptoms() then
7 inform(symptoms, $symptoms) => agent($doctor, %doctor) then
8 waitfor
9 (inform(refer) <= agent($doctor, %doctor) or
10 inform(norefer) <= agent($doctor, %doctor))
11 timeout (e)) or
12 reject(appointment) <= agent($doctor, %doctor))
13 timeout (e)
14 agent(!Doctor1, %doctor) =
15 waitfor (request(appointment) <= agent($patient, %patient)) timeout (e) then
16 ($appointment = makeAppointment($patient) then
17 accept(appointment, $appointment) => agent($patient, %patient) then
18 waitfor
19 (inform(symptoms, $symptoms) <= agent($patient, %patient) then
20 ($ref = doReferral($patient, $symptoms) then
21 inform(refer) => agent($patient, %patient)) or
22 inform(norefer) => agent($patient, %patient))
23 timeout (e)) or
24 reject(appointment) => agent($patient, %patient)

Figure 3: General Practitioner Protocol.

$M$ will be stored in a buffer on the recipient. Receiving a message involves an additional unification step. The message $M$ supplied in the definition is treated as a template to be matched against any message in the buffer. For example, in line 6 of the protocol, a message must match $accept(appointment, $appointment)$, and the variable $appointment$ will be bound to the second term in the message if the match is successful. Sending a message will fail if no agent matches the terms, and receiving a message will fail if no message matches the message template.

Communication is non-blocking in that the send and receive actions do not delay the agent. For this reason, all of the receive actions are wrapped by $waitfor$ loops to avoid race conditions. For example, in line 17 the agent will loop until a message is received. If this loop were not present the agent may fail to find an appointment request and the protocol would terminate prematurely. The advantage of non-blocking communication is that we can check for a number of different messages. For example, in lines 9 through 12 of the protocol the agent waits for either a refer or norefer decision. The $waitfor$ loop includes a $timeout$ condition which is triggered after a certain interval has elapsed. This can be useful in handling certain kinds of protocol failure, though we do not make use of timeouts in our example.

At various points in the protocol, an agent is required to perform tasks, such as making a decision, or retrieving some information. This is done through the use of decision procedures. As stated earlier, decision procedures provide an interface between the dialogue protocol and the rational processes of the agent. In our language, a decision procedure $p$ takes a number of terms as arguments and returns a single result variable $v$. The actual implementation of the decision procedure is external to the dialogue protocol. In effect, the decision procedure acts as a hook between the dialogue and the rational processes. For example, the $makeAppointment$ decision procedure in line 18 of the dialogue refers to an external appointment procedure, which can be arbitrarily complex (e.g. a timetabling application).

The operations in the protocol are sequenced by the $then$ operator which evaluates $op_1$ followed by $op_2$, unless $op_1$ involved an action which failed. The failure of actions is generally handled by the $or$ operator. This operator is defined such that if $op_1$ fails, then $op_2$ is evaluated, otherwise $op_2$ is ignored. For example, if the $doReferral$ procedure in line 22 fails, then a norefer message will be sent in line 24. Our language also includes a $par$ operator which evaluates $op_1$ and $op_2$ in parallel. This is useful when an agent is involved

6
in more than one action simultaneously, though we do not use this in our example.

External data is represented by constants $c$ in our language. We do not attempt to assign types to this data, rather we leave the interpretation of this data to the decision procedures. For example, in line 7 the symptoms are returned by the `getSymptoms` procedure, and interpreted by the `doReferral` procedure in line 22. Constants can therefore refer to complex data-types, e.g. flat-file data, XML documents, images.

It should be clear that MAP is a powerful language for expressing multi-agent dialogue. We have used this language to specify a wide range of protocols, including a range of popular negotiation and auction protocols. It is important to note that MAP is not intended to be a general-purpose language, and therefore the relative paucity of features (e.g. no user-defined data-types) is entirely appropriate. Nonetheless, dialogue protocols are executable, given a suitable communication platform and an appropriate definition of the decision procedures.

4 Semantics of MAP

The provision of a clean and unambiguous semantics for our MAP language was a primary consideration in the design of the language. The purpose of the semantics is to formally describe the meaning of the different language constructs, such that dialogue protocols expressed in the language can be interpreted in a consistent manner. We consider this to be a failing of the formal semantics of FIPA [FIP99], which is expressed in BDI logic. The FIPA semantics is an abstract description, which neglects practical aspects such as a definition of the communication primitives. Furthermore, the BDI modalities can be interpreted in a number of different ways, e.g. [Jen93], meaning that implementations of BDI agents have typically been ad-hoc in nature.

There are many different styles of semantics for expressing the evaluation behaviour of programs, commonly divided into operational and denotational styles. In colloquial terms, the operational style is concerned with defining how programs are evaluated, and the denotational style is concerned with what (mathematically) is being computed. Generally, the operational semantics is most useful for specification and implementation, while the denotational semantics is most useful for proofs of properties of a language. We consider the BDI semantics of FIPA to be a denotational-style semantics. By contrast, we have chosen to use a relational-style operational semantics for our MAP language. The form of semantics which we adopt is natural semantics [Kah87], so called because the evaluation of the rules is reminiscent of natural deduction.

\[ \begin{align*}
\Delta & \in \text{Agent Environment} \quad ::= \quad a \xmapsto{map} (r, AE, VE, PE, ME^{(k)}) \\
AE & \in \text{Agent Protocols} \quad ::= \quad \phi^{(k)} \xmapsto{map} op \\
VE & \in \text{Variables} \quad ::= \quad v \xmapsto{map} \phi \\
PE & \in \text{Decision Procedures} \quad ::= \quad p \xmapsto{map} \phi^{(k)} \\
ME & \in \text{Messages} \quad ::= \quad (a, r, M)
\end{align*} \]

Figure 4: MAP Evaluation Environment.

The natural semantics style is convenient because the entire evaluation of an agent dialogue can be captured within a (semi-)compositional derivation that can be reasoned about inductively. The rules of the semantics can be implemented directly (e.g. as Prolog Horn clauses) and a derivation can be performed incrementally (depth-first) from the root to the leaves. In natural semantics, we define relations between the initial and final states of program fragments. A program fragment in MAP is either an operation $op$, or an action $\alpha$. The state is captured by an agent environment $\Delta$ which is defined in Figure 4. The environment contains an n-tuple for each agent comprising the agent role $r$, the agent protocols $AE$, the bound variables $VE$, the decision procedures $PE$, and a message queue $ME$. The agent protocols $AE$ map from arguments $\phi^{(k)}$ to operations $op$, where an empty sequence of arguments is the initial agent protocol. The decision procedures $PE$ are represented as a map from the procedure name $p$ to the argument terms $\phi^{(k)}$. The message queue $ME^{(k)}$ is a sequence of n-tuples $(a, r, M)$, where $a$ and $r$ are the name and role of the sender, and $M$ is the actual message. For brevity we omit the rules for constructing the initial environment, and for checking well-formedness of the environment from our definition.
of messages between agents we assume that the environment $\Delta$ is shared between agents. Thus, sending a message to an agent is captured by placing the message into the message queue $\xi$.

We define the evaluation rules for the program fragments of MAP in Figure 5. To capture the exchange of messages between agents we assume that the environment $\Delta$ is shared between agents. Thus, sending a message to an agent is captured by placing the message into the message queue $\xi$ of the recipient.

**Figure 5: MAP Operational Semantics.**

We define the evaluation rules for the program fragments of MAP in Figure 5. To capture the exchange of messages between agents we assume that the environment $\Delta$ is shared between agents. Thus, sending a message to an agent is captured by placing the message into the message queue $\xi$ of the recipient.
Rules 1 through 6 define the evaluation of the different types of operations \( op \). The form of these rules is \( \Delta, a \vdash op \Rightarrow \Delta' \), where \( \Delta \) is the state at the start of evaluation, \( a \) is the name of the agent performing the evaluation, \( op \) is the operation, and \( \Delta' \) is the state on completion. Similarly, rules 7 through 10 capture the evaluation of the actions \( a \). The form of these rules is \( \Delta, a \vdash \alpha \Rightarrow \Delta' \), which is as before where \( \alpha \) is the action. We also define a substitution function, \( VE \vdash subst(\phi) \Rightarrow \phi' \) which substitutes variables for their values, and a unification function \( VE \vdash unify(\phi_1, \phi_2) \Rightarrow VE' \) which matches terms and binds variables to values. The \( VE \vdash eval(p, v) \Rightarrow VE' \) function evaluates the external decision procedure \( p \), binding the result to \( v \) in \( VE' \).

The rules in Figure 5 are applied from the bottom-left in an approximately clockwise manner. For example, Rule 2 defines the evaluation of the sequence \( op_1 \text{ then } op_2 \). In order to evaluate this sequence, \( op_1 \) is evaluated in the environment \( \Delta' \) which yields the environment \( \Delta'' \) as the result. This is followed by the evaluation of \( op_2 \) in \( \Delta'' \), where the resulting environment \( \Delta''' \) is passed by the whole rule as the result. It is clear that the evaluation of \( op_1 \) and \( op_2 \) will involve further rules from the semantics, resulting in a derivation tree. The application of these rules to a dialogue protocol will result in a very large derivation tree which denotes a complete evaluation of the protocol.

### 5 Conclusions and Further Work

In this paper we have defined a novel language for representing dialogue protocols in Multi-Agent Systems. Our language of multi-agent dialogue protocols (MAP) fills an essential gap between the low-level communication and high-level reasoning processes found in such systems. The language is founded on process calculus and is expressive enough to describe a large range of agent protocols. The language also addresses a number of issues which we faced in the implementation of the Electronic Institution (EI) specification of dialogue protocols. For example, by removing the finite state-chart representation of scenes, we overcome a number of issues associated with the dissemination and synchronisation of dialogues.

Dialogue protocols specified in the MAP language are designed to be directly executable by the agents participating in the dialogue. To this end we have presented an operational semantics for the language, which precisely defines the evaluation behaviour of the language. Our presentation in the natural semantics style enables a direct implementation of the evaluation rules of the language. We have implemented these rules directly as Prolog Horn clauses using LINDA for inter-agent communication. We have also implemented the MAP language in Java using concurrent threads for the individual agents. To implement the evaluation rules in Java we have defined an interpreter which provides the necessary back-tracking and unification behaviour. The decision procedures are native to the agent in both implementations, and in principle (given a suitable communication platform) agents from both should be able to inter-operate through MAP dialogues, though this remains future work.

The implementation of MAP in Java uses an XML representation of the dialogues protocols. This choice of representation is related to our intention to use MAP as a mechanism for specifying dialogues between web services. Web services [BHM+03] refers to a new group of agent technologies and appear to be a viable successor to the FIPA-ACL/BDI model as a method of specifying and constructing MAS. A web service can be viewed as an abstract specification for a kind of functionality, which is subsequently implemented in a concrete agent. Web services are specified in the web service description language (WSDL) and communication between web services is achieved by the Simple Object Access Protocol (SOAP). At present, communication between web services is limited to simple call/return behaviour. However, we believe that MAP would enable complex interactions between distributed web services.

In this paper we have stated that the verification of dialogue protocols is an important consideration. Dialogue protocols specify complex asynchronous and concurrent interactions, and therefore it is difficult to design correct protocols. Our experience with defining protocols in MAP has shown that predicting undesirable behaviour is a non-trivial task. We can obtain a measure of confidence in a protocol through repeated simulation of a protocol. However, this is imprecise and can fail to catch many of the problems which may be present in the protocol. To address this issue we are currently investigating the use of model-checking techniques [CGP99] to perform automated verification. The appeal of this approach over simulation is that an exhaustive exploration of the dialogue space is performed. Our early experiments with this technique have shown a high success rate in the detection of failures (e.g. non-termination) in our dialogues.
References


