Implicit Commitments through Protocol-Level Semantics [*]

Keywords: Social Agents, Conversational Policies, Commitments, Social Norms, Standards for Agents

Gregg Economou Maksim Tsvetovat Katia Sycara Massimo Paolucci

ABSTRACT

Conversation policies often incorporate the notion of commitment as an elementary building block of agent interactions. However, high-level commitment semantics rely on heavy infrastructure and advanced reasoning capabilities which then become inseparable from agent interactions, a lower-level concept.

We show that it is possible to express the notion of commitment as a part of protocols themselves. We present a simple mechanism for implicit commitments based on finite state machines, and reason about actions an agent is required to fulfill to follow the state machine.

We also demonstrate the utility of implicit commitments within an interaction- oriented programming environment and present a toolkit that implements this scheme, allowing for dramatic savings in MAS development time.

1. INTRODUCTION

A number of theories of commitment have been developed which rely on the concept of agent beliefs, desires and intentions [12]. However, we believe the concept of commitment is a more fundamental one, bound instead to an agent's social interactions and communicative acts. In this paper we show that implicit commitment concepts derivable from communications are sufficient and guaranteeable even in a heterogeneous environment of self-interested agents, thus obviating the need for explicit commitment mechanisms.

Commitments are needed to provide to other agents a predictive indication as to the future activities or state of the agent that enters such a commitment. Forms of commitment include acceptance of potential consequences of breaking such a commitment, or simple assertions as to the future regarding the agent. To other agents, the knowledge of a commitment provides a guidance to predict characteristics of future interactions. The primary focus of this paper is to derive commitments from communicative actions of the agents independently of their internal states.

We approach the issue from two sides: the first, concerning an agent's guarantees and assurances of available information about other agents, and second, concerning the meaning of commitment.

The most commonly considered environment for an agent is a cooperative one, in which other agents can be reasonably expected to be following some common interest and generally be honest to one another. Explicit expressions of intent, commitment, desire, and belief then can be used as a reasonable basis of inter-agent knowledge-sharing and agents can rely on such information with a considerable degree of confidence.

The usual issues encountered in cooperative environments are those concerning expression of such semantics-dense concepts, synchronization of belief and intent databases among agents. It is common to presume that if there was a sufficient means of declaring one's commitments, other agents would be able to take down these announcements and reason about them. Thus, most research effort has focused on formal ways to express commitment.

However, it is readily apparent that multiagent systems are increasingly going to involve self-interested agents and whole communities of competitive agents. Breaking of commitments and distribution of misleading information is par for the course in such an environment. In a universe of competitive agents, the trust implicit in a cooperative system vanishes. Agents expressing intention and belief are to be suspected or trusted along a gradient from trust to complete distrust. Ultimately, they can only be judged on experience, with no support from a pervasive infrastructure.

In a cooperative environment, when an agent expresses an intent or a commitment, this expression can be assumed to convey the meaning that the agent itself believes those assertions. In a competitive environment, however, the best one can assume is that that agent wishes others to believe that he believes those assertions - and creation of such belief is somehow strategically beneficial to it.

[*]This material is based on work supported in part by MURI contract N00014-96-1222 and CoABS Darpa contract F30602-98-2-0138 and NSF grants IRI-9612131 and IRI-9712607

In light of this, we have to rethink the concept of an explicit commitment infrastructure. As we show below, explicit commitment mechanisms are not needed to enjoy the benefits of reasoning about commitments, and these benefits can still be reaped in a competitive environment.

Since no assumption can be made on the agents "mental state", no assumption can be made on their reasoning process either. Throughout the paper we assume that we have no knowledge of the problem solving mechanisms that any agent follows. Agents will be just black boxes that are expected to follow a protocol. Any model of what the agent thinks or knows will be exclusively based on the messages exchanged. This assumption has two important consequences: first, we cannot assume that what an agent says corresponds to what the agent thinks. Ultimately, sincerity assumptions made by virtually every theory of agent communication [4, 5, 3, 11] is not guaranteed to hold, quite the opposite: agents should be assumed to be un-trustworthy. The second consequence is that the results of this paper apply to any agent architecture and hold for any heterogeneous MAS.

Commitments are generally broken in one of three situations:

- agent is malfunctioning or buggy, or is incorrectly implemented,
- agent has incomplete information about side effects of certain actions.
- agent is deliberately malicious,

In the first case, the agent may be able to gracefully recover from the malfunction and continue operation. However, it is still up to the agent developer to make sure that the agents comply with the specified protocols.

In the second case, detecting a broken commitment in learning what the agent does not know and needs to learn from its environment.

In the third case, there is not much that an agent can do to remedy the situation. However, if certain agents are known to be consistently malicious, they will eventually be ostracized from the agent community (using some agent reputation mechanism - which is outside the scope of this paper).

The paper is organized as follows: In Section 2, we discuss how the concept of commitment is connected to the concept of conversation policy. We then proceed to show an algorithm for inferring commitments from a given DFA-based conversation policy, through the concept of deontic states. To allow the DFA-based approach to be more firmly rooted in commonly used speech act semantics, we then proceed to illustrate commitments as implicitly defined by speech acts. We finish with a presentation of an implemented toolkit for conversational policies with implicit commitments.

2. RELATED WORK

It has been well-established [13, 6] that any inter-agent communication or conversation can be described by a protocol, for which there exists an automaton that formally defines the possible interactions in this communication.

In particular, M.P. Singh [11] has introduced a paradigm called *interaction-oriented programming*, and the notion of agent skeletons. Agent skeletons are state machine-like automata that define behaviors of individual agents by specifying the sequences of external events but does not specify the problem solving process that drives the agents. A similar concept, called a *multi-plane state machine* was introduced by L. Bölöni and D.Marinescu [1, 2] as part of the Bond toolkit for development of multi-agent systems.

J.Pitt and A.Mamdani [8] have proposed an architecture where the agents derive their intentional semantics through protocol-driven interactions. They define agent reasoning as a layered process, consisting of:

- Action-level semantics, which is concerned with reacting in appropriate ways to the received messages and is external to the agent.
- Content-level semantics, which is concerned with interpretation and understanding of content of a message, and is internal to the agent.
- Intention-level semantics, which is concerned with maintaining an internal belief-desire-intention system and making sure that the communications are consistent with it.

In our paper, we are mainly concerned with the action-level semantics and agents' abilities to determine higher level semantic meanings - such as commitments - merely from observed communications.

M.Colombetti [3] has recently proposed a model of agent communications that relied on the deontic logic to derive the commitment-based semantics of speech acts. The model deals with conversations as a sequences of pre-commitments and contracts. Every speech act is defined in terms of one or more agents committing to perform an action or to assert a value of a preposition. However, the speech act theory still relies on mentalistic assumptions that may or may not be valid in a non-cooperative environment.

3. CONVERSATION POLICIES IN A COMPETITIVE ENVIRONMENT

In a real-world heterogeneous multiagent system, agents will not always be in communication with friendly, cooperative agents. Very often agents will be self-interested and in competition with others. The concept of an agent's commitment to some future condition is still an important one, perhaps even more so in a system of self-interested agents since agents can assume so much less about others and reputation becomes a more important issue. Also, the nature of commitment infrastructures must be rethought when one can no longer presume cooperative intentions from other agents. Most current work on commitments depends at one point or another on some structure for verification and enforcement of commitment policies. In a system of noncooperative agents, there is no better guarantee that agents will obey the framework for commitments than they will obey their commitments themselves.

In a competitive environment, an agent cannot be guaranteed of any special knowledge about another agent's real intentions, future activities, or prior history. All an agent can know about another agent is what it can derive from that other agent's communicative acts. There is no other source of information that any agent can be guaranteed of save that which the agent who is the object of speculation itself send out into circulation. Agents need to employ some reasoning methods to determine some higher-level propositions regarding some other agent, based on information gathered from its interactions.

These interactions, as a sole output of an agent, are also the sole manifestation of its commitments: commitment is a social mechanism.

4. MANAGING ACTION-LEVEL COMMITMENT SEMANTICS VIA A PROTOCOL

Let us define an agent interaction protocol P as a set of I/O automata a_i such that each automaton $a_i \in P$ represents a role that an agent can play (e.g. In an auction protocol, such roles are auctioneer, bidder and observer). Let us also assume that automata comprising P are compatible: the outputs of one automaton are suitable inputs for one of the others.

Compliance with or keeping one's commitments is merely a matter of following legal transitions in the automaton of the chosen protocol. Any agent involved in the conversation can tell if another agent it is conversing with is following the protocol by merely observing the communications. It cannot necessarily know the exact state of the other agent(s), because it cannot know what communications and input the other has received from sources other than itself, but it can know, for any interaction, if the interaction is compliant with the protocol.

Not only is this knowledge sufficient, it is the most an agent is guaranteed to know in a non-cooperative environment.

What of super-protocols which contain or refer to other protocols, and are structure for moving about from one protocol to another?

If an agent was to violate the protocol and thus default on its commitments (or some subset of them) by changing its state through some illegal transition, it would no longer be in accordance with the protocol, and its interactions would not match those interactions which are expected by others following the protocol. Thus, other agents will be able to detect that this agent has broken its commitment, and its implied obligation to follow legal transitions to predictable future interactions.

Since any super-protocol can be broken just as easily as the protocol it seeks to enforce, there can be no protocol for breaking protocol. Thus, the most one can truly depend on, and simultaneously all one really needs to learn, can be learned from the automaton, the transition history, an agent's observation of other agents and its assessment of whether or not protocol is followed.

4.1 Protocol-Level Semantics

The action-level semantics described by J.Pitt [8] has a lot in common with the protocol-based semantics that we are proposing.

Pitt attaches a deontic obligation between conversationally linked message input-output pairs. An agent that receives a message with a performative of a certain type, is thus obligated to reply with a message of a prescribed performative as dictated by conversational policy. He goes further to point out a relationship between *intention-level* semantics (which are relevant to the internal activity of an agent in responding to and handling of an incoming message) and the *action-level* semantics of the communication.

However, action-level semantics are limited to specification of input-output pairs, describing statelessly sets of appropriate interactions. Such input-output pairs are not sufficient to describe the interactions of most nontrivial agents. More specifically, any interaction where the same performative is used more than once is in danger of being improperly described by such a scheme. This is because simple pairing is too inflexible and fails to take into account changes in the internal state of an agent during the course of a conversation. They fail to sufficiently differentiate serialization and ordering of operations, branching of conversation flow, and iteration through some portion of a conversation without reverting to a workaround such as a greatly increased proliferation of performatives.

To increase the expressive power of conversational policies to describe more complex interactions, one must extend the action-level semantics to encompass interaction protocols by adding stateful conversations between agents. *Protocol-level* semantics extends the simple reciprocal commitment to encompass whole paths of arbitrary length and complexity through a state machine, including loops, and nested interactions. The protocol-level semantics inherits the inferable intention-level properties of action-level semantics and binds them to complex stateful interactions of agents instead of performative pairs.

5. THE UTILITY OF DETECTING COMMITMENTS

In a system of self-interested agents, it is in the interest of agents to achieve successful communication. If a self-interested agent plans to take advantage of other agents, it must communicate with them to achieve that end. Because protocol is the basis of nontrivial communication, it is generally in the interests of agents to communicate truthfully about protocols, even if they don't intend to follow them with sincerity in mind.

With this in mind, we do make the assumption that since all communications follow protocols, an agent can expect to have access to descriptions of the protocols as DFAs.

The consequences, however, are that either communication cannot take place for practical purposes (things simply don't work because input and outputs don't match) or because Y

might detect through its interactions that X's real DFA is not what X said it was. The computation of commitments is more important then for Y to know ahead of time what it's getting into and determine what commitments X would take upon itself in exchange.

5.1 Example

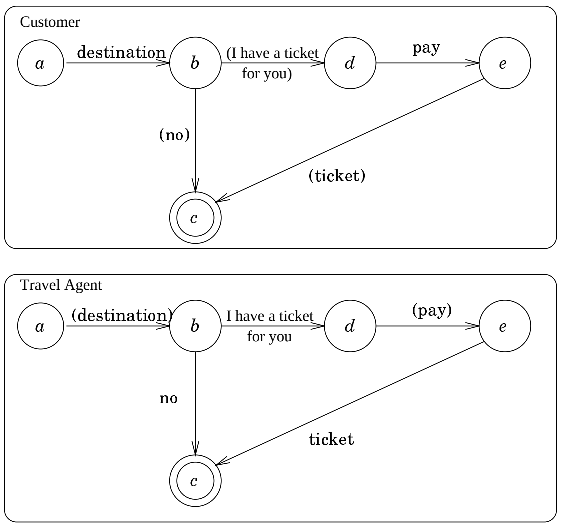

Figure 1: Travel Agent and Customer DFAs

Let us take up an example of an interaction protocol implemented by a well-known travel web site. In Figure 1, Agent X is looking to buy a plane ticket. Agent Y is a travel agent, advertising a capability to sell plane tickets. X approaches Y and asks Y 'how do I buy tickets from you?'. Y sends a protocol description to X, who then analyzes it. The protocol basically involves the following interactions:

The customer supplies the travel agent with a city and date, and the travel agent then will say if he has a ticket to the city or not. If the agent doesn't have a ticket, then he mentions so and ends the conversation. If the agent does have a ticket, he says that he has a ticket, the customer pays him, and the agent gives the customer the ticket.

This seemingly innocent protocol has an obvious flaw: the customer cannot choose to not buy the ticket once he has requested a destination. If the protocol had just an extra transition to allow the customer to evaluate the ticket and decline it before paying, then X would have a way out. X can use some algorithm, (for example algorithm described below6.2), to find possible commitments in this new DFA.

Once X has examined this, he would see that there is a rather serious commitment to buying a ticket just from asking for a destination, with no way out. In a case like this, if the ticket Y tells X about is not satisfactory, X is stuck either buying an undesirable ticket or being forced to break his commitment, perhaps by breaking off communication with Y, or by sending Y communications that are inappropriate for the DFAs that are running at the time. If there was a reasoner in X, it might decide that Y is taking advantage of the fine print.

X also might decide that this protocol is unfair, and choose not to involve himself at all with Y, and avoid broken commitments by finding someone more agreeable to talk to.

Though nothing would stop X from entering the protocol and then breaking it, thus breaking his implied commitment to pay for a ticket, X can avoid the matter entirely before getting involved.

6. INFERRING COMMITMENT FROM CONVERSATIONAL POLICY

In most inter-agent communications, if a protocol is to be followed, then any interlocutors involved need to have some common knowledge of this protocol. Effectively, each has access to an automaton that implements the protocol or the portion of the protocol that that agent needs to participate in.

Since the only thing an agent can truly discover about another agent is what it can gather through its interactions, commitments really serve to predict future interactions of a given agent. Since such interactions follow protocols which are described by some automaton, what a commitment really can be said to encapsulate is a prediction about some portion of an automaton that an agent is bound (committed) to traversing and can no longer escape.

For example, once a bidder submits a bid in an auction, he must assume he is liable to win the auction and pay the amount of his bid. There exist transitions out of this path (e.g. the rejection of the bid by the auctioneer) but such transitions depend on some entity external to the agent and cannot be included in the calculation of what is a committed section of the automaton.

6.1 Deontic States

States where such commitments are entered are commonly called *deontic states*. When an agent needs to choose a correct path through the states of the conversational policy, he needs to be able to detect deontic states and reason about obligations that it will have once it enters such state.

We have developed a fairly simple rule for detecting deontic states in a given DFA: *A state is a deontic state with respect to some action A when at least one of the transitions out of it leads to inevitable execution of that action. Once this transition is taken, it is no longer in the power of the agent to stop the action from happening (although other agents can still cancel the transaction). Once the agent has taken such a transition, it is said to have committed to performing the action A.*

For example, let agent A be at State 1 (see Figure 2). By taking the *accept* transition to State 2, A can no longer cancel the transaction. If B, the second party to the contract, also accepts, it is unavoidable that agent A has to pay. Therefore, it is no longer in power of A to break the agreement and one can conclude that A has committed to executing the action at State 3.

It is important to make the distinction between transitions which depend on some external activity (such as the recep-

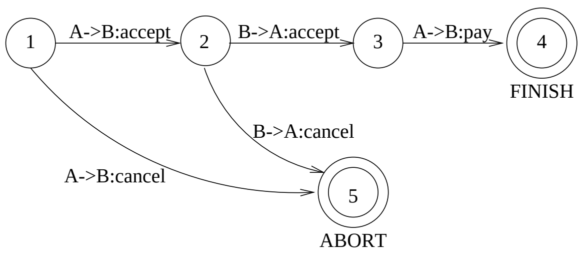

Figure 2: A state diagram illustrating a deontic state.

tion of a message from another agent) and those which can be influenced solely from within the agent, such as a test of some boundary condition (e.g. determine if the bid price is too high and quit the auction) or some reasoning entity's decision making or planning results. This is because an agent cannot predict with any surety at all that any transition that it does not fully influence can be taken, and thus must use this distinction in analyzing the automaton for potentially 'unavoidable' paths or regions of the automaton that the agent incurs the risk of entering by taking any given transition.

It is thus possible, given an automaton, the current state, and the history of transitions within the automaton, to determine if any portion of the reachable states in the state machine imply commitments or unavoidable actions on the part of the agent, as we will show in section 6.2 . Moreover, it is possible to detect and mark all deontic states within an automaton prior to the execution of the automaton.

Thus, if an agent has access to state machines on both sides of the protocol (as in the case of publicly posted protocol specifications), it would be able to infer fairness of the protocol with regard to the obligations of parties. (e.g is it possible that agent A will have to commit to paying for a service while agent B is not committed to provide the service?). It is often only necessary to search the part of the protocol that deals with execution of the transaction - since both parties have ways to bail out during negotiation.

6.2 An algorithm for finding deontic states

In this section we present an algorithm that, given a finite state automaton representing the protocol, can find and isolate commitments that the agent will have to make during execution of the protocol.

A commitment within a protocol is a path through the protocol's automaton defined by the following parameters:

- commitment origin state (i.e. the state where the commitment begins)
- terminating state (the state where the commitment ends)
- action with respect to which this commitment is active

By default, all actions that result in sending of messages are considered potentially leading to a commitment. It is then up to the agent's planner component (if it has one) to determine if a particular commitment is important within the agent's environment. However, to reduce the running time of the algorithm, it is possible to impose stricter conditions on the actions (e.g. actions leading to sending of a message to a non-trusted party).

In Algorithm 6.1 we show a recursive algorithm for finding implicit commitments given a finite state automaton for a particular agent.

The algorithm scans backwards through a DFA from some state q, and finds all sequences of states that represent commitments in the subset of the DFA that can possibly precede q.

The CFind algorithm 6.1 is passed three arguments: A state q to begin its search from, a number *depth* which keeps track of the recursion depth a particular call to the algorithm is made at, and a pending commitment *pending* which is an ordered pair indicating the transitions which respectively originate and terminate a commitment. The originating transition enters the deontic state for the destination state of the terminating transition. Its sole input aside from the arguments is a representation of a DFA of which q is some state. It returns either zero or one, depending on whether changes were made to *pending*, and possibly also outputs pairs describing commitments. These have the same form as *pending*. In our example, we add them to some imaginary list of true commitments. When the algorithm is first run, an empty pair is passed to *pending*.

Within a frame of recursion, the algorithm keeps track of various transitions it needs to recurse through (potentially), using a variable t to refer to a given transition. A simple boolean flag, $flag$ is used to save a return value, which is influenced at various places in the course of execution. States have a mark to indicate that they have already been traversed; the mark is written with the depth of the frame that traverses it, so that we can allow deeper frames to re-traverse but frames above to not pas through any state more than once.

The algorithm starts out by incrementing the call depth it was passed, so that it now reflects its proper depth. Then it examines the state q's mark, and if its depth is not greater than the mark, the call returns with a zero value. If it continues, then it loops on any transitions t it finds leading into q. It scans depth-first for the potential initial or final transition of a commitment, which means scanning for transitions which send messages. When it finds one, it examines the values in *pending* to see whether it should update a pending commitment or if *pending* is empty, to set *pending*'s terminating transition to t. If *pending* already has a terminator set, then t is possibly a previous state that leads to the terminator unavoidably, that is, t potentially commits to the transition in *pending*'s terminator. In either case that *pending* was modified, we set the return value to one. Then the algorithm recurses, calling the algorithm on the origin state of t, with the new *pending* passed to the child frame, and wait on its return.

Since the algorithm returns with a one if it caused modifi-

Algorithm 6.1: CFIND(State q, int $depth$, Commitment $pending$)

```
local transition $t$
local int $flag \leftarrow 0$
$depth \leftarrow depth + 1$
if $q.mark \geq depth$
  then return (0)
$q.mark \leftarrow depth$;
for each transition t that enters state q
  do
    if $pending \rightarrow terminator = true$
      then
        $pending.origin \leftarrow null$;
        $pending.terminator \leftarrow null$;
        $flag \leftarrow 1$;
    if $t$ is a sendmessage transition
      then
        if ($pending.terminator \neq null$)
          then $pending.origin \leftarrow t$
          else $pending.terminator \leftarrow t$
        $flag \leftarrow 1$;
    comment: Recursive execution
    if $CFind$(origin of t, $depth, pending$)
      then
        $flag = 1$;
        if ($pending.origin \neq null$ and $pending.terminator \neq null$)
          then
            $pending.terminator \leftarrow pending.origin$
            $pending.origin \leftarrow null$
      else
        if $pending.origin \neq null$ and $pending.terminator \neq null$)
          then
            $add - to - true - list(pending)$
            comment: We have found a valid commitment here
            $pending.origin \leftarrow null$
            $flag \leftarrow 1$
$depth \leftarrow depth - 1$
return ($flag$)
```

cation to the value of *pending* passed to it, and zero if it does not. If a zero is returned to it, it knows that down that path there were no further modifications to *pending* and it must be as complete as it can be from this state. If it is a complete commitment, that is, if it has both origin and terminator values, it is considered a true commitment and added to some list of found commitments. Then *pending*'s origin value is zeroed to render it incomplete again, so that other passes through the loop on t could find alternate paths that share a common terminating transition with *pending*.

When the algorithm has finished recursing and the final frame returns, the contents of the true-list will be all committed pairs of transitions in the portion of the DFA that precedes the initial q passed to the top frame. Because it only finds sequences in the portion of a DFA that can lead to q, the algorithm may potentially need to be run on more than one state of a given DFA.

If all states can eventually lead to the exit state (i.e. the DFA has no infinite loops), q only needs to be the exit state. However, if there are isolated loops in the state machine, entered but never exited, then the algorithm must be run for at least one state in each such loop.

Trivially, one could simply run the algorithm for every state in the state machine except the start state but the application of it to the minimal set of states would be more efficient for most state machines. The start state is excluded because it is obviously not party to any commitment meaningful inside the given state machine. The computational complexity of the algorithm per state q is $O(n^2)$ where n is the number of states in the DFA, and the worst case complexity for the complete DFA is $O(n^3 - n)$.

The algorithm builds a list of commitments it finds completed as 'true' commitments. The final list can potentially contain commitments which are totally contained inside other commitments; this is fine. These sub-commitments could be considered either redundant and thrown away or they could be considered important for the reasoning entity to think about as well; it is up to the implementor.

The algorithm has been tested on a variety of DFAs, including examples containing the three general cases of DFA topology; a simple, one-commitment DFA with no looping

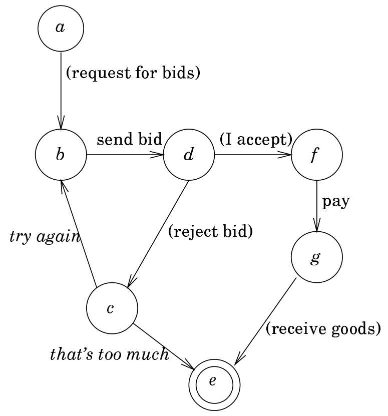

Figure 3: A simple bidder DFA.

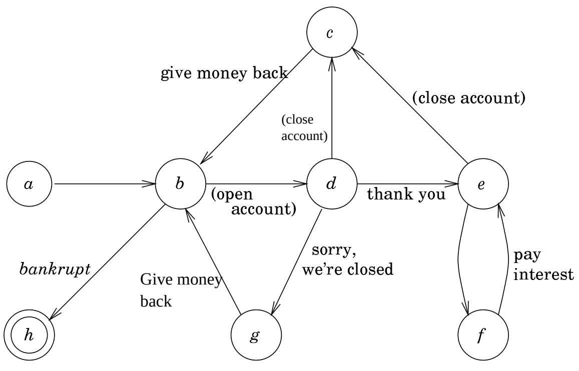

Figure 4: A Bank DFA (with loops).

commitments (see Figure 3), a DFA with connected loops (see Figure 4), both open and closed, and a DFA with an isolated loop (see Figure 5).

In Figure 3, there is a commitment upon sending a bid to pay. It is potentially unavoidable that the state machine will continue along to the post-payment state if one takes the bid transition. Though it is just as likely that the DFA will receive a rejection to its bid, the payment is the only edge that has some potentially unavoidable precursor-chain that begins with the sending of a message, i.e it is the only transition that involves an outward (social) action by the DFA and is potentially unavoidably taken as a result of taking some other transition which did an outward social action.

In Figure 4, we have a slightly more complex case: one of the commitments here is a recurring commitment, a loop that in this case does not necessarily exit. Also, this is an example of a DFA with multiple commitments. It implements a simplistic bank, which takes a deposit from some customer to open an account, and pays interest on that deposit until the customer closes his account, at which time the customer gets his deposit back. The bank also will hand a customer back his deposit if he changes his mind, and also gives it back if the bank doesn't want to do business at that time. Otherwise, the bank computes interest in f and the customer gets regular interest payments. Algorithm 4.1 detects these commitments without a problem. The bank is com-

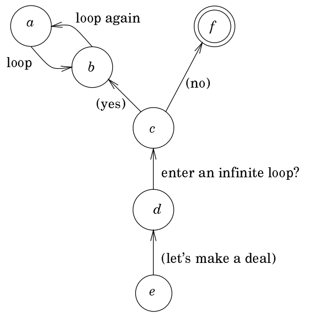

Figure 5: An inescapable commitment

mitted to giving the customer his money back if the bank says that they're not taking any business at the moment, and the bank is also committed to paying interest once they do accept the customer's deposit (shown where the bank thanks the customer for the deposit). Note that the commitment to pay interest only continues until the customer wishes to close his account, and it is only the initial 'thank you' that commits.

In Figure 5, we illustrate the final topological form that such a DFA can take: an infinite loop. This protocol is never really finished. This agent makes an offer to someone else, who decides the outcome. He cannot control if the offer will be accepted or rejected, but he is in danger of getting into an inescapable commitment once he makes the offer.

7. IMPLICIT COMMITMENTS WITHIN AN AGENT-FACTORY TOOLKIT

We have implemented a toolkit to assist in automatic generation of protocol-compliant agent communication code, utilizing the protocol-level semantics of conversational policies and commitment evaluation algorithms. The toolkit allows a protocol designer to formulate the agent interaction independently of the agent implementation process, in a manner similar to Interaction-Oriented Programming [10].

The Interaction-Oriented Programming paradigm defined by M.Singh is a layered construct that separates the reasoning components of the agent from the underlying communications infrastructure by means of a middle layer that incorporates explicit conversation policies.

The AgentFactory toolkit provides the infrastructure required for building an explicit conversation policy layer. The toolkit consists of several modules:

- An XML language for expressing state machine-based conversational policies
- Conversational policy verification tools, consisting of
  - Compile-Time verifier which asserts that the state machines comprising the conversational policy are well-formed and contain no unreachable or undefined states (problems that are often due to simple programmer errors)
  - State Machine verifier which determines whether a given state in a state machine can be reached

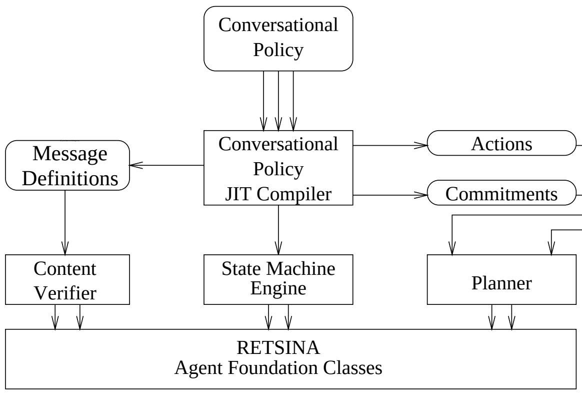

Figure 6: The IOP Toolkit as a Part of RETSINA.

through a sequence of communicative acts - thus determining whether or not a protocol is fit for use in a particular context

– Commitment verifier which uses the CFind algorithm described above to determine fitness of a particular protocol given constraints on commitment fairness

- A re-targetable conversation policy compiler, generating Java and C source code for agent interaction skeletons.
- Automatic generator for ACL-independent message syntax verification code.
- Agent Linker which links interaction skeletons generated by compilers with predefined actions and planner objectives.

The toolkit also provides the following functions:

- A convention for action bindings allowing a protocol designer to demand particular support activity from the agent implementation as necessary to meaningfully implement the protocol
- A convention for recursively including protocols within other protocols to enhance reusability, passing information from protocol exit states to other, 'outer' protocols
- Methods for defining protocol-wide transition policies
- Automatically generated message verification code.

However, it should be noted that while AgentFactory provides a set of convenient tools and infrastructure to develop agents that faithfully follow pre-defined protocols, it is still up to the agent developer to design his agent in such way that it would be compliant with not only the letter but also the spirit of the protocol.

The AgentFactory toolkit has been designed as a part of the RETSINA Agent Foundation Classes, which provide functionality necessary for agent development - such as a communication infrastructure, capability advertising services and matchmaking services. The agent infrastructure includes a general purpose reusable planner [7] which can be programmed to reason about commitments and influence its planning decisions based on its conclusions.

The conversational policies are expressed in XML format, and given to a Just-In-Time compiler. The compiler transforms the conversational policy into an executable state machine, and runs the CFind algorithm to isolate commitments, actions and message format definitions. The commitment and action definitions are then added to the planner's environment and further processed through the planner's high-level reasoning capabilities.

However, it is key that the reasoning layer of the agent is not completely separated from the conversational policy. As mentioned above, it is necessary to reason about the deontic states of the protocol in order to determine current and future commitments.

In the ideal situation - when the planner has all domain knowledge it needs to reason about the actions and commitments - the agent can aquire a new conversation policy, process it and start communicating through it on the fly. In the case where the domain knowledge is insufficient, the planner raises an error and prompts for the agent developer to augment its knowledge base.

The classical one-sided relationship between high-level reasoning entities and low-level automaton-based interaction models is eased in this system by allowing bi-directional control flow between planning and protocol components. When appropriate, reasoning drives the agent's traversal of the protocol automaton, and when appropriate, the protocol, following its state machine, modifies the reasoning system's goals and environment.

Remaining true to the semantics of a particular interaction remains the burden of the protocol designer, but those protocols are then at the disposal of agent designers. With the higher-level concept of commitment expressed in the protocol implementations themselves, the agent designer is able to more conveniently reason about those issues in a competitive, self-interested environment which much more closely resembles the future of multiagent systems.

7.1 Agent Development with AgentFactory

The process of agent development using AgentFactory is as follows:

1. *Write Conversational Policy* by writing a state-transition diagram of the interactions to be implemented, then converting it to the XML format. In the future, graphical programming tools will be developed to make this step more intuitive.
2. *Run Verifiers* to determine whether the conversational policy contains errors or is unfit for use.

   The process may have to be repeated until a suitable conversational policy is developed.

   Once the policy passes verification and is determined to be suitable for use, it can be distributed to agent

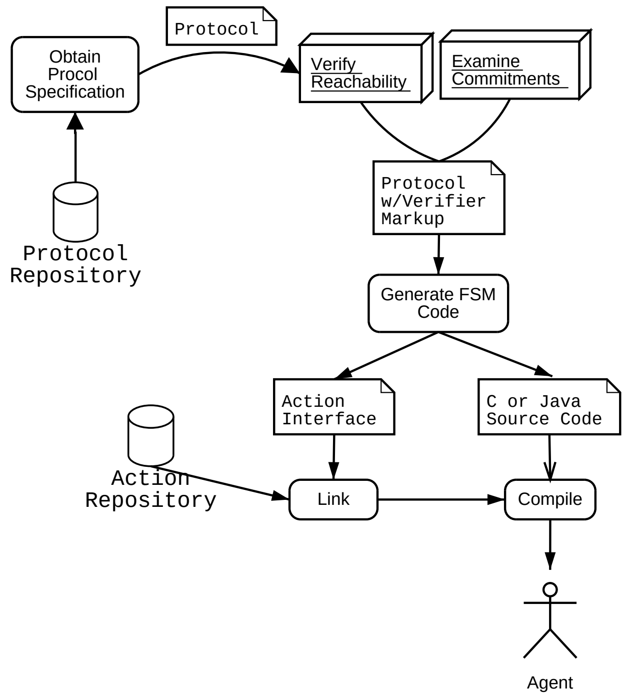

Figure 7: Agent Development with AgentFactory.

developers. However, by ensuring that all interactions are specified in advance, the AgentFactory toolkit dramatically decreases time spent on development and debugging of interactions.

3. *Compile Interaction Skeletons* by running the Conversational Policy Compiler. Among the generated code will be an action interface - which specifies what actions are required to be implemented by the agent developer

4. *Fill the action interface* with references to library functions or planner objectives

5. *Re-run the Conversational Policy Compiler* to link the interaction skeleton with the action libraries

6. *Compile the resulting source code* with a Java or C compiler.

The resulting agents allow the agent programmer to implement and debug agent interaction separately from the reasoning process. They can be used standalone or as a part of a larger agent.

The conversational policy compiler is a command-line tool that can be easily integrated with project management tools such as *make* or graphical development environments such as *JBuilder* or *Visual C++*.

7.2 AgentFactory in Action

The toolkit has been used on several multi-agent system projects and has been proven to dramatically decrease development time.

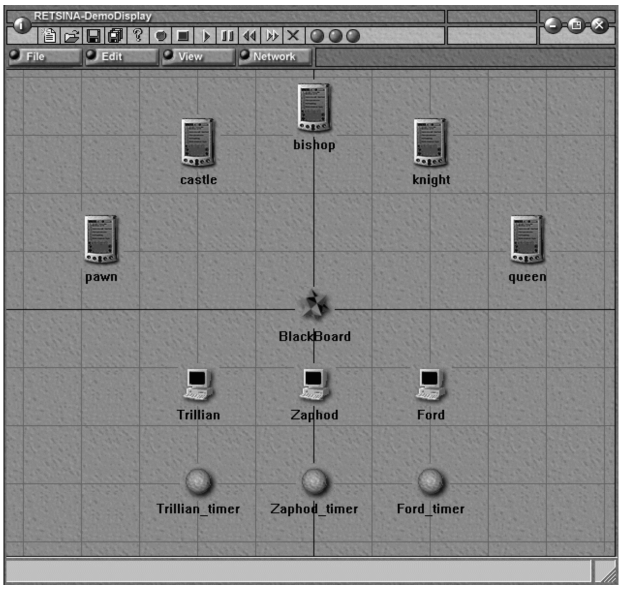

Figure 8: Agents in auction MAS.

As a first example of a deployed system using the AgentFactory toolkit, we built a protocol-agnostic auction system. All agent interactions in the system have been specified as conversational policies and agents were built in C and Java using the interaction skeletons built by conversational policy compilers.

Each of the interaction skeletons was built and tested independently of the others, which allowed for compartmentalization and eased the debugging process. Later the skeletons were integrated with the RETSINA planner and user interface components to complete the agent development process.

The MAS (see figure 8) consists of the following agents :

- *Auctioneers* conduct auctions for items in their inventory, based on English, Dutch, Spanish (fish-market) or Vickrey auction protocols. The three auctioneers in the figure are named Zaphod, Trillian and Ford
- *Timers* provide cryptographic timestamps for auctioneer agents.
- *Bidders* participate in auctions run by the auctioneer agents, and dynamically switch their protocols based on decisions made by their internal planners.
- *Blackboard* is a middle agent that provides auction listing service for auctioneer and auction news subscription service for bidders.

The protocols involved in the system are as follows:

- *Auctions:* English, Dutch, Spanish and Vickrey
- *Cryptographic Timestamping*
- *Event Supplier and Event Listener* protocols.

Once the conversational policies have been developed and verified, several agent developers cooperated to write agents that executed these policies. As a result, the entire system was developed in under 2 weeks and addition of a new auction protocol can be done in the matter of hours. Time spent debugging agent interaction (which is normally the most difficult process in MAS development) accounted for less then 10% of total debugging time.

8. CONCLUSIONS

In this paper, we have presented a scheme through which agents can infer commitments merely through observable communications, thus obviating the need for an explicit commitment infrastructure or high-level semantics. We have also presented a computationally feasible algorithm for detecting such commitments using the finite-state automaton representation of communication protocols, previously thought to be too impoverished to convey such information.

We have utilized the protocol-level semantics in the design of AgentFactory toolkit, which has been used in development of several multi-agent systems and was proven to be an efficient way to develop such systems.

9. REFERENCES

[1] L. Bölöni and D. C.Marinescu. A multi-plane state machine agent model. Technical Report CSD-TR 99-027, Purdue University, September 1999.

[2] L. Bölöni, K. Hoffmann, D. Mlynekand, and D. C.Marinescu. An object-oriented framework for building collaborative network agents. *Intelligent Systems and Interfaces*, 1999.

[3] M. Colombetti. A commitment-based approach to agent speech acts and conversations. In *Proceedings of Workshop on Conversational Policies at Autonomous Agents 2000*, Aug 2000.

[4] T. F. Consortium. Fipa 2000 specifications. www.fipa.org, 2000.

[5] T. Finin, Y. Labrou, and J. Mayfield. Kqml as an agent communication language. In J. Bradshaw, editor, *Software Agents.* MIT Press, 1997.

[6] N. A. Lynch. *Distributed Algorithms*, chapter pages 200-231. Morgan Kaufman, 1996.

[7] M. Paolucci, D. Kalp, A. Pannu, O. Shehory, and K. Sycara. A planning component for RETSINA agents. In *Proceedings of Workshop on Agent Theories And Lanugages at ICMAS99.* 1999.

[8] J. Pitt and A. Mamdani. A protocol-based semantics for agent communication language. In *Proceedings of International Joint Conference on Artificial Intelligence*, 1999.

[9] T. Sandholm, S. Sikka, and S. Norden. Algorithms for optimizing leveled commitment contracts. In *International Joint Conference on Artificial Intelligence*, pages 535–540, 1999.

[10] M. P. Singh. Toward interaction-oriented programming. In *International Conference on Multiagent Systems (ICMAS) Workshop on Norms, Obligations, and Conventions*, December 1996.

[11] M. P. Singh. Developing formal specifications to coordinate heterogeneous autonomous agents. In *IJCAI98*, pages 261–268, 1998.

[12] I. Smith, P. Cohen, J. Bradshaw, M. Greaves, and H.Holmback. Designing conversation policies using joint intention theory. In *Proceedings of International Joint Conference on Artificial Intelligence*, 1998.

[13] T. Winograd and F. Flores. *Understanding computers and cognition: A new foundation for design.* Ablex, Norwood, NJ, 1986.