Designing and Executing Protocols Using The Event Calculus

Pınar Yolum and Munindar P. Singh
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7535, USA
{pyolum, mpsingh}@eos.ncsu.edu

1. INTRODUCTION

Multiagent protocols regulate the interactions between agents. In addition to ensuring meaningful conversations, protocols should also respect agents' autonomy and enable them to interact flexibly to exploit opportunities and to handle exceptions. Our approach to specifying protocols is based on capturing the intrinsic meaning of actions and explicitly representing them as part of the protocol. We model these intrinsic meanings through *social commitments*. Conceptually, social commitments capture the obligations from one party to another. Operations to create and manipulate commitments, combined with the reasoning rules, enable agents to reason about their interactions. By representing the intrinsic meaning of the actions, we develop protocols that permit the agents to reason about their and others' behavior during the execution of the protocol, and to modify their actions as best suits them.

2. EVENT CALCULUS

The event calculus (EC), introduced by Kowalski and Sergot [1], is a formalism to reason about events. It is based on many-sorted first order predicate calculus. *Events* in EC initiate and terminate *fluents*, which are properties that are allowed to have different values at different *time points*. Their value is manipulated by the occurrence of events. A fluent starts to hold after an event that can initiate it occurs. Similarly, it ceases to hold when an event that can terminate it occurs. The event calculus used in this paper is a subset of Shanahan's version [2].

In the following, a refers to an event, f refers to a fluent, and t refers to a time point. The variables that are not explicitly quantified are assumed to be universally quantified. $\leftarrow$ denotes implication and $\wedge$ denotes conjunction. The time points are ordered by the $<$ relation, which is defined to be transitive and asymmetric. *Initiates(*a, f, t*)* means that fluent f holds after event a at time t. *Terminates(*a, f, t*)* means that fluent f does not hold after event a at time t. *Happens(*a, t*)* means that event a takes place instantaneously at time t. *HoldsAt(*f, t*)* means that the fluent f holds at time t.

AGENTS'01, May 28-June 1, 2001, Montréal, Quebec, Canada.

3. COMMITMENTS IN EC

Social commitments are commitments made by one agent to another agent to carry out a certain course of action. Commitments result from communicative actions. That is, agents create commitments and manipulate them through the protocol they follow. We represent commitments as properties in the event calculus, and develop a scheme where we model the creation and manipulation of commitments as a result of performing actions. Further, by allowing preconditions to be associated with the initiation and termination of properties, different commitments can be associated with communicative acts to model the communications among agents more concretely. Formally, a social commitment C*(x, y, G, p)* relates a debtor *x*, a creditor *y*, and a condition *p*, in the scope of a context group *G* [3]. When a social commitment of this form is created, *x* becomes responsible to *y* for satisfying *p*. The context group *G* is the organization within which the commitment exists. The condition *p* may involve relevant predicates and commitments, allowing the commitments to be nested or conditional. A commitment is *base-level* if *p* does not refer to other commitments; it is a *metacommitment* if *p* refers to a base-level commitment. Next, we formalize the operations that can be performed to create and manipulate commitments. In the following discussion, x, y, z denote agents, c, c' denote commitments, and e, f denote events.

1. *Create(e, x, c)* establishes the commitment c. The *create* operation can only be performed by the debtor of the commitment, x. Initiation of the event e by x creates the commitment c.

   Create($e, x, \mathsf{C}(x, y, G, p)$*)*:
   $\{$*Happens(*e, t*)* $\wedge$ *Initiates(*$e, \mathsf{C}(x, y, G, p), t$*)*$\}$

2. *Discharge(e, x, c)* resolves the commitment c. Again, the *discharge* operation can only be performed by the debtor of the commitment to mean that the commitment has successfully been carried out. When x performs the event e, the commitment c is terminated.

   Discharge($e, x, \mathsf{C}(x, y, G, p)$*)*:
   $\{$*Happens(*e, t*)* $\wedge$ *Initiates(*e, p, t*)*$\}$

3. *Cancel(e, x, c)* cancels the commitment c. Usually, the cancellation of a commitment is followed by the creation of another commitment to compensate for the former one. When x performs the event e, the commitment c is terminated.

   Cancel($e, x, \mathsf{C}(x, y, G, p)$*)*:
   $\{$*Happens(*e, t*)* $\wedge$ *Terminates(*$e, \mathsf{C}(x, y, G, p), t$*)*$\}$

4. *Release(G, c)* or *Release(e, y, c)* releases the debtor from the commitment c. It can be performed either by the creditor

or the context group, to mean that the debtor is no longer obliged to carry out his commitment.

Release$(e, y, \mathsf{C}(x, y, G, p))$: {*Happens*$(e, t) \wedge$ *Terminates*$(e, \mathsf{C}(x, y, G, p), t)$}

5. *Assign(e, y, z, c)* eliminates the commitment c, and creates a new commitment c' for which z is appointed as the new creditor.

   Assign$(y, z, \mathsf{C}(x, y, G, p))$: {*Happens*$(e, t) \wedge$ *Terminates*$(e, \mathsf{C}(x, y, G, p), t) \wedge$ *Initiates*$(e, \mathsf{C}(x, z, G, p), t)$}

6. *Delegate(e, x, z, c)* eliminates the commitment c, and creates a new commitment c' in which the role of the debtor is transferred to z.

   Delegate$(e, x, z, \mathsf{C}(x, y, G, p))$: {*Happens*$(e, t) \wedge$ *Terminates*$(e, \mathsf{C}(x, y, G, p), t) \wedge$ *Initiates*$(e, \mathsf{C}(z, y, G, p), t)$}

We formalize reasoning rules about commitments that operationalize the impact of a commitment on the flow of the protocol. Postulate 1 states that a commitment is no longer in force if the condition committed to comes to hold. For the condition p to hold, an event must occur to initiate it. In Postulate 1, when the event e occurs at time t, it initiates the property p, and therefore the commitment $\mathsf{C}(x, y, G, p)$ can be terminated.

POSTULATE 1. *Terminates*$(e, \mathsf{C}(x, y, G, p), t) \leftarrow$ *HoldsAt*$(\mathsf{C}(x, y, G, p), t) \wedge$ *Happens*$(e, t) \wedge$ *Initiates*(e, p, t) ∎

In order to represent metacommitments, we introduce a fluent *meta(p, q)* that captures the conditional commitments. The *meta(p, q)* requires q to hold when p holds and the fluent itself to be false if p is false. Using this fluent, we represent a metacommitment by C*(x, y, G, meta(p, q))* meaning that the debtor x becomes committed to bringing about q when p is true.

The following two postulates capture how a metacommitment evolves based on the temporal ordering of the commitments it refers to. When the metacommitment $\mathsf{C}(x, y, G, meta(p, q)), t)$ holds, if p becomes true, then the original metacommitment ceases to exist but a new base-level commitment is created, since the debtor x is now committed to bring about q. In Postulate 2, when the event e occurs, it initiates p, which results in the termination of the original metacommitment, and the initiation of $\mathsf{C}(x, y, G, q)$.

POSTULATE 2. *Initiates*$(e, \mathsf{C}(x, y, G, q), t) \wedge$ *Terminates*$(e, \mathsf{C}(x, y, G, meta(p, q)), t) \leftarrow$ *HoldsAt*$(\mathsf{C}(x, y, G, meta(p, q)), t) \wedge$ *Happens*$(e, t) \wedge$ *Initiates*(e, p, t) ∎

Again, when the metacommitment $\mathsf{C}(x, y, G, meta(p, q))$ holds, if an event e that can initiate q occurs, q starts to hold and the original metacommitment is satisfied and terminated. No additional commitments are created.

POSTULATE 3. *Terminates*$(e, \mathsf{C}(x, y, G, meta(p, q)), t) \leftarrow$ *HoldsAt*$(\mathsf{C}(x, y, G, meta(p, q)), t) \wedge$ *Happens*$(e, t) \wedge$ *Initiates*(e, q, t) ∎

4. PROTOCOLS IN EC

We represent the flow of execution within the protocol through the *Initiates* and *Terminates* clauses. In addition to defining which fluents they initiate or terminate, the required preconditions for activating these predicates can be specified. Thus, we define a *protocol specification* as a set of *Initiates* and *Terminates* clauses that define which properties pertaining to the protocol are initiated and terminated by each action.

Next we define how a protocol run is structured, which we represent by a set of actions that take place at specific timepoints. That is, a *protocol run* is a set of *Happens* clauses along with an ordering of the timepoints referred to in the predicates.

Notice that our protocol definition does not indicate any starting states, final states or transitions among executions states. An agent can start a protocol by performing any of the actions whose preconditions match the current state of the execution. By appropriately increasing or decreasing the preconditions of the actions, a protocol can be abbreviated or enhanced to allow a broader range of interactions. Although we do not represent the final states of a protocol explicitly, we can examine a protocol run to determine if any agent has backed out of its commitment. A protocol run is *complete* if all the base-level commitments that have been created are resolved. Formally, $\exists e$*Happens*$(e, t) \wedge$ *Terminates*$(e, \mathsf{C}(x, y, G, p), t)$ $\leftarrow$ *HoldsAt*$(\mathsf{C}(x, y, G, p), t)$.

If a protocol run is not complete, that is, if there is an open base-level commitment after the execution of the protocol, we know that a participant has not fulfilled its commitment. This signals a violation of the protocol. Although we don't investigate the compliance of agents in this work, incomplete protocol runs provide evidence to identify non-compliant agents.

Agents can generate protocol runs at run time to decide if an action is appropriate at a particular state of the execution. We have used Shanahan's abductive event calculus planner [2] to demonstrate how a protocol can be defined based on the preconditions and the effects of actions, and how possible paths can be generated between an initial state and a goal state [4]. Generating protocol runs at run time enables the agents to cope with exceptions by reconstructing plans as necessary. Thus, agents can execute protocols flexibly by taking advantage of opportunities, and handling exceptions.

5. CONCLUSION

Commitments have been studied before but have not been used for protocol specification as we have done here. The specification of protocols in terms of commitments of the agents to one another allows agents to reason about their actions, thus enabling them to take care of the unexpected situations that may arise at run time. Event calculus enables us to represent commitments, operations on them, and reasoning rules about them uniformly. Based on this formal grounding, multiagent protocols can be specified rigorously yet flexibly.

6. ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under grant IIS-9624425 (Career Award).

7. REFERENCES

[1] R. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[2] M. Shanahan. An abductive event calculus planner. *Journal of Logic Programming*, 44:207–239, 2000.

[3] M. P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.

[4] P. Yolum. Flexible execution of e-commerce protocols: A commitment-based approach. Master's thesis, Department of Computer Science, North Carolina State University, Raleigh, June 2000.