Formalization of Commitment-Based Agent Interaction

Jie Xing
Operations Research
North Carolina State University
Raleigh, NC 27695-7913, USA
jxing@eos.ncsu.edu

Munindar P. Singh
Computer Science Department
North Carolina State University
Raleigh, NC 27695-7534, USA
singh@ncsu.edu

ABSTRACT

We develop a generic agent interaction model that supports agent coordination. We propose commitment patterns, which accommodate revisions and exceptions, to model agent interaction. We formalize commitment patterns declaratively in temporal logic. We apply statecharts to specify behavior models of agents who follow our commitment patterns. The statecharts provide an operational semantics, which can be used as a rigorous basis for agent coordination. We propose a generic agent behavior model and prove that it operationally supports our temporal logic semantics. In this manner, we provide a basis for formally designing coordinated multiagent systems.

Keywords

Multi agent; commitments; operational semantics; Statecharts; temporal logic

1. INTRODUCTION

Current applications such as electronic commerce arise in environments that are open, heterogeneous, distributed, dynamic, and with autonomous components. Such components are naturally modeled as agents and solutions are naturally built as multiagent systems. Agents are persistent computations that can perceive, reason, act and communicate. Agents can be autonomous and heterogeneous, and represent different components and organizations. Conventional software engineering lacks the abstractions necessary to model multiagent systems. We consider the problem of creating specifications for agent behavior and interaction to achieve the necessary coordination to support various kinds of communicative or "conversational" interactions. Following [2][9], we propose to model the interactions among agents in terms of agents' commitments to one another. We show how temporal logic [6] and statecharts [7] can be applied in modeling and operationalizing commitments.

EXAMPLE 1. *The Contract Net Protocol (CNP) is among the most well known protocols in DAI [4]. CNP involves one manager agent and several contractor agents. The manager calls for bids for a number of widgets with a defined price and deadline. Some of the contractors propose bids. The manager chooses a bid and assigns a task to the selected contractor. When the contractor finishes its assigned task, the manager evaluates the results. Traditional CNP specifications cannot handle the following revisions and exceptions.*

- *The manager may change its task assignment by requesting additional widgets with the same price and deadline after it has awarded the task to the contractor. The contractor has to perform the revised task again and send the revised results to the manager.*
- *The contractor may send requests to increase the price if the manager changes the task assignment.*
- *The contractor may be unable to finish its task by the specified deadline. The manager may either cancel the task assignment or grant an extension.*

This paper studies how to engineer flexible interactions among agents. We rely on two crucial properties of agents. First, the agents must interact at a high level by forming and managing the commitments to one another. These commitments are about the information that the agents exchange, about the changes to that information, and about each other's needs. Second, the agents must be persistent. This is essential so that agents may form, manage, and act according to their commitments. We model interactions as a set of composable *metacommitment patterns*. We formalize the patterns in temporal logic. Each pattern invokes commitment operations. These patterns cover different requirements for interactions.

Current approaches include only simple request-response interactions, whereas our approach incorporates more flexible interactions that are needed to accommodate exceptions and revisions. We develop a generic agent behavior model in statecharts. Statecharts provide an operational semantics. Importantly, we show how the operational semantics relates to the declarative semantics for agent coordination.

This paper is organized as follows. Section 2 introduces interaction patterns and our specification language for agent interactions. Section 3 formalizes metacommitment patterns. Section 4 introduces the operational semantics. Sec-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC 2001 Las Vegas, NV USA
© 2001 ACM 1-58113-324-3/01/02 ..$5.00

tion 5 proves the soundness of operations semantics. Section 6 discusses the relevant literature.

2. SPECIFYING INTERACTION

We now discuss the main concepts of agent interactions. A *role* is an abstract entity, which captures the underlying organizational structure. A role can reason based on its local knowledge and send results to other roles. Importantly, roles are used to specify the applicable commitments. During execution, an agent with matching capabilities is bound to each role. Agents can volunteer to assume certain roles that would require them to perform certain reasoning. Thus the agent playing a role must implement all the capabilities that the role provides and must honor the metacommitments of that role.

A social commitment $\mathsf{C}(x, y, G, p)$ relates a debtor role x, a creditor role y, and a condition p, in the scope of a context group G. The condition p may involve relevant predicates and commitments, allowing the commitments to be nested. The context group G is the organization within which the commitment exists. Making the context explicitly enables us to specify which roles exist in it and what kinds of commitments exist among them.

For example, the contractor *ctr* promises a bid to the manager *mgr*. CNP is the context of the contract net protocol and *bidp* is combination of predicates about the number, price and date of the widgets. The commitment is C(*ctr*, *mgr*, CNP, *bidp*).

DEFINITION 1. *A commitment can be viewed as an abstract data type. The following operations are relevant in the paper. Here, x and y denote roles, and c denotes a commitment of the form* $\mathsf{C}(x, y, G, p)$.

- *Create(x, c) establishes the commitment c. The create operation can only be performed by the debtor of the commitment.*
- *Discharge(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. Thus, after the discharge operation the condition p starts to hold.*
- *Cancel(x, c) cancels the commitment c. Generally, cancellation of a commitment is followed by the creation of another commitment to compensate for the former one.*
- *Release(y, c) or Release(G, 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.* ∎

A *message* carries information exchanged between roles. A message is identified by its sender, receiver, content, and type. We use the common agent communication language primitives of *inform* (statements of facts), *request* (commands, requests or advice), *reject*, and *withdraw*. We introduce a primitive called *correct*, which can be used for requesting updated results.

2.1 Interaction Patterns

Design patterns are valuable to object-oriented software design. Following similar situations, we define patterns of interaction, each of which captures an important scenario. These patterns specify in term of the participants' commitments. Different combinations of these patterns can yield different kinds of agent interactions.

The patterns relate to conversation-intensive scenarios. An agent can reason about its actions based on its local knowledge. Each pattern involves two roles, a consumer and a provider. An agent can plays different roles in different patterns. Predicates with some arguments (i.e., data values) are used to represent the information exchanged by the agents.

1. *Notify the consumer.* This comes into effect when a role has just completed its reasoning for the first time. It informs another role of the computed data values, and becomes committed to the specified predicates. For example, a contractor who proposes a bid will notify the manager and commit to the price.
2. *Entertain request.* This means that a role will accept requests from another role and reason about the request. For example, a contractor will entertain a call for bids and start reasoning about the call.
3. *Renotify the consumer.* This comes into effect when a role has just completed its reasoning for the second or a later iteration, and some of its existing commitments are violated by the recently completed reasoning. The violation would typically occur because the predicates to which the agent had committed have been falsified by the results just obtained. For example, the contractor requests to correct a task assignment at a higher price.
4. *Entertain update.* This comes into effect when a role accepts requests to correct some data values that another role may supply. For example, a manager may request additional widgets at the same price and deadline. The contractor would reason again with the updated data values.
5. *Dissatisfaction.* This comes into effect when a consumer role is dissatisfied with the results. It sends a reject to the provider. For example, the manager sends a reject to the contractor to cancel the task assignment and release the associated commitment.
6. *Abort.* A provider role may cancel its commitment to a consumer role by sending a withdraw message to the consumer role. For example, the contractor may cancel its task assignment if it is unable to finish the task by the deadline.

2.2 Specification Language

We define the syntax of the specification language through the following grammar whose starting symbol is *Specification*. The braces { and } indicate that the enclosed items are repeated 0 or more times.

SYN 1. *Specification* $\longrightarrow$ {*Role*}, {*PatternSpec*} $\ll$ *Sets of roles and patterns* $\gg$

SYN 2. *PatternSpec* $\longrightarrow$ *PatternFormula(Role, Role, Group, Predicate)* | *PatternFormula (Role, Role, Group, Predicate, Predicate)* $\ll$ *Formula of commitment patterns* $\gg$

A specification consists of sets of roles and pattern specifications. Each pattern specification is denoted by a formula name along with roles involved in it, the context group these roles are in, and the predicate. A pattern formula can be expressed as a CTL [6] formula, which is introduced in Section 3.

3. FORMALIZATION

We now define the domain-independent propositions that correspond to actions for communications of a generic agent. We formalize our communication primitives as propositions. $Pred$, $Pred_1$, and $Pred_2$ are predicates with domain arguments. As before, x and y denote two roles.

- *inform*$(x, y, Pred)$: x informs y about $Pred$.
- *request*$(x, y, Pred)$: x requests y for information about $Pred$.
- *correct*$(x, y, Pred_2, Pred_1)$: x sends a correction to y, replacing $Pred_1$ by $Pred_2$.
- *reject*$(x, y, Pred)$: x rejects y's information about $Pred$.
- *withdraw*$(x, y, Pred)$: x retracts information about $Pred$ from y.

We define the following domain event propositions about status changes of agent computation.

- *computed_done* : a role finishes its computation for the first time.
- *recomputed_done* : a role finishes its computation for the second or the later time.
- *abort* : a role aborts its computation.
- *valid_commitment* : a role's commitments with another role are still valid after it just finishes its computation for the second or the later time.

The formal declarative semantics for the metacommitment patterns is given in Computation Tree Logic (CTL) [6], a branching time logic. The following development suffices for our purposes. We give a semantics to CTL in terms of a CTL structure, roughly a finite rooted directed graph whose paths correspond to different computations. One can imagine the graph being unraveled into an infinite tree. In the following, AP is a set of atomic propositions.

DEFINITION 2. *A CTL structure* τ *is a four-tuple* (Q, R, P, s_0)*, where*

- Q *is a finite set of states.*
- R *is a binary relation on* Q*, which gives the possible transitions between states and must be total; that is,* $\forall\ x \in Q\ \exists\ y \in Q.\ (x, y) \in R$.
- P*:* $Q \rightarrow 2^{AP}$ *assigns to each state the set of atomic propositions that are* true *at that state.*
- s_0 *is the root state of* Q*.* $P(s_0) = \emptyset$.

Our formal language involves the usual boolean operations plus some temporal operations. Roughly $\mathsf{AG}p$ holds in s iff p holds at all future states on all paths through s. $\mathsf{AF}p$ holds at s iff p holds eventually on each path through s. To check if a CTL structure satisfies a formula f, we can apply the model checker algorithm [3].

The communication propositions *inform*, *reject*, and *withdraw* imply some commitment operations.

- $\forall\ x, y, Pred, \vec{v}$: $\mathsf{AG}[inform(x, y, Pred(\vec{v})) \rightarrow \mathsf{AF}\ [create\ (x,\ \mathsf{C}(x,\ y,\ G,\ Pred(\vec{v})))]]$
- $\forall\ x, y, Pred, \vec{v}$: $\mathsf{AG}[reject(x, y,\ Pred(\vec{v})) \rightarrow \mathsf{AF}\ [release\ (x,\ \mathsf{C}(y,\ x,\ G,\ Pred(\vec{v})))]]$
- $\forall\ x,\ y,\ Pred,\ \vec{v}$: $\mathsf{AG}[withdraw(x,\ y,\ Pred(\vec{v})) \rightarrow \mathsf{AF}\ [cancel\ (x,\ \mathsf{C}(x,\ y,\ G,\ Pred(\vec{v})))]]$

Based on the commitment operations and communication primitives, we now give our specification language for commitment patterns.

SYN 3. *PatternFormula* $\longrightarrow$ AG*[statusProp* $\rightarrow$ AF *commProp]* | AG *[commProp* $\rightarrow$ AF*PatternFormula]*

SYN 4. *commProp* $\longrightarrow$ *communicationOP(Role, Role, Pred)* | *communicationOP(Role, Role, Pred, Pred)*

SYN 5. *communicationOP* $\longrightarrow$ *inform* | *request* | *correct* | *reject* | *withdraw* $\ll$*Domain Independent Propositions*$\gg$

SYN 6. *statusProp* $\longrightarrow$ *computed_done* | *recomputed_done* | *abort* | *valid_commitment* $\ll$ *status propositions* $\gg$

Pred is defined as the predicate with a vector of domain arguments $\vec{v}$. We now apply the specifications to formalize our commitment patterns.

- *notify* $(x, y, G, Pred) = \forall\ \vec{v}.\ \ \mathsf{AG}[computed\_done \rightarrow \mathsf{AF}inform(x, y, Pred(\vec{v}))]$
- *entertain-request* $(x, y, G, Pred) = \forall\ \vec{v}.\ \mathsf{AG}[request(y,\ x,\ Pred(v)) \rightarrow \mathsf{AF}notify(x,\ y,\ G,\ Pred(\vec{v}))]$
- *renotify* $(x, y, G, Pred, Pred) = \forall\ \vec{v_1},\ \vec{v_2}.\ \mathsf{AG}\ [recomputed\_done \wedge \neg valid\_commitment \rightarrow \mathsf{AF}[inform\ (x,\ y,\ Pred(\vec{v_2})) \wedge withdraw(x,\ y,\ Pred(\vec{v_1}))]]$

- *entertain-update* $(x, y, G, Pred, Pred) = \forall\ \vec{v_1},\ \vec{v_2}.\ \mathsf{AG}$ [*correct* $(y,\ x,\ Pred(\vec{v_2}),\ Pred(\vec{v_1})) \rightarrow \mathsf{AF}[renotify(x,\ y,\ Pred(\vec{v_2}),\ Pred(\vec{v_1})) \vee abort\ (x,\ y,\ G,\ Pred(\vec{v_1}))]]$

- *abort* $(x, y, G, Pred) = \forall\ \vec{v}.\ \mathsf{AG}[abort \rightarrow \mathsf{AF}withdraw\ (x,\ y,\ Pred(\vec{v}))]$

- *dissatisfy* $(x, y, G, Pred, Pred) = \forall\ \vec{v_1},\ \vec{v_2}.\ \mathsf{AG}[reject(x,\ y,\ Pred(v_1)) \rightarrow \mathsf{AF}[abort\ (x,\ y,\ G,\ Pred(\vec{v_1})) \vee renotify\ (x,\ y,\ Pred(\vec{v_2}),\ Pred(\vec{v_1}))]]$

4. OPERATIONAL SEMANTICS

To support our operational semantics for agent interactions, we require that the agents follow a generic behavioral model, which is expressed as a statechart. Statecharts are well established in software engineering as means to specify concurrent computations [7]. A statechart is composed of *states* (OR-states, AND-states, and basic states) and *transitions*.

DEFINITION 3. *A state is the main element in a statechart. A state may be idle, perform actions, or invoke activities. Each state s has a type. If type(s) is basic, s has no child state. If type(s) is AND, s comprises a number of child states; being in an AND state implies being in all its child states simultaneously. If type(s) is OR, s consists of a number of child states; being in an OR state means being in exactly one of its states. A state is defined as a root state if it isn't a child state of any state. Activities are operations performed in a state and are not represented graphically.*

DEFINITION 4. *A basic configuration of a statechart is a maximal set of basic states that the system can be in simultaneously. S_{bc} is the set of all legal basic configurations. An initial configuration of a statechart is a basic configuration with the root state.*

DEFINITION 5. *Triggers are the dynamic elements of a statechart. Triggers cause state transitions. Events, conditions, or a combination can be triggers. The events are classified into two types: external and internal events. An external event occurs randomly outside a statechart while an internal event is generated deterministically by the statechart transition.*

DEFINITION 6. *A transition label lbl is composed of an event, E, a condition, C, and an action, A, and is written as E[C]/A. E and A may be empty and C may be* true *(trivial components may be omitted).*

DEFINITION 7. *A transition $t = (S_1, S_2, lbl) \in (2^S \times 2^S \times L)$ is composed of a source state set S_1, a target state set S_2, and a transition label lbl. S_1 and S_2 are basic configurations. L is a label set.*

DEFINITION 8. *A statechart S_c is a four-tuple (S, S_{bc}, T, c_0). S is a set of states. $S_{bc} \subseteq 2^S$ is a set of legal basic configurations. T is the set of transitions. c_0 is the initial basic configuration.*

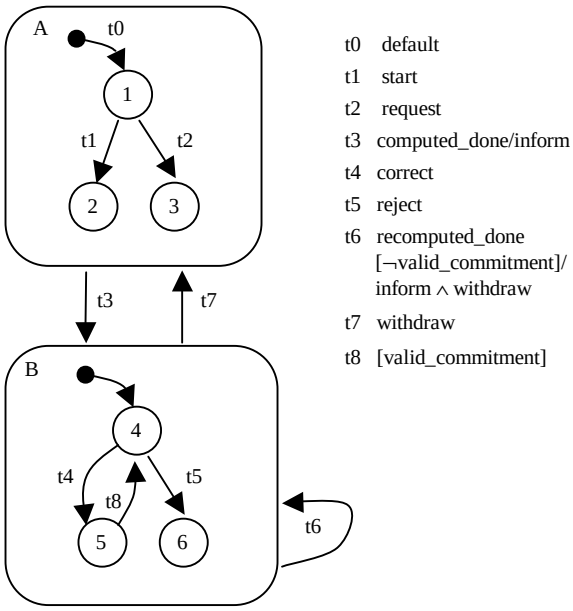

Figure 1: Generic agent behavior model

Figure 1 represents our agent behavior models denoted by S_{ca} in a statechart. An agent can instantiate several threads for different interaction scenarios. Each thread follows our agent behavior model. For example, the contractor agent has a thread for proposing bids and another thread for performing tasks. Both threads use the same behavior model S_{ca}.

For the model S_{ca}, on receiving an initial trigger signal, an agent is in state 1. After the agent receives a start event, or a request event, it begins reasoning in state 2 or the state 3. The start event means that the agent can start its reasoning by itself. The request event means that the agent can start its reasoning by requests from other agents. Upon completion of the reasoning, the agent sends the reasoning results to some selected agents and commits to the results sent. The agent is in state 4.

A correct event causes the agent to reexecute its reasoning in state 5. After finishing its reasoning, the agent obtains new results. If the results change substantially to invalidate the agent's commitments, the agent sends the updated results again (canceling the old commitments and creating new ones). If the agent receives a reject event in state 4, the agent transits to state 6 and reasons in state 6. If an abort event occurs, it withdraws the commitments with its consumer agent. If a recomputed_done event occurs and current commitments are invalid, the agent sends a new results to satisfy its consumer agent.

5. SOUNDNESS

To relate our operational semantics with the temporal logic specifications, we must produce a CTL structure $\tau = (Q, R, P, s_0)$ from our statechart. It is difficult to prove eventuality with statecharts. An external event may never occur during the execution of a statechart. Thus eventuality of a condition is always falsified if external events are required

to reach the condition. To address the problem and capture possible executions of a statechart, we separate states in our CTL structure corresponding to the same basic configuration. Further we introduce a state for each transition. We add additional members of the relationships into R. We provide the following rules for deriving CTL structure from the statechart.

For each transition $t = (S_1, S_2, lbl)$ and $lbl = E[C]/A$, we create a state S' which is between S_1 and S_2. S' has atomic propositions (denoted by {E,C}) which can be derived from E and C - for reason of space, we don't elaborate this here. Similarly action A generates atomic propositions (denoted by $\{A\}$) in S_2. We place the pair (S_1, S'), (S', S_2) into R. If S_2 is reached from different transitions and has different propositions, we separate the basic configuration S_2 into different states in the CTL structure. If E includes an external event, we place the (S_1, S_1) into R. The propositions in S_1 don't change. This forces an execution from S_1 to S_1 if the external event doesn't occur. If there are a set of activities $\{acts\}$ in S_2, each activity generates an atomic proposition in state S_2. We denote the set of atomic propositions as $\{acts\} \subseteq P(S_2)$.

Now CTL structure $\tau = (Q, R, P, s_0)$ may be derived from a statechart $S_c = (S, S_{bc}, T, c_0)$ by the following rules.

1. We introduce a state $S' \in Q$. $P(S')$ = {E,C}, $P(S_2) = \{A\} \bigcup \{acts\}$, $(S_1, S'), (S', S_2) \in R$ iff $t = (S_1, S_2, lbl) \in T$, $S_1, S_2 \in S_{bc}$ and $\{acts\}$ are a set of activities in S_2, where $lbl = E[C]/A$

2. If E consists of an external events in the transition $t = (S_1, S_2, lbl)$, we introduce a relationship, $(S_1,S_1) \in R$

3. If the basic configuration $s_{bc} \in S_{bc}$ is reached from different transitions and has different propositions generated by the first rule, we separate the s_{bc} into different states

Figure 2 shows the CTL structure derived from the statechart of Figure 1. To derive CTL structure from the statechart S_{ca}, we have to know the event type of each transition. *start*, *computed_done*, *recomputed_done*, and *abort* are internal events. *Request*, *correct* and *reject* are external events.

DEFINITION 9. *τ,s $\models \phi$ means that the formula ϕ holds at state s in the CTL structure τ.*

DEFINITION 10. *A statechart S_c is sound with respect to a formula ϕ iff ($\forall\tau$: (S_c generates τ) $\rightarrow \tau, s_0 \models \phi$). A statechart S_c is sound with respect to a specification $\{\phi_1, \phi_2, \ldots, \phi_n\}$ iff S_c is sound with respect to each ϕ_i.*

THEOREM 1. *S_{ca} is sound with respect to any specification consisting of the patterns {entertain-request, entertain-update, dissatisfy, notify, renotify, abort}.*

PROOF. *For agent behavior model S_{ca}, the event type of a transition is given by the above. In Figure 2, the derived CTL structure τ shows that it has 19 states and 31 binary relationships among these states. We can verify each formula of our commitment patterns by hand or by the model checker [3]. Thus we can show that the computation τ derived from statechart S_{ca} satisfies every formula ϕ of commitment patterns. Thus we prove the theorem.* ∎

EXAMPLE 2. *The manager and contractor agents in CNP can follow our agent behavior model. The manager has two threads, for a call for bids and for task assignment. The contractor has also two threads, which implement agent behavior model S_{ca}, for proposing a bid and for performing a task. For conventional CNP, we have the following:*

- *The manager initializes a thread. It sends out request for a call for bids. To satisfy the request by entertain-request, the contractor initializes a thread to reason whether to bid.*
- *If contractor decides to bid, it send a bid to the manager (notify).*
- *The manager chooses the optimal bid and sends a task assignment to the willing contractor. The willing contractor satisfies entertain-request for tasks by performing task assignment.*
- *When the task results are available, the contractor notifies the manager.*

To allow revisions in CNP, we have the following:

- *If the manager changes its the number of widgets, it can send the corrected information to the contractor (renotify).*
- *The contractor reasons again (entertain-update). The contractor may request to increase the price. If so, it renotifies the manager.*

To handle exceptions in CNP, we have the following:

- *If the contractor does not finish its task by the deadline, it can withdraw the task assignment and cancel its commitment by using abort.*

6. DISCUSSION

Because of the autonomy and decentralization of the participants represented by agents in dynamic environments, specifying and managing agent interactions can be challenging. Conventional techniques fall into one of two extremes, being either too rigid (over-restricting the designer) or too unstructured (not helping the designer). Our commitment-based approach takes the middle path, emphasizing the coherence desired from the activities of autonomous decentralized entities, but allowing the entities to change their mind in a controlled manner, which enables them to achieve progress in a dynamic, unpredictable world. We formalize

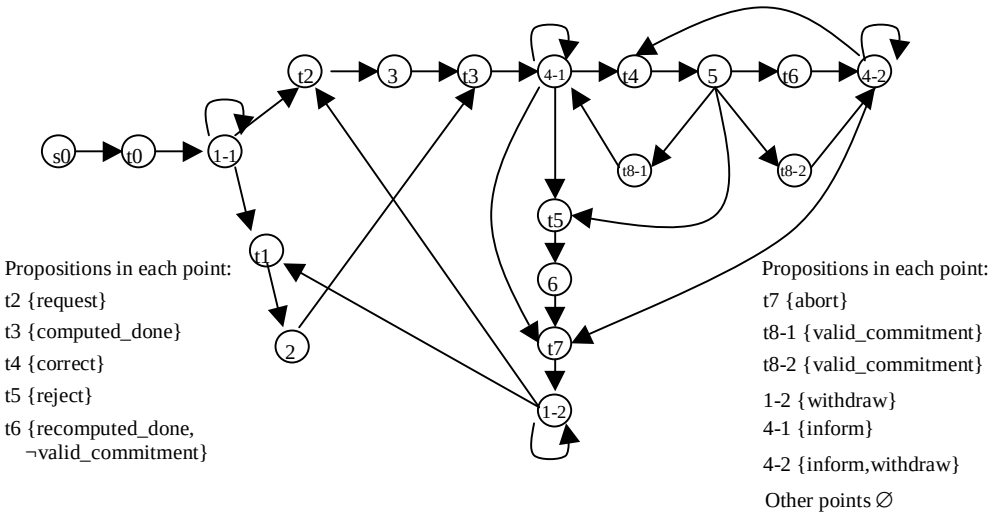

Figure 2: CTL structure of the statechart S_{ca}

agent interactions represented by commitment patterns. We provide a behavioral model to execute autonomous, heterogeneous agents.

Dignum and van Linder [5] propose a framework for social agents based on dynamic logic. Agents interact each other based on deontic relationship among agents. They deal with the motivational attitudes of agents and define many concepts such as wishes, goals, intentions, and commitments or obligations. This complements our approach. We focus more on operational semantics. The reasoning results represented by predicates with data values as the arguments trigger the interaction.

Barbuceanu and Fox [1] describe a language for specifying coordination among agents. Their approach involves finite state representations of an entire conversation. While their approach is quite effective in coordinating agents, it leaves open the question about how the given conversation is acquired. For specifying agent behavior, we use a statechart to represent agent behavior, not using flat finite state machine (FSM), which may cause state exposition problem when there are a large number of states in FSM. For verification purposes, we use temporal logic to verify our behavior model.

Klein and Dellarocas exploit a knowledge base of generic exception detection, diagnosis, and resolution expertise [8]. Specialized agents are dedicated to exception handling. This approach is complementary to ours; special exception handling roles could be included in our approach with commitments by other roles.

7. REFERENCES

[1] M. Barbuceanu and M. S. Fox. COOL: A language for describing coordination in multi agent systems. In *Proceedings of the International Conference on Multiagent Systems*, pages 17–24, 1995.

[2] C. Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proceedings of the International Conference on Multiagent Systems*, pages 41–48, 1995.

[3] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transations on Programming Languages and Systems*, 8(2):244–263, 1986.

[4] R. Davis and R. G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.

[5] F. Dignum and B. van Linder. Modelling social agents: Communication as action. In *Intelligent Agents III: Agent Theories, Architectures, and Languages*, pages 205–218. Springer-Verlag, 1997.

[6] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. North-Holland, Amsterdam, 1990.

[7] D. Harel and A. Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[8] M. Klein and C. Dellarocas. Exception handling in agent systems. In *Proceedings of the 3rd International Conference on Autonomous Agents*, pages 62–68, Seattle, Washington, 1999.

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