Autonomous Agents and Multi-Agent Systems, 00, 1–56, 2005

A Real-Time Negotiation Model and A Multi-Agent Sensor Network Implementation

LEEN-KIAT SOH lksoh@cse.unl.edu
Department of Computer Science and Engineering, University of Nebraska, 256 Avery Hall, Lincoln, NE 68588-0115,USA

COSTAS TSATSOULIS
Information and Telecommunication Technology Center (ITTC), Department of Electrical Engineering and Computer Science, University of Kansas, Nichols Hall, 2335 Irving Hill Road, Lawrence, KS 66045-7612,USA

Abstract. This paper describes a negotiation model that incorporates real-time issues for autonomous agents. This model consists of two important ideas: a real-time logical negotiation protocol and a case-based negotiation model. The protocol integrates a real-time Belief-Desire-Intention (BDI) model, a temporal logic model, and communicative acts for negotiation. This protocol explicitly defines the logical and temporal relationships of different knowledge states, facilitating real-time designs such as multi-threaded processing, state profiling and updating, and a set of real-time enabling functional predicates in our implementation. To further support the protocol, we use a case-based reasoning model for negotiation strategy selection. An agent learns from its past experience by deriving a negotiation strategy from the most similar and useful case to its current situation. Guided by the strategy, the agent negotiates with its partners using an argumentation-based negotiation protocol. The model is time and situation aware such that each agent changes its negotiation behavior according to the progress and status of the ongoing negotiation and its current agent profile. We apply the negotiation model to a resource allocation problem and obtain promising results.

Keywords: real-time negotiation, temporal logic, argumentative negotiation protocol, case-based reasoning, multi-agent system.

1. Introduction

While negotiation has been used in the past in problem solving in multi-agent systems, in our work we focus on negotiations and activities that must occur in real time. The introduction of hard real-time constraints in negotiation and action execution complicates the problem greatly, and existing negotiation protocols cannot provide an adequate solution. In this paper we describe a real-time negotiation model that is used in resource allocation problems. As an example domain we use multi-sensor target tracking, where each agent controls a sensor with a limited sensing coverage area. As a target moves across space, agents have to cooperate to track it. Each agent (together with the sensor it controls) consumes resources such as time, battery power, bandwidth of the communication channel, and some percentage of the CPU where the agent resides, and each agent strives to manage and utilize

Journal : AGNT | Dispatch : 10-3-2005 | Pages : 56
PIPS No. : N000000539 | ☐ LE | ☐ TYPESET
MS Code : N000000539 | ☑ CP | ☑ DISK

its resources efficiently and effectively. This motivates the agents to share their knowledge about a problem based on their viewpoints in their effort of arriving at a solution. The problem of global resource allocation becomes a problem of locally negotiated compromises and local constraint satisfaction.

We propose a logical negotiation protocol that incorporates a real-time Belief-Desire-Intention (BDI) model [43, 44] to dictate the rules of encounter among our autonomous agents. A feature of our problem involves generating a "good-enough, soon-enough" solution[1] to resource allocation. Since time is critical – for example, to make a good triangulation for the location of a target, three different sensors have to make a measurement within 2 seconds of each other – agents use time to guide their negotiation behavior. We base our temporal model on [1] in which logical events or propositions can be ordered consistently along a timeline and durations of events or propositions holding true can be derived from their relationships with others. This temporal logic allows us to define explicitly the transition of a BDI state to another, including causality and co-existence. Equipped with the definition of time, we are able to model our negotiation activity with more accuracy, spelling out how and when a state changes and how and when it changes with other states, such as state s_1 triggers state s_2, s_1 has to occur before s_2, s_1 must hold true for some time during which s_2 must hold true as well, and so on. Therefore, states may change their truth values during a reasoning process as long as the states are needed to hold constant during that time period do – releasing other states to be updated or changed by other events or states. This is critical in our agent design as each agent is multi-threaded, meaning that several threads may attempt to access and modify the same variable (state) at the same time. To maintain data integrity, when a thread is accessing or modifying a variable, other threads will be blocked, and this is the common approach. However, software designers must find out how and for how long the threads will be blocked; and this information is very important in a real-time system like ours. With temporal logic, we know for how long threads will be blocked and how these threads will be blocked awaiting which variables (states) to become accessible. This allows us to fine-tune the system to increase the efficiency of the negotiation process. Further, by incorporating temporal logic and BDI models into our negotiation protocol, we can improve the temporal and logical structure that facilitates the completion of a negotiation. We know what states are needed (and when they are needed) for a negotiation to logically complete, and we also can model the time distribution or usage needed for each step of the negotiation to complete within certain time constraints.

Note that temporal components have been in place in the BDI model [9, 43] to determine how the three modalities are related over time. For example, taking time into consideration allows one to have persistent intentions, inevitable outcomes, and so on. In our model, we use temporal logic to control the stability of a state, which in turn facilitates our multi-threaded solution.

We further define two sets of communicative acts – one for handling incoming messages and one for handling outgoing messages. A communicative act that handles incoming messages is a function that turns an event (the arrival of a message) to a set of BDI states. The function parses the incoming message and generates states that are necessary for the agent reasoning during a negotiation process. Similarly, a

communicative act that handles outgoing messages is a function that composes a message based on the agent's current BDI states and sends it out via a communication channel. We qualify these acts with temporal logic and incorporate them into the negotiation protocol. In addition to the communicative acts, we utilize a suite of real-time enabling functional predicates to assist agents in negotiations. These predicates are events that take time to execute and they also generate or modify states.

Since each agent is autonomous and reacts to its environment, each has its own knowledge base and its own monitoring of the world events, including its sensor, its neighbors, and the targets. To increase the fault tolerance of the multi-agent system, each agent is responsible only for the resources it controls (in our example domain, its sensor and associated components), and it controls the minimal set of resources it requires to achieve its task. In our work agents have minimal knowledge and information – they know how to perform their tasks, have a local, limited view of the world provided to them by the equipment they control, and know of the existence of other similar agents, but they do not have an explicit view of the information of the other agents. There is some implicit knowledge, namely that the other agents control a set of resources, that they are willing to cooperate, that they are capable of negotiation for resource sharing, and that they are truthful. To establish a common reasoning basis during a collaborative effort, an agent is required to communicate to its potential partner why it needs to share the resources controlled by the partner. This knowledge exchange can be done via different mechanisms such as a blackboard where agents post information on a common site, or auctions where a contractor-agent oversees the message passing among contractee-agents, or through agent-based negotiations where agents exchange information directly. In our approach, we use negotiations motivated by a global goal – to track as many targets as accurately as possible – guided by a set of local optimization criteria that affect the strategies. During a negotiation agents exchange information of their individual viewpoints of the current (and relevant) world situation. In this manner, the agents are able to argue and attempt to persuade each other explicitly, resulting in efficient knowledge transfer. We thus do away with a centralized information facility that requires constant updates and polling from agents, and, instead, knowledge is exchanged when necessary resulting in less communication traffic. Knowledge inconsistencies are resolved in a task-driven manner, making knowledge management easier.

One important part of the negotiation process is the determination of the negotiation strategy based on the current task description. To do so, we use a model derived from case-based reasoning (CBR) [23] that is time-constrained and that retrieves the most similar cases, selects the best case based on utility theory, adapts the case to the current situation, and then uses the case's negotiation strategy to perform negotiations. The CBR approach limits the time needed to decide on a negotiation strategy – selection (through retrieval) and generation (through adaptation) of a situation-appropriate strategy – and enables the agent to learn autonomously and adapt itself to different scenarios in the domain. We will only briefly discuss our CBR approach in this paper.

In the following, we first describe briefly the characteristics of our agents and multi-agent system and its real-time constraints. In Section 3, we describe in detail

the logical model of our negotiation protocol. Then we describe our implementation of the negotiation protocol, including a suite of real-time enabling functional predicates, in Section 4. Next, we discuss some results of our negotiation model applied to the multi-sensor target-tracking problem. Then we study some related work in agent-based negotiations. Finally, we conclude with some suggestions for further work in the area.

2. Agent characteristics

Each agent has the following characteristics:

(1) Autonomous – Each agent runs without interaction with human users. It maintains its own knowledge base, makes its own decisions, and interacts with its sensor, neighbors and environment.

(2) Rational – Each agent is rational in that it knows what its goals are and can reason and choose from a set of options and make an advantageous decision to achieve its goal [57].

(3) Communicative – Each agent is able to communicate with others, by initiating and responding to messages, and carrying out conversations.

(4) Reflective (or Aware) – According to [8], a reflective agent reasons based on its own observations, its own information state and assumptions, its communication with another agent and another agent's reasoning, and its own control or reasoning and actions. By being reflective, each agent is time aware and situationally aware. When an agent is time aware, it observes time in its decision making and actions. Its reasoning takes time into account, and thus, the outcome of a reasoning process is partially dependent on time. When an agent is situationally aware, it observes its current situation, the situation of its neighbors, and that of the world and makes decisions based on these observations. In general, an agent that is situationally aware observes the resources that it shares with other agents, its current tasks, messages, profiles and actions of its neighbors, and the external changes in the environment. In this paper, we require a stronger level of situational awareness. An agent also observes its own resources that *sustain the being* of the agent. For a hardware agent, these resources may be the battery power, the radio-frequency (RF) links, etc. For a software agent, these resources may be CPU, RAM, disk space, communication channels, etc. Note that, for example, in [48], a *bounded rationality* model is used where each agent has to pay for the computational resources (CPU cycles) that it uses for deliberation, assuming that the resources are available. In our model, however, we require an agent to be aware of whether the resources are available before even starting a negotiation.

(5) Honest – Each agent does not knowingly lie or intentionally give false information. This characteristic is also known as veracity [16].

(6) Adaptive – Each agent is able to adapt to changes in the environment and learns to perform a task better, not only reactively but also from its past experience.

(7) Cooperative – Each agent is motivated to cooperate if possible with its neighbors to achieve global goals while satisfying local constraints.

Generally, the agents in a multi-agent system may be controlling different resources and use different reasoning and negotiation techniques. In our approach, we require (1) that all agents be capable of negotiation in which they share a common vocabulary that enables message understanding, and (2) that each agent knows what resources may be used or controlled by a non-empty subset of the other agents in the environment so that it can determine whom to negotiate with. In our particular domain of application, each agent controls the same resources, since each one controls the same type of sensor. Also, each agent uses the same negotiation methodology based on CBR, but the individual case bases differ.

Formally, our multi-agent system architecture is defined as follows. In our study, a multi-agent system consists of a finite set of autonomous agents, denoted by Ω. Suppose that we define a neighborhood of an agent α_i, Ψ_{α_i} such that, $\Psi_{\alpha_i} \subseteq \Omega$, $\Psi_{\alpha_i} \neq \varnothing$ and that the agent α_i knows about all other agents in the neighborhood.

For the given agent α_i and any agent from its neighborhood, Ψ_{α_i}, $\lambda(\alpha_i, \alpha_j)$, where $\lambda(a,b)$ means agent a knows about the existence of agent b and can communicate with agent b. A neighborhood is different from a team as defined in [55], which is task-driven and formed among a set of agents to accomplish a task. A neighborhood is a subset of agents of the multi-agent system that *could* form a team. In our particular domain of application (multi-sensor target tracking), a neighborhood consists of a set of agents that control sensors that are physically close and whose sensing beams overlap. So, in our multi-agent system Ω, there is a set of neighborhoods, and each neighborhood can form any number of teams. Neighborhoods do not necessarily have the same number of members, and neighborhoods may share members.

When a target is sensed, an agent tracks the target, refers to its neighborhood information, and dynamically forms a *negotiation coalition*, that is, a subgroup of its neighborhood agents with which it *may* negotiate to request resources to assist it in its task. For example, when an agent detects a target in its sensing area, the agent immediately obtains an estimate on the position and velocity of the target. It then projects the future positions of the target and identifies the neighbors whose sensors are able to cover the target moving in the projected path. These are agents that control resources (i.e. sensor beams) that it needs to track the target, and these are the agents that will be part of the negotiation coalition.

Figure 1 shows the agent architecture. The agent consists of several threads: communication, execution, core, and multiple negotiation threads, allowing parallel processing of multiple tasks. The core thread maintains the lifecycle of the agent, linking all managers together. The execution manager manages the execution thread, scheduling radar-related tasks in a job queue and actuating them to affect the agent's environment. The communication manager manages the communication thread, sending and receiving messages through sockets, and storing messages in a mailbox that is accessed by other threads. The negotiation manager manages the negotiation threads, spawning, activating, and updating them. The core thread consists of several managers. The profile manager maintains the status of the agent, radar, its neighbors, and its environment. The CBR manager maintains the casebase and performs CBR such as retrieval, adaptation, and learning. The Radar manager is a domain-specific module that hosts a geometric model of the tracked target within the radar

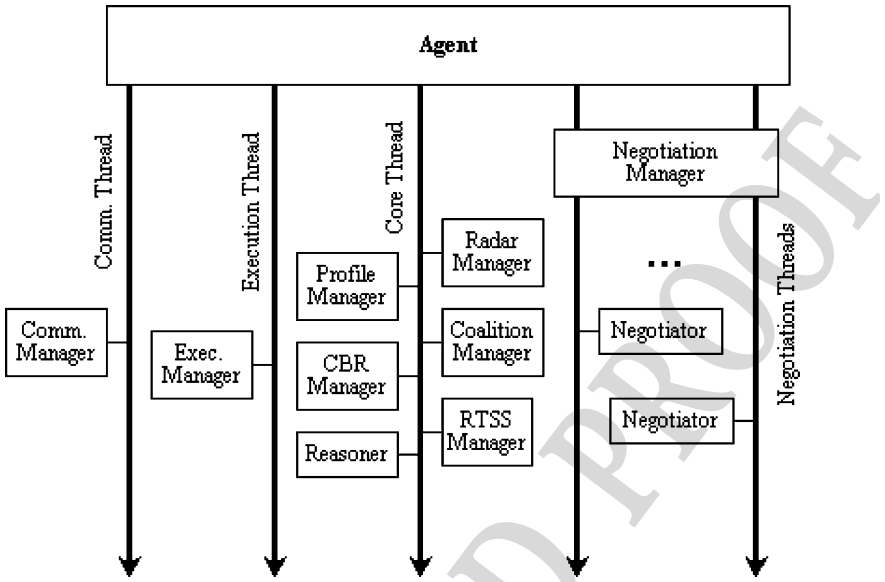

Figure 1. Our multi-threaded agent architecture.

coverage of an agent and its neighbors. When the agent encounters an event, the coalition manager forms a coalition, oversees the negotiations to recruit members, and confirms or discards a coalition depending on the status of the event and the outcome of the negotiations. The RTSS manager interacts with a real-time scheduling service (RTSS) at the kernel level, allowing the agent to monitor and request for its CPU allocation and usage. The reasoner is responsible for the general thinking tasks of the agent. For each negotiation thread, a negotiator shoulders the responsibility of carrying out communicative acts, basing its negotiation strategy on the information provided by the profile and CBR managers, monitoring time using the RTSS manager, receiving coalition information from the coalition manager, retrieving negotiation messages by checking the mailbox maintained by the communication manager, and, most importantly, executing the real-time argumentative negotiation, the main focus of this paper. Readers are referred to [50] for a detailed description of the agent architecture.[2] Note that the multi-threaded design of an agent infrastructure has been implemented elsewhere in MAS research [41, 49, 54].

3. A logical protocol for real-time argumentative agent negotiations

In this section, we describe the logical protocol for our real-time argumentative agent negotiations. Argumentative negotiations differ from traditional negotiations because the agents conducting the former negotiate about *why* one of the agents needs to perform a certain task in addition to what the task is. Therefore, our work is similar to [40]. However, we assume that agents have the same inference rules. Parsons' work assumes that agents may have different ones, and thus his argumentation protocol requires agents to exchange inference rules as well. Moreover, we incorporate real-time issues into our design guidelines.

3.1. *Real-time negotiation design guidelines*

In this section, we outline several design objectives for our argumentative model. Note that in our domain, each agent stores a local viewpoint of the world and information is distributed among the agents. For an agent to convince another it needs to exchange information with the partner during negotiations. In a real-time, dynamic, and distributed environment, the agents do not have sufficient time to perform detailed negotiations, leading to the following design objectives. We believe that these guidelines are general and applicable to real-time argumentative inter-agent negotiations, where time is critical.

We also offer a brief discussion of their operationalization in our system:

Design objective 1 A negotiation should be bounded by time.

In a real-time environment a negotiation should complete (with success or failure) within a predefined window of hard time, requiring time-bounded rationality from the agents. In our model, before a negotiation begins, the agent determines the time allotted for the negotiation based on the speed of the target and the coverage time available. If a negotiation has run beyond the allotted time window, then the agent terminates the negotiation since any deal reached after that and the subsequent task performed will no longer be useful.[3] Also, an agent should be able to react to the pace of its negotiation. If the pace of the negotiation is slow, then, an agent counter-offers to narrow the constraints further so that both agents can come to an agreement or disagreement more quickly.

Design objective 2 Each step of the negotiation should be fast.

A negotiation process consists of multiple actions (steps): parsing of and responding to messages, evaluation of offers, submission of offers, etc. Each one of these steps should be performed efficiently and quickly to achieve real-time behavior.

Design objective 3 A negotiation should be kept short – the number of iterations minimized.

If a negotiation takes more iterations, then it incurs more processing time, more communication latency, more bandwidth usage, and a higher opportunity for noise to distort the messages. Towards this end, we use ranked arguments such that the most important argument is sent over to the responding agent first, hoping to convince it sooner.

Design objective 4 A negotiation-related message should be kept short.

This is meant to reduce loss and to improve communication speed.

Design objective 5 An agent should be able to abort a negotiation unilaterally when communication fails.

In a real-time environment, the communication channels may be jammed or simply congested. An agent should be able to recognize when its partner's channel is no longer receptive, terminate the negotiation immediately, and move on to the next neighbor or task.

Design objective 6 A negotiation strategy should be determined quickly and reflectively.

A negotiation strategy guides how an agent should conduct its negotiation. It consists of a set of parameters such as the number of negotiation iterations, time allotted, ranking of arguments, etc. These values should be derived quickly such that by the time the negotiation strategy is obtained, it still is reflective of the agent's status and the target. Slow determination of the negotiation strategy renders the negotiation behavior less situated to the agent's current status.

Design objective 7 A negotiation should only be *efficiently reflective*.

During a negotiation, an agent's status changes constantly. If we check the agent's state at each time step[4] and dynamically fine-tune the negotiation parameters, then the agent is fully reflective. However, both the monitoring of the agent attributes that might affect the negotiation, and the response to changing attributes, are time consuming and may slow down the negotiation. Hence, there must be a delicate trade-off and a negotiation should only be *efficiently reflective*, that is, consider only important changes to attributes and ignore others. In our model, an agent reflectively considers only time and the need for a negotiation.

3.2. Negotiation protocol

Figure 2 shows our negotiation protocol in a state diagram between two agents: *a* and *b*. State 0 is the initial state, the double-circle. State 1 is the first *handshake* state, indicating whether the initiated negotiation will be entertained. State 4 is the initiating state while state 5 is the responding state. The initiating state is where the initiating agent, *a*, returns to, basically the processing loop of the negotiator module. The responding state is where the responding agent, *b*, returns to, respectively. Agent *a* initiates a negotiation request to *b* by sending an INITIATE message (initiate (*a,b*)), the state transitions to 1. At this juncture, there are four possible scenarios. First, agent *b* may outright refuse to negotiate by sending a NO_GO message (no_go (*b,a*)). This results in a final state of failure (state 2, rejected). Second, agent *b* may outright agree to the requested task by sending an AGREE message (agree (*b,a*)). This results in a final state of success (state 3). Third, agent *b* may decide to entertain the negotiation request and thus sends back a RESPOND message (respond (*b,a*)). This transitions the state to 4. Fourth, there may be no response from agent *b*. Thus agent *a*, after waiting for some time, has no choice but to declare a no response (no_response (*a*)) and moves to a state of failure (state 8, channel _jammed).

When the agents move to state 4, the argumentative negotiation begins and iterates between states 4 and 5 until one side opts out or both sides opt out or both sides agree. During the negotiation, (1) agent *a* provides information or arguments to *b* by sending INFO messages (info(*a,b*)), (2) agent *b* demands information or arguments from *a* by sending MORE_INFO messages (more_info(*b,a*)), (3) if agent *a* runs out of arguments, it sends a INFO_NULL message to *b* (info_null(*a,b*)), (4) if agent *b* runs out of patience, it counter-proposes by sending a COUNTER message to *a* (counter(*b,a*)), and (5) agent *a* can agree to the counter offer (agree(*a,b*)) and move to the state of

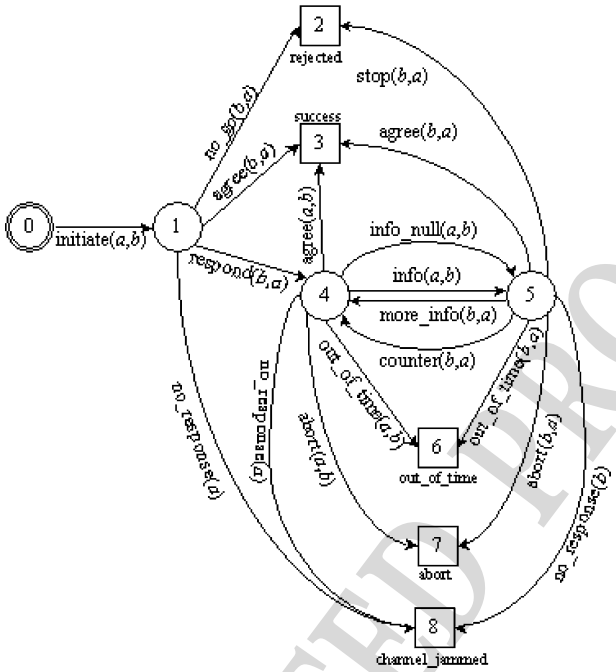

Figure 2. Our negotiation protocol. Squares are final states. The double-circle is the initial state. This state diagram shows how two negotiating agents interact based on responses to offers, and the real-time constraints of the environment.

success (state 3), or provide more information (info(*a,b*)) as requested, or provide no information (info_null(*a,b*)) if it has time to do so, hoping that agent *b* might come up with a better offer, or simply disagrees (abort(*a,b*)). Thus, an initiating agent will always negotiate until it has run out of time or when the responding agent opts out. However, an initiating agent may abort a negotiation, and this is where the conditions come into play. If the agent realizes that it has already obtained what it wants from other negotiations happening in parallel, then it aborts the current negotiation; or if the agent realizes that it no longer cares about the current negotiation, then it aborts. These conditions are based on desires and intentions, which in turns are based on beliefs of the agent. When an agent runs out of time, it issues an OUT_OF_TIME message to the other agent and quits the negotiation with a failure (state 6, out_-of_time). When an agent aborts, it issues an ABORT message to the other agent and quits the negotiation with a failure (state 7, abort). Finally, whenever an agent does not hear from the other agent within an allocated time period, it assumes that the communication channel has been jammed or congested and quits with a failure (state 8, channel_jammed). Note that we distinguish NO_GO, STOP, OUT_OF_TIME, and ABORT in the above protocol. With the above different end states, agent *a* can determine whether the negotiation has failed because it has exhausted all its arguments (STOP) or otherwise and subsequently learn from the failure.

Note that our work does not attempt to address the issues with AI in Law, especially in the dialectics in legal arguments (e.g., [21, 42]). Our ''persuasion'' is a

negotiation protocol for information exchange between two agents to reach a deal in which the initiating agent provides arguments to support its request in order to convince the responding agent to agree to the request. Hence, we do not look into the maneuvers found in legal arguments such as defenses, proposals, positions, and attacks for our design.

3.3. *Logical framework*

We use an expanded multi-context BDI agents framework of [36, 40] to describe the logical framework of our negotiation protocol. As presented in [44], there are three modalities. First, beliefs (B) represent the states of the environment and the agent. There are also belief states that arise during negotiations, as agents learn each other's beliefs and intentions. Second, desires (D) represent the motivations of the agent. Third, intentions (I) represent the goals that the agent wants to achieve. We also assume the following axioms [40]:

(1) $B : B(p \rightarrow q) \rightarrow (B(p) \rightarrow B(q))$
(2) $B : B(p) \rightarrow \neg B(\neg p)$
(3) $B : B(p) \rightarrow B(B(p))$
(4) $B : \neg B(p) \rightarrow B(\neg B(p))$

Axiom 1 states that if one believes that p implies q, then if one believes p, then one believes q as a consequence. Axiom 2 states that if one believes p, then one does not believe the negation of p. Axiom 3 states that if one believes p, then one also believes that it believes p. This ensures that an agent *knows* and believes in what it believes. Similarly, Axiom 4 states that if one does not believe p, then one believes that it does not believe p. Similar axioms for desires and intentions are:

(5) $D : D(p \rightarrow q) \rightarrow (D(p) \rightarrow D(q))$
(6) $D : D(p) \rightarrow \neg D(\neg p)$
(7) $I : I(p \rightarrow q) \rightarrow (I(p) \rightarrow I(q))$
(8) $I : I(p) \rightarrow \neg I(\neg p)$

We adopt partially the strong realist BDI agent model of [40] such that (1) an agent only intends to do what it desires, and (2) an agent only desires what it believes. However, we do not adopt other rules presented in [40], regarding the communication unit, because in our case (1) an intention for performing a task does not necessarily imply a communication of the performance of the task – in our domain, a task may sometimes be performed by the agent itself, and (2) we do not require an agent to report a completion of a task to another agent – because in our current design, an agent assumes that if another agent agrees to perform a task, it knows that that agent will try its best to perform and complete the task, and that whether it believes the task has been performed is no longer important. In the future, however, we plan to include a monitoring mechanism in an agent to enrich our real-time modeling of events and agent behavior in making better decisions. At that time, an agent would have to care whether a task has been performed successfully to update its own belief states, but would still not depend on the communicated information for the update.

We incorporate temporal logic into our protocol to explicitly define the various belief, desire, and intention states of an agent and when they are true. This is key to the real-time implementation of the protocol. To satisfy real-time constraints, our agent, as shown in Figure 1, consists of multiple concurrent processing threads. Each thread carries out a set of tasks, and these tasks access and modify the same states at different times. Some states must not be modified before certain actions have been carried out; some states should not be accessed before certain results have been obtained. To implement such a *state synchronization* across multiple concurrent processes, we use temporal logic to define when a state must be true and the duration for that state to stay true. Without the temporal logic component, it would have been close to intractable to manage the inter-thread, real-time activities.

As previously mentioned, Cohen and Levesque [9, 10] and Rao and Georgeff [43, 45] have incorporated temporal components into the BDI model. In [9], intentions are defined in terms of temporal sequences of an agent's beliefs and goals. Each possible world extendable from a current state at a particular time point is a time line representing a sequence of events. As such, the inter-modal relationships are stronger than those in [43]. For example, an agent fanatically committed to its intentions will maintain its goals until either they are believed to be achieved or believed to be unachievable. Thus, intentions are seen as a *special* class of desires. Rao and Georgeff [43], on the other hand, present an alternative possible-worlds formalism for BDI-architectures. Instead of a time line, they choose to model the world using a temporal structure with a branching time future and a single past, called a time tree, where a particular time point in a particular world is called a situation. There are three crucial elements to the formalism. First, intentions are on a par with beliefs and goals. This allows them to define different strategies of commitment and to model a wide variety of agents such as blinded, single-minded, and open-minded agents. Second, they distinguish between the choice an agent has over the actions it can perform and the possibilities of different outcomes of an action, factoring in the uncertainty that the environment brings into the determination of the outcomes. Third, they specify an interrelationship between beliefs, goals, and intentions that allows them to avoid problems such as commitment to unwanted side effects. Further, there has been work in for branching-time temporal logic such as the computation tree logic (CTL) to carry out temporal resolution [7, 45].

In our model, the incorporation of the temporal elements is different. In [9], the temporal component was used to define intentions through commitment and persistence, derived from beliefs and goals. In [43], it was used to order possible worlds from a situation in both the time and space dimensions. Each time tree denotes the optional courses of events choosable by an agent in a particular world. For example, an agent has a belief ϕ, denoted $B(\phi)$, at time point t if and only if ϕ is true in all the belief-accessible worlds of the agent at time t. In our model, however, we use the temporal component to define the temporal duration that a state needs to be stable, or needs to occur in order for another state to take place. Thus, our motivation is to help design and implement agents that are multi-threaded and multi-tasking. We do not use temporal logic to define the BDI modalities.

In the following subsections, we formalize our theory of the actions depicted in the state diagram of Figure 2.

3.3.1. Temporal logic To incorporate real-time concerns into our logical negotiation protocol, we use several interval relationships outlined in [1, 3]. Each interval t has a start time, t_s, and a finish time, t_f, and its duration is $t_f - t_s$. If $t_f - t_s$ equals the smallest amount within the resolution of the domain problem, then the interval becomes a *moment* or a *point*. There are seven basic relations between temporal intervals:

(1) *Before*(*i, j*) where interval *i* ends before interval *j*.

(2) *Meets*(*i, j*) where as soon as *i* finishes, interval *j* starts, i.e., the two intervals are consecutive.

(3) *Overlaps*(*i, j*) where a portion of interval *i* overlaps a portion of interval *j* in time and *i* starts before *j* and *i* ends before *j*.

(4) *Starts*(*i, j*) where interval *i* starts at the same time as interval *j* but interval *i* has a shorter duration.

(5) *Finishes*(*i, j*) where interval *i* finishes at the same time as interval *j* but interval *i* has a shorter duration.

(6) *During*(*i, j*) where interval *i* starts after interval *j* and interval *i* ends before interval *j*.

(7) *Equals*(*i, j*) where both intervals have the same durations and start and end at the same times. We also adopt the homogeneity axiom schema such that a proposition is homogeneous if and only if when it holds over an interval t, it also holds over any sub-interval within t. Within the framework of our negotiation protocol, the BDI states are all homogeneous propositions, hence the use of strong negation $\neg$. The predicates such as the communicative acts are anti-homogeneous [2] since, for example, the action of composing and sending a message is a process that does not generally hold unless completed in the end.

We also introduce a notational convenience $[e]$ as the maximum interval of an event/action, or, in the case of homogeneous positions, as the interval of a proposition holding true.

3.3.2 Communicative acts In our protocol, we have two sets of corresponding communicative acts. One is for receiving and parsing an incoming message; the other for composing and sending an outgoing message. For example, a communicative act that composes a negotiation request and initiates contact with a potential negotiation partner is of the following form:

$$C_{out} : initiate(i, r, Do(r, \rho), t)$$

where the predicate *initiate* is the communicative act (composing and sending), i is the initiating agent, r is the responding agent, $Do(r,\rho)$ is the requested task, i.e., "r do task, ρ" and t is the time taken for the communicative act to start and finish.

A communicative act is an event that performs a set of tasks consecutively such that the sum of the durations of the tasks is the duration of the communicative act. A communicative act may generate new propositions, may cause a new state externally to another agent, and may be terminated by another proposition or event. For example, at the end of the interval t, if the communication is successful, then the responding agent will receive the request, $C_{in} : initiate(i, r, Do(r, \rho), k)$, where, in theory, *Meets*(t,k), and

in practice (due to communication latency), *Before* (*t,k*). On the other hand, if the communication is unsuccessful (e.g., due to communication channel being jammed), then the predicate *initiate* of the initiating agent will be terminated by a terminator [56], as will be discussed further in Section 3.3.3. To simplify our discussions, we use $C_{in} : f(\langle sender \rangle, \langle receiver \rangle, \langle request \rangle, t)$ and $C_{out} : f(\langle sender \rangle, \langle receiver \rangle, \langle request \rangle, t)$ to differentiate between incoming and outgoing message handling, where f is one of the acts defined in our protocol. Currently, we have the following communicative acts, as depicted in Figure 1: *initiate*, *respond*, *no_go*, *agree*, *abort*, *out_of_time*, *counter*, *more_info*, *info*, *info_null*, and *stop*, for a total of 22. An example of a message is $C_{in} : agree(A1, A2, Do(A1, \rho), T_1)$, which means that $A2$ receives as an input message from $A1$ an agreement that states that $A1$ will perform the task ρ, and the receipt and processing of the message occurs at the time interval T_1.

One of the objects generated by a C_{out} communicative act is the message. Our message syntax is $msg(\langle sender \rangle, \langle receiver \rangle, \langle type \rangle, \langle request \rangle, \langle contents \rangle)$ where the type of the message denotes one of the communicative acts and the contents consist of whatever pertinent to the request. In the following discussion, we often mention the messages in the same breath as the communicative acts and use them interchangeably.

3.3.3. Terminator Vere [56] described a proposition as bounded by its holding true over a time interval, with a limited life span terminated by later contradictory assertions. Under Vere's definition, an assertion T is a terminator for an assertion A if and only if: (1) A and T are contradictory, (2) T follows A in time, and (3) no assertion T' exists satisfying the first two conditions such that T follows T' in time. Note that the third condition ensures that there are no conditions between A and T that might pre-empt T as a terminator. Basically, *Meets*([*A*],[*T*]).

Take our communicative acts for example. Suppose an initiating agent performs *initiate*. As we shall see later, the agent has a set of BDI states (including its belief that the communication channel is operating, $B : B_i(channel\_good, t)$) that holds true such that the collective interval, Θ, of those states overlaps the communicative act's interval, *Overlaps*([Θ], [*initiate*]). That is, the communicative act may continue after Θ no longer holds. Further, if the communication fails, which in this case means the message is not sent, the agent generates a belief $B : \neg B_i(channel\_good, k)$. This becomes a terminator of *initiate* since one of the preconditions for the communicative act and the new belief state are contradictory and *Meets*(*t,k*). As a result, the action is terminated.

Note that a proposition or an assertion in our logical negotiation protocol holds until terminated by a terminator, and so does a terminator. A termination may lead to the stoppage of a set of actions or tasks, which in turn may lead to the stoppage of a set of events. An action or task may, however, terminate by itself normally as it completes within a time interval.

3.3.4. "CanDo," "Do," and "Doing," and time Here we define the predicates *CanDo*, *Do*, and *Doing* and their association with time. Basically, if an agent believes that it can perform a requested task, then it is willing to be convinced to desire to perform that task. Once the agent has the desire, it intends to perform the task and believes that it will be performing the task soon. Once the agent is performing the

task, it continues to do so until the task is completed or until the agent no longer intends to perform the task or until it does not believe it can perform the task anymore. We present these three predicates since they are related to the underlying motivations of our negotiations.

Predicate 1 CanDo

When, at time t_1, an α agent believes it can do a task ρ, and it holds this belief until time t_2 when a terminator contradicts the belief, we have $B_\alpha(CanDo(\alpha, \rho), t_B)$ where t_B is the interval with the start time t_1 and the end time t_2.

Predicate 2 Do

When, an agent α desires to do a task ρ, then we must have $B_\alpha(CanDo(\alpha, \rho), t_B)$ and $D_\alpha(Do(\alpha, \rho), t_D)$ such that $During(t_D, T_B)$. That is, the desire to do must be supported by a *CanDo* belief. Following a strong realist model, we also must have the following: $D_\alpha(Do(\alpha, \rho), t_D \Rightarrow I_\alpha(D_o(\alpha, \rho), t_I)$ such that $During(t_1, T_D)$. Note that we simplify this example by assuming that the only way to satisfy the desire to do a task is to perform the task. From this definition, an agent may no longer intend to perform a task even when it desires to. This is because our agents are susceptible to external states such as the communicative acts and the sensory information. In cases where the intended task is one of the many choices available to satisfy a desire, then agents are also susceptible to internal states. For example, suppose an agent desires to be not hungry, and it has two choices: *Cook* or *Go-To-Restaurant*. And, suppose initially, the agent chooses to cook at home: $D_\alpha(\neg hungry(\alpha), t_D \wedge I_\alpha(Cook(\alpha), t_I)$. But after cooking for a time of t_c, such that $During(t_c,t_I)$, the agent realizes that the faucet stops working and there is no water in its kitchen, which constitutes a termination condition to *Cook*, thus $I_\alpha(\neg Cook(\alpha), t'_I)$ but $D_\alpha(\neg hungry(\alpha), T_D)$. This allows the agent to replan and to $I_\alpha(Go - To - Restaurant(\alpha), t''_I)$.

Predicate 3 Doing

If an agent α intends to perform a task, then it believes that it is performing the task in the following temporal relationship: $I_\alpha(Do(\alpha, \rho), t_I) \Rightarrow B_\alpha(Doing(\alpha, \rho), t'_B)$ such that $Overlaps(t_I, t'_B) \vee During(t'_B, t_I)$. That means, an agent may still believe that it is performing a task that it no longer intends to perform. In Vere's [56] definition, $I_\alpha(Do(\alpha, \rho), t_I)$ is a trigger pre-condition. For example, an agent first intends to re-orient its sensors. During the re-orientation, it receives other information and decides to no longer intend to do so. However, once a re-orientation is launched, the agent has to wait for it to end. This type of task is considered to be *atomic* and *non-interruptible*, and this situation is observed in our domain of application.

3.3.5. Coalition formation When an agent, for example, senses a target, it needs to form a coalition from which to ask for help. To do so, it collects all its current BDI states and external information to obtain a list of potentially helpful neighbors. Since each task is new, that means in the beginning of the coalition formation, there exist no belief states such that $B : B_\alpha(CanDo(n, \rho), t_B)$ where n is a member of the set of all known neighbors, N_α, of the agent α and ρ is the new task. After a domain-specific search process,[5] the agent obtains a list of potentially helpful neighbors, $N'_\alpha \subseteq N_\alpha$. At

this point, equipped with this list, the agent $\forall nB : B_\alpha(CanDo(n, \rho), t_B)$ where $n \in N'_\alpha$. Then the agent checks its needs and creates desires for only a certain number of these neighbors to perform the tasks (since enlisting everybody one knows to help out a task is counter-productive). In our actual agent design, these neighbors are ranked according to their utility values.[6] The agent then checks the number of available negotiation threads that it may use to negotiate with these neighbors. That number trims N'_α to obtain N''_α. At this point, the agent $\forall nD : D_\alpha(Do(n, \rho), t_D)$ where $n \in N''_\alpha$. Subsequently, as the agent begins to send out requests, with each successful contact, the agent forms $I : I_\alpha(Negotiate(n, Do(n, \rho)), t_I)$ with the neighbor n that has made contact. In the end, all neighbors contacted are in the coalition $C_\alpha(\vec{\rho})$ such that $C_\alpha(\vec{\rho}) \subseteq N''_\alpha$ and where $\vec{\rho} = \{\rho_1, \rho_2, \cdots, \rho_{|C_\alpha(\vec{\rho})|}\}$ is a set of subtasks that contribute to the original task. In our model, we define subtasks of a task in terms of the resources required. For example, in our application to a distributed sensor network, the task of tracking a target requires three subtasks: each subtask requires an agent to turn on its sensor towards a particular direction at a given start time. In general, though not required in our model, subtasks could be seen as horizontal partitioning of the overall task. In this way, each neighbor is contacted to perform a subtask and a coalition formation drives an agent to negotiate with its neighbors.

In terms of the temporal interval relationships, since a negotiation process is stepwise and interruptible, if an agent does not have the desire for a neighbor to perform a task, then it does not intend to negotiate with that neighbor regarding that particular task. Thus, we have the following condition: $During(t_D, t_B) \wedge During(t_I, t_D)$.

Note that in our model, we have N 1-to-1 negotiations but do not conduct *direct* 1-to-N negotiations. That is, during a negotiation, a negotiation thread does not consult directly other negotiation threads of the same agent. However, the parent agent of the negotiation threads does examine the completion status of its negotiation threads and may change the beliefs, desires, and intentions of the negotiation threads through negotiation-related predicates (Section 3.4) due to the results of other negotiations. This design choice is motivated by real-time concerns. Instead of having the negotiator module of a negotiation thread monitoring the activities of other negotiation threads of the same agent, the core thread of the agent monitors the negotiation activities through the coalition manager. The coalition manager determines whether a coalition is still viable, whether a coalition has been achieved, and whether a coalition is to be aborted, and commands each individual negotiation thread accordingly. Thus, each negotiator can concentrate on their negotiation task at hand.

Here we briefly describe our coalition formation approach. Interested readers are referred to [50]. Our coalition formation consists of three stages: (1) coalition initialization where an agent obtains a ranked list of potentially helpful neighbors, $N'_\alpha \subseteq N_\alpha$ based on the current problem, (2) coalition finalization where the agent contacts the neighbors in $N'_\alpha \subseteq N_\alpha$ to negotiate, and (3) coalition acknowledgment where the agent concludes the success or failure of the coalition and inform neighbors who have agreed to help. In coalition initialization, a neighbor is ranked based on its potential utility in helping with the current problem. This potential utility is based on the past and current relationships between the agent and the neighbor, and the ability of the neighbor with the current problem. For

example, if the target is moving towards the sensor coverage of the neighbor and will be inside the coverage for a long time, then the neighbor has a high potential utility. During the coalition finalization step, the agent negotiates with the ranked neighbors concurrently. To negotiate, each agent uses CBR to derive an appropriate strategy to deal with each neighbor. As each negotiation thread reports its final status to the core thread (the parent agent), the parent agent decides whether to abort meaningless negotiations or to modify negotiation tactics. After the finalization step, the agent knows whether it has a coalition. If it does, it sends a confirmation message to all the neighbors who have agreed to help. If it does not, it sends a discard message to those who have agreed to help. This is the acknowledgment step.

3.3.6. Implicit assumptions As discussed in Section 2.1, our agents are cooperative and are also directed to satisfy global goals. Each is motivated to look for help from its neighbors and to entertain negotiation requests from its neighbors, and each genuinely wishes to have a successful negotiation. First, we have the implicit assumption $D : D_\alpha(Cooperate, t_{always})$ where $t_{always,s} = time0$ is the start time and $t_{always,f} = \infty$ is the finish time, meaning that the desire is always true. Second, we have $D : D_\alpha(Satisfy - Global - Goal, t_{always})$. These assumptions are the meta-level knowledge in an agent's context.

When an initiating agent, i, negotiates with a neighbor, r, the agent has the following BDI states, as shown in the previous section:

$$D : D_i(Cooperate, t_{always}) \wedge B : B_i(CanDo(r, \rho), t_B) \wedge$$
$$D : D_i(Do(r, \rho), t_D) \wedge I : I_i(Negotiate(r, Do(r, \rho)), t_I).$$

Note that in [36, 40, 44], the mental states were *layered* or divided into different levels: beliefs, desires, and intentions, and bridge rules were used to transition across these levels. Here we simply outline a set of *propositions*, linking the multiple contexts together in conjunctions. The key here is that each proposition is also quantified by a time interval. So, in a way, our model indicates that at certain time intervals, there are certain mental states (beliefs, desires, or intentions) that are true simultaneously (if their time intervals overlap). Thus, the above conjunction simply means that for some time interval there is a belief of CanDo, for some other interval, there is a desire of Do, and so on.

Combining the above states with $D : D_i(Satisfy - Global - Goal, t_{always})$, we assume that each agent:

$$I : I_i(succeed(Negotiate(r, Do(r, \rho))), t_I).$$

The above intention motivates an initiating agent to continue negotiating. In the later discussion, we use this intention to explicitly drive the negotiation axioms, while keeping other BDI states implicit.

When a responding agent, r, receives a request to negotiate from an initiating agent, i, the agent parses the message and examines its own current states. If it believes it can perform the requested task yet does not have the desire to do so, then because of the desire to cooperate, it has the following BDI states:

$$D : D_r(Cooperate, t_{always}) \wedge B : B_r(CanDo(r, \rho), t_B) \wedge$$
$$\neg \exists D : D_r(Do(r, \rho), t_D) \wedge I : I_r(Negotiate(i, Do(r, \rho)), t_I).$$

Similarly, combining the above states with $D : D_r(Satisfy - Global - Goal, t_{always})$, we have the following that keeps the agent negotiating:

$$I : I_r(succeed(Negotiate(i, Do(r, \rho))), t_I).$$

A completely successful negotiation results in $D : D_r(Do(r, \rho), t_D)$. The negotiating strategy of an initiating agent is to help the responding agent achieve $D : D_r(Do(r, \rho), t_D)$ while that of the responding agent is to let itself be persuaded by the initiating agent's arguments in order to achieve $D : D_r(Do(r, \rho), t_D)$. As we will see in the next two sections, there are partially successful negotiations and different types of failures.

Furthermore, we have to deal with the duration of a BDI state. A BDI state holds true only sufficiently long: (1) no longer than the duration of the task to be performed, or (2) until ended by a terminator. For the implicit assumptions, the tasks *Cooperate* and *Satisfy-Global-Goal* have infinite duration. So, in $D : D_r(Do(r, \rho), t_D)$, the agent r only desires to do the task for at most the duration of the task ρ. Thus, we have $During(t_D, [Do(r, \rho)])$ where $[Do(r, \rho)]$ is the time interval for r performing the task. This assumption applies to all BDI states related to performing a task. On the other hand, when a negation of performing a task is involved, such as $D : D_r(\neg Do(r, \rho), t_D)$, the agent cannot rely on $[Do(r, \rho)]$ to quantify t_D. In our design, instead of making t_D a *moment* or a *point*, we let it be until terminated by another assertion/proposition (e.g., generated by a later coalition process). These assumptions allow an agent to negotiate regarding the same task at two different times as long as the task negotiated first has completed since the BDI states of performing a task are self-terminating and those of *not* performing a task are terminated by assertions generated by other agent activities.

Finally, we have to deal with arguments since our negotiation approach is argumentation-based. Suppose the request is for the responding agent r to perform the task $\rho : Do(r, \rho)$. The responding agent has a set of internal arguments, γ_r, for and against performing the task. If the agent believes it can perform the task but $\Gamma_r \; D_r \; (Do(r, \rho), t_D)$ (meaning the arguments do not support the desire for performing the task), then it has to rely on the initiating agent for more arguments. The initiating agent has its own set of arguments, γ_i, for the responding agent performing the task. The underlying approach is to send over a subset Γ_i' of γ_i to the responding agent until

$$\Gamma_r \cup \Gamma_i' \models D_r(Do(r, \rho), t_D) \text{ (in which case the negotiation succeeds),}$$

or until

$$\Gamma_r \cup \Gamma_i \; D_r \; (Do(r, \rho), t_D) \text{ (in which case the negotiation fails),}$$

where $\Gamma_i' \subseteq \Gamma_i$ is the set of arguments already communicated to the responding agent from the initiating agent. This assumption is a critical element in our negotiation protocol as it facilitates a stepwise evaluation of arguments to move closer to a

conclusion of the negotiation. An example of arguments supporting a requested task in the sensor network application includes the approximate direction of the target detected, the current, activated sensor sector, and the helpfulness of the agent to its neighbors. The approximate direction of the detected target would help the responding agent deliberate whether it has available sensor sector for the task, for example.

3.3.7. Initiating behavior Here we outline the axioms that link an agent's communication and its internal states for conducting negotiations as an initiating agent. Before moving further, we would like to note that the implications in the following axioms are basically bridge rules. The pre-conditions of these rules do give the impression that multiple contexts are present. As previously described in Section 3.3.6, we mean to have them as separate contexts which happen to be conditions required to be present to trigger the rules. The formulation of these contexts in the pre-conditions allows us to develop a time schedule for the various mental states to hold true.

Initiate: When an initiating agent (i) believes that it intends to negotiate with the responding agent (r) to perform a task ρ, it initiates a negotiation request to the responding agent.

$$I : I_i(Negotiate(r, Do(r, \rho)), t_I) \Rightarrow C_{out} : initiate(i, r, Do(r, \rho), t_{cout,initiate})$$

where $During(t_{cout,initiate}, t_I)$. The predicate *initiate* encapsulates the act of composing an INITIATE-type message and sending the message to agent r.

Failure 1: When an initiating agent (i) receives a NO_GO message from a responding agent (r), it believes that the responding agent r cannot perform the requested task ρ and stops intending r to perform the task.

$$\begin{aligned} C_{in} : no\_go(r, i, Do(r, \rho), t_{cin,no\_go}) \wedge I : I_i(Negotiate(r, Do(r, \rho)), t_I) \Rightarrow \\ I : I_i(\neg Negotiate(r, Do(r, \rho)), t'_I) \wedge B : B_i(\neg CanDo(r, \rho), t_B) \wedge \\ D : D_i(\neg Do(r, \rho), t_D) \wedge rejected \end{aligned}$$

where $During(t_{cin,no\_go}, t_I) \wedge Meets(t_I, t'_I) \wedge Meets(t_I, t_B) \wedge Starts(t_B, t_D)$. The *no_ go* communicative act is the encapsulation of receiving and parsing a NO_GO (an outright refusal to negotiate) message. This rule allows an agent to move on to the next neighbor after the responding agent has outright refused to negotiate. This is real-time motivated:[7] instead of trying to come up with a lesser task and trying to establish a negotiation with the responding agent, the initiating agent simply gives up and shifts its focus to other neighbors.

The proposition *rejected* indicates the failure of a negotiation.

Note also that in the above rule, the changes in the internal states of the agent are triggered by the incoming message. This is one of our design characteristics and goals: agents communicate only when necessary since the environment is real-time and resource constrained, and information is exchanged only during negotiation.

Success 1 When an initiating agent (i) receives an AGREE message from a responding agent (r), it believes that the responding agent r intends to perform and will perform the requested task ρ.

$$C_{in} : agree(r, i, Do(r, \rho), t_{cin,agree}) \wedge I : I_i(Negotiate(r, Do(r, \rho)), t_I) \Rightarrow$$
$$B : B_i(D_r(Do(r, \rho)), t_B) \wedge I : I_i(\neg Negotiate(r, Do(r, \rho)), t'_I) \wedge$$
$$D : D_i(\neg Do(r, \rho), t_D) \wedge success$$

where $During(t_{cin,agree}, t_I) \wedge Meets(t_I, t'_I) \wedge Meets(t_I, t_B) \wedge Starts(t_B, t_D)$. The *agree* communicative act is the encapsulation of receiving and parsing an AGREE message. In this axiom, if an agent receives an AGREE message from the responding agent, then (1) it believes that the responding agent desires to perform the task, (2) it intends no longer to negotiate, and (3) it desires no longer that the responding agent perform the task. This third desire may seem counter-intuitive at first glance. Its purpose is to say "If I believe that you have the desire to do the task, then I don't have to desire you to do the task anymore," and that does not prevent the agent to desire the responding agent to perform the task in the future.

The proposition *success* indicates the success of a negotiation.

Info 1 When an initiating agent (i) receives a RESPOND message from a responding agent (r), it (1) believes that the responding agent r intends to negotiate and (2) intends to obtain a successful negotiation. Consequently, the initiating agent i intends to help r to desire to perform the task by supplying available necessary information.

$$C_{in} : respond(r, i, Do(r, \rho), t_{cin,respond}) \wedge I : I_i(Negotiate(r, Do(r, \rho)), t_I) \wedge$$
$$B : B_i(I_r(Negotiate(i, Do(r, \rho))), t_B)$$
$$\wedge I : I_i(succeed(Negotiate(r, Do(r, \rho))), t'_I) \wedge \exists p : (p \in \Gamma_i \wedge p \notin \Gamma'_i)$$
$$\Rightarrow C_{out} : info(i, r, Do(r, \rho), t_{cout,info})$$

where

$$During(t_{cin,respond}, t_I) \wedge Finishes(t'_I, t_I) \wedge Equals(t_B, t'_I) \wedge$$
$$Starts(t_{cout,info}, t'_I) \wedge Before(t_{cin,respond}, t_{cout,info}).$$

The communicative act *respond* is the encapsulation of receiving and parsing a RESPOND message, and the communicative act *info* is the encapsulation of composing and sending an INFO message, including selecting a p from the set of arguments, γ_i, that is not a member the set of arguments already sent, Γ'_i to r. Clause 1 of the axiom explicitly derives the motivation for the initiating agent to continue negotiating as it intends to have a successful negotiation since it now believes that the responding agent intends to negotiate as well. Clause 2 of the axiom drives the agent to send over more arguments to help bring a successful conclusion to the negotiation.

Note that as discussed in Section 3.3.2, the syntax of a message is $msg(\langle sender\rangle, \langle receiver\rangle, \langle type\rangle, \langle request\rangle, \langle contents\rangle)$. When sending out an INFO-type message, the $\langle contents\rangle$ holds the arguments.

Info_null 1 When an initiating agent (i) receives a RESPOND message from a responding agent (r), it (1) believes that the responding agent r intends to negotiate and (2) intends to obtain a successful negotiation. However, if i has run out of information or arguments, then it notifies r that it can no longer provide arguments.

$$C_{in} : respond(r, i, Do(r, \rho), t_{cin,respond}) \wedge I : I_i(Negotiate(r, Do(r, \rho)), t_I)$$
$$\wedge B : B_i(I_r(Negotiate(i, Do(r, \rho))), t_B)$$
$$\wedge I : I_i(succeed(Negotiate(r, Do(r, \rho))), t'_I) \wedge \exists p : (p \in \Gamma_i \wedge p \notin \Gamma'_i)$$
$$\Rightarrow C_{out} : info\_null(i, r, Do(r, \rho), t_{cout,info\_null})$$

where

$$During(t_{cin,respond}, t_I) \wedge Finishes(t'_I, t_I) \wedge Equals(t_B, t'_I)$$
$$\wedge Starts(t_{cout,info\_null}, t'_I) \wedge Before(t_{cin,respond}, t_{cout,info\_null}).$$

This rule is the counterpart to *Info 1* previously discussed. When an agent runs out of arguments, it notifies the responding agent about it. Instead of giving up on the negotiation right away – since obviously the initiating agent knows that it has not been able to persuade the responding agent and now it has run out of arguments, the initiating agent informs the responding agent of its situation and hopefully the responding agent will be able to counter-offer. So, in a way, this shifts the responsibility of achieving a successful negotiation to the responding agent from the initiating agent. Up until this point, the initiating agent has been responsible for keeping the negotiation going by supplying arguments/information to the responding agent, trying to convince it. Finally when the initiating agent can argue no further, the decision shifts to the responding agent.

Info 2 When an initiating agent (i) receives a MORE_INFO message from a responding agent (r), it simply supplies more unused arguments.

$$C_{in} : more\_info(r, i, Do(r, \rho), t_{cin,more\_info})$$
$$\wedge I : I_i(succeed(Negotiate(r, Do(r, \rho))), t'_I)$$
$$\wedge \exists p : (p \in \Gamma \wedge p \notin \Gamma'_i) \Rightarrow C_{out} : info(i, r, Do(r, \rho), t_{cout,info})$$

where $During(t_{cin,more\_info}, t'_I) \wedge During(t_{cout,info}, t'_I) \wedge Before(t_{cin,more\_info}, t_{cout,info})$. This axiom is similar to *Info 1*.

Info_null 2 When an initiating agent (i) receives a MORE_INFO message from a responding agent (r), if it does not have any more arguments, then it notifies r of its status.

$$C_{in} : more\_info(r, i, Do(r, \rho), t_{cin,more\_info}) \wedge$$
$$I : I_i(succeed(Negotiate(r, Do(r, \rho))), t'_I)$$
$$\wedge \neg\exists p : (p \in \Gamma_i \wedge p \notin \Gamma'_i) \Rightarrow C_{out} : info\_null(i, r, Do(r, \rho), t_{cout,info\_null})$$

where $During(t_{cin,more\_info}, t'_I) \wedge During(t_{cout,info\_null}, t'_I) \wedge Before(t_{cin,more\_info}, t_{cout,info\_null})$. This axiom is similar to *Info_null 1*.

Info 3 When an initiating agent (i) receives a counter-offer (ρ') from a responding agent (r), i believes that r desires to perform ρ'. However, if ρ' is not acceptable, then the agent i continues to send unused arguments to r.

$$C_{in} : counter(r, i, Do(r, \rho), t_{cin,counter}) \wedge B : B_i(D_r(Do(r, \rho')), t_B)$$
$$\wedge B : B_i(\neg acceptable(\rho'), t'_B)$$
$$\wedge \exists p : (p \in \Gamma_i \wedge p \notin \Gamma'_i) \wedge I : I_i(succeed(Negotiate(r, Do(r, \rho))), t'_I)$$
$$\Rightarrow C_{out} : info(i, r, Do(r, \rho), t_{cout,info})$$

where

$$During(t_{cin,counter}, t'_I) \wedge Meets(t_{cin,counter}, t_B) \wedge During(t'_B, t'_I)$$
$$\wedge Finishes(t'_B, t_B) \wedge Meets(t_B, t_{cout,info})$$
$$\wedge During(t_{cout,info}, t'_I).$$

This axiom is similar to *Info 1*. The communicative act *counter* is the encapsulation of receiving and parsing a COUNTER-type message, in which ⟨*contents*⟩ holds the counter-offer ρ'. Clause 1 states that when an initiating agent receives a counter-offer from the responding agent, it believes that the responding agent desires to perform the counter-offer. Now, the initiating agent checks the acceptability of the counter-offer (see Section 5). If the counter-offer is not acceptable and the agent still has unused arguments, then it sends over more arguments. This is how an initiating agent counter-offers a counter-offer: sending over more arguments in hope that the responding agent will come back with a better counter-offer, closer to the original request.

Note also the temporal relationships among $t_{cout,info}$, t_B, and t'_B. As soon as the initiating agent realizes that the counter-offer is not acceptable, both its beliefs that the responding agent desires to perform the counter-offer and that the counter-offer is unacceptable terminate and trigger the communicative act *info*. In other words, when the initiating agent counters a counter-offer, all beliefs regarding the counter-offer no longer hold.

Info_null 3 When an initiating agent (i) receives a counter-offer (ρ') from a responding agent (r), i believes that r desires to perform ρ'. However, if ρ' is not acceptable and the agent does not have any more unused arguments, it notifies r that it can no longer provide arguments.

$$C_{in} : counter(r, i, Do(r, \rho), t_{cin,counter}) \wedge B : B_i(D_r(Do(r, \rho')), t_B)$$
$$\wedge B : B_i(\neg acceptable(\rho'), t'_B) \wedge$$
$$\neg\exists p : (p \in \Gamma_i \wedge p \notin \Gamma'_i) \wedge I : I_i(succeed(Negotiate(r, Do(r, \rho))), t'_I)$$
$$\Rightarrow C_{out} : info\_null(i, r, Do(r, \rho), t_{cout,info\_null})$$

where

$$During(t_{cin,counter}, t'_I) \wedge Meets(t_{cin,counter}, t_B) \wedge During(t'_B, t'_I)$$
$$\wedge Finishes(t'_B, t_B) \wedge Meets(t_B, t_{cout,info\_null}) \wedge During(t_{cout,info\_null}, t'_I).$$

This axiom is the counterpart of *Info 3*.

Success 2 When an initiating agent (i) receives a counter-offer (ρ') from a responding agent (r), i believes that r desires to perform ρ'. If ρ' is acceptable, then i agrees.

$$C_{in} : counter(r, i, Do(r, \rho), t_{cin,counter}) \wedge B : B_i(D_r(Do(r, \rho')), t_B)$$
$$\wedge B : B_i(acceptable(\rho'), t'_B) \wedge I : I_i(succeed(Negotiate(r, Do(r, \rho))), t'_I)$$
$$\Rightarrow C_{out} : agree(i, r, Do(r, \rho), t_{cout,agree}) \wedge B : B_i(\neg CanDo(r, \rho), t''_B)$$
$$\wedge D : D_i(\neg Do(r, \rho), t_D) \wedge$$
$$I : I_i(\neg Negotiate(r, Do(r, \rho)), t''_I) \wedge I : I_i(\neg succeed(Negotiate(r, Do(r, \rho))), t''_I)$$
$$\wedge success$$

where

$$During(t_{cin,counter}, t'_I) \wedge During(t'_B, t'_I) \wedge Finishes(t'_B, t_B) \wedge Meets(t_B, t_{cout,agree})$$
$$\wedge Meets(t'_I, t''_I) \wedge Meets(t_B, t''_B) \wedge Starts(t_D, t''_I) \wedge During(t_{cout,agree}, t''_I).$$

The communicative act *agree* is the encapsulation of composing and sending an AGREE message, in which ⟨*contents*⟩ holds the counter-offer ρ'. With this axiom, as soon as the initiating agent agrees to a counter-offer by the responding agent, it (1) believes that the responding agent cannot do the originally requested task, (2) desires no longer that the responding agent performs the task, (3) intends to negotiate no further regarding the task, and (4) intends no longer to have a successful negotiation regarding the task. However, the negotiation still ends with a *success* tag because even though the initiating agent does not get what it wanted originally, it does obtain a portion of its original request.

Failure 2 When an initiating agent (i) receives a STOP message from a responding agent (r), it believes that the responding agent r does not desire to perform the requested task ρ and thus stops negotiating with r to perform the task, and the negotiation fails.

$$C_{in} : stop(r, i, Do(r, \rho), t_{cin,stop}) \wedge I : I_i(succeed(Negotiate(r, Do(r, \rho))), t'_I) \Rightarrow$$
$$B : B_i(\neg CanDo(r, \rho), t_B) \wedge D : D_i(\neg Do(r, \rho), t_D)$$
$$\wedge I : I_i(\neg Negotiate(r, Do(r, \rho)), t''_I)$$
$$\wedge I : I_i(\neg succeed(Negotiate(r, Do(r, \rho))), t''_I) \wedge rejected$$

where $During(t_{cin,stop}, t'_I) \wedge Meets(t'_I, t_B) \wedge Meets(t'_I, t_D) \wedge Meets(t'_I, t''_I)$. The communicative act *stop* is the encapsulation of receiving and parsing a STOP message. This rule is similar to *Failure 1*. In addition, it also states that the agent intends to not negotiate. In our current design we do not differentiate between an outright failure (failure type 1) and an opt-out failure (failure type 2) – both end with a *rejected* tag.

Failure 3I This rule is similar to *Failure 2* except for that it deals with an ABORT message and ends with an *abort* tag. See Appendix A.

Failure 4I This rule is similar to *Failure 2* except for that it deals with an OUT_OF_TIME message and ends with an *out_of_time* tag. See Appendix A.

Abort I When an initiating agent (i) no longer intends to negotiate with a responding agent (r) to perform a requested task ρ, it aborts the negotiation.

$$I : I_i(\neg Negotiate(r, Do(r, \rho)), t'_I) \Rightarrow C_{out} : abort(i, r, Do(r, \rho), t_{cout,abort}) \wedge$$
$$D : D_i(\neg Do(r, \rho), t_D) \wedge I : I_i(\neg succeed(Negotiate(r, Do(r, \rho))), t''_I) \wedge abort$$

where $During(t_{cout,abort}, t'_I) \wedge During(t_{cout,abort}, t_D) \wedge During(t_{cout,abort}, t''_I) \wedge Finishes(t''_I, t'_I)$. The communicative act *abort* is the encapsulation of composing and sending an ABORT message. This rule says that if an initiating agent aborts a negotiation, then it informs the responding agent.

Out_of_time I When an initiating agent (i) runs out of its allocated time for the negotiation with a responding agent (r) to perform a requested task ρ, it aborts the negotiation.

$$B : B_i(\neg time(Negotiate(r, Do(r, \rho))), t_B)$$
$$\wedge I : I_i(succeed(Negotiate(r, Do(r, \rho))), t'_I) \Rightarrow$$
$$C_{out} : out\_of\_time(i, r, Do(r, \rho), t_{cout,out\_of\_time}) \wedge I : I_i(\neg Negotiate(r, Do(r, \rho)), t''_I) \wedge$$
$$I : I_i(\neg succeed(Negotiate(r, Do(r, \rho))), t''_I) \wedge out\_of\_time$$

where $During(t_B, t'_I) \wedge During(t_{cout,out\_of\_time}, t''_I) \wedge Meets(t'_I, t''_I)$. The *time* predicate encapsulates the acts of obtaining and comparing the time elapsed for the negotiation against the time allocated for the negotiation (Section 3.4). The communicative act *out_of_time* predicate is the encapsulation of composing and sending an OUT_OF_TIME message to the responding agent. This rule states that when the agent has run out of time allocated for the negotiation, it no longer intends to negotiate. This is real-time motivated.

No_response I When an initiating agent (i) detects receives no response from a responding agent (r) during a negotiation, then it unilaterally quits the negotiation with a failure.

$$B : B_i(no\_response(r), t_B) \wedge I : I_i(succeed(Negotiate(r, Do(r, \rho))), t'_I) \Rightarrow$$
$$I : I_i(\neg Negotiate(r, Do(r, \rho)), t''_I) \wedge I : I_i(\neg succeed(Negotiate(r, Do(r, \rho))), t''_I)$$
$$\wedge channel\_jammed$$

where $During(t_B, t'_I) \wedge Meets(t'_I, t''_I)$. The *no_response* predicate is one of our real-time enabling functional predicates to be discussed in Section 3.4. This axiom allows an agent to bail out of a negotiation when the negotiation partner fails to respond.

Figure 3 shows the time lines of the initiating agent's behavior when faced with an outright rejection or agreement. These are the simple cases of the axioms above. The length of the process is based on $t_{cout,initiate}$, t_{cin,no_go}, and $t_{cin,agree}$. During such time, $I : I_i(Negotiate(r, Do(r, \rho)), t_I)$ holds true. After that, the intention can be removed or modified.

Figure 4 shows the negotiation process, from the initiating agent's point of view once the responding agent agrees to negotiate. The negotiation process is a manifestation of the axioms discussed above, proving a flow of communicative acts and BDI states that drives the completion of the negotiation. It is with the temporal BDI

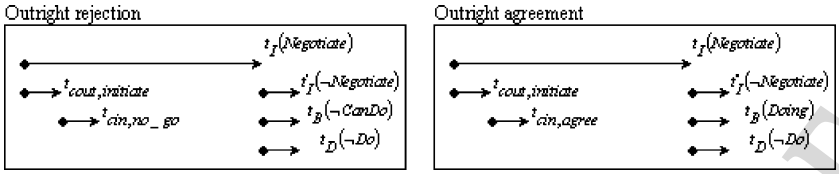

Figure 3. Time lines of the initiating agent's behavior when faced with an outright rejection or agreement.

axioms that we are able to produce Figure 4, an explicit outline of the interactions of the communicative acts with various BDI states – specifying when and how longs certain states must hold true, cannot be modified, can change, can be accessed, or are of no concern. It is also through the axioms that we are able to guarantee the completion of a negotiation process within a certain time. For example, if the initiating agent goes through the following steps: *initiate*, parse *respond*, *info*, parse *more_info*, *info*, parse *more_info*, *info_null*, parse *counter*, check to see whether the counter-offer is acceptable, and agree, then we know how much time it takes to do so by summing up the temporal intervals associated with each step. The acceptability of the counter-offer has to be held constant throughout the agreement step. This explicit declaration of time constraints is important since each of our agents is

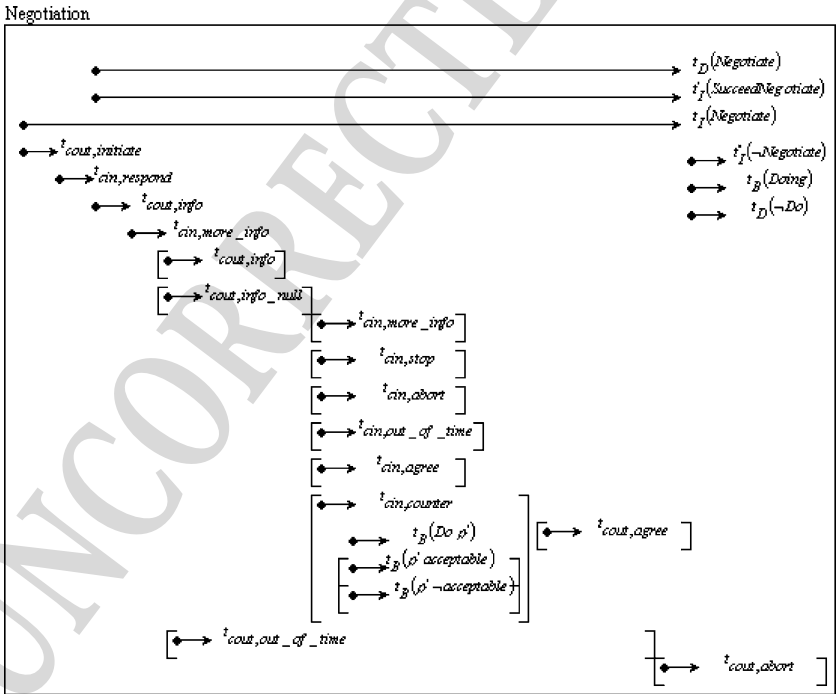

Figure 4. Time lines of the negotiation from the initiating agent's point of view (a simplified version). Temporal intervals under brackets are options. Note that once the initiating agent receives a RESPOND message from the responding agent, it is committed to negotiate successfully since it believes that the responding agent desires to negotiate.

multi-threaded, where several threads may access and need to modify the same variable at the same time. Without the axioms, a variable such as the acceptability of the counter-offer might *accidentally* be modified by another negotiation thread, rendering the current negotiation process ambiguous. Moreover, with the temporal intervals, we are able to fine-tune the system by observing the time usage of each communicative act for speedup. For example, with the BDI states, we know the minimal time we need to hold the value of a variable constant. The shorter the time needed for a variable to be held constant, the more frequent the variable can be updated and accessed, allowing other threads to proceed.

3.3.8. Responding behavior Here we outline axioms that link an agent's communication and its internal states for conducting negotiations as a responding agent.

No_go When a responding agent (r) believes that it cannot perform a requested task ρ from an initiating agent i, it outright refuses to negotiate.

$$C_{in} : initiate(i, r, Do(r, \rho), t_{cin,initiate}) \wedge B : B_r(\neg CanDo(r, \rho), t_B) \Rightarrow$$
$$C_{out} : no\_go(r, i, Do(r, \rho), t_{cout,no\_go}) \wedge exit$$

where $Meets(t_{cin,initiate}, t_B) \wedge During(t_{cout,no\_go}, t_B)$. The communicative act *no_go* encapsulates the act of composing a NO_GO message and sending the message to agent i. Agents are responsible in that if a responding agent refuses to negotiate, it informs the initiating agent.

Agree 1 When a responding agent (r) (1) believes that it is already performing a requested task ρ or (2) desires to perform ρ, it agrees to perform the task.

$$C_{in} : initiate(i, r, Do(r, \rho), t_{cin,initiate}) \wedge (B : B_r(Doing(r, \rho), t_B) \vee D : D_r(Do(r, \rho), t_D))$$
$$\Rightarrow C_{out} : agree(r, i, Do(r, \rho), t_{cout,agree}) \wedge D : D_r(Do(r, \rho), t'_D) \wedge success$$

where

$$(Overlaps(t_{cin,initiate}, t_B) \wedge Before(t_{cin,initiate}, t_{cout,agree}) \wedge During(t_{cout,agree}, t_B)$$
$$\wedge Meets(t_{cout,agree}, t'_D))$$
$$\vee (Overlaps(t_{cin,initiate}, t_D) \wedge Before(t_{cin,initiate}, t_{cout,agree}) \wedge Finishes(t_{cout,agree}, t_D)$$
$$\wedge Meets(t_D, t'_D)).$$

The communicative act *agree* encapsulates the acts of composing an AGREE message and sending the message to the initiating agent. The rule *extends temporally* the desire of the agent to continue performing the task. As previously discussed in Section 3.3.4, an agent may be still performing a task while it no longer desires to do so because a task may be atomic and non-interruptible. So, when the responding agent realizes the initiating agent requests for the same task to be performed, then it re-asserts its desire to ensure the continuation of the task. We also use the relation $Meets(t_D, t'_D)$ to transition the desire.

Note also that with this rule, the responding agent agrees to help only because it is already performing the task, but the *agree* predicate does not reveal that to the

initiating agent. This simplifies our agent design in two ways: (1) the responding agent does not have to explain to the initiating agent why it agrees to perform a requested task, and (2) the initiating agent does not have to remember why the responding agent agreed to a requested task.

Respond When a responding agent (r) believes that it can perform a requested task ρ, and there is no desire to perform ρ nor belief that it is performing ρ, it responds to the negotiation request, i.e., it agrees to negotiate.

$$\begin{aligned} & C_{in} : initiate(i, r, Do(r, \rho), t_{cin,initiate}) \wedge B : B_r(CanDo(r, \rho), t_B) \wedge \\ & \quad \neg\exists D : D_r(Do(r, \rho), t_D) \wedge \neg\exists B : B_r(Doing(r, \rho), t'_B) \\ & \Rightarrow C_{out} : respond(r, i, Do(r, \rho), t_{cout,respond}) \wedge \\ & \quad I : I_r(Negotiate(i, Do(r, \rho)), t_I) \wedge I : I_r(succeed(Negotiate(i, Do(r, \rho))), t'_I) \end{aligned}$$

where

$$\begin{aligned} & Meets(t_{cin,initiate}, t_B) \wedge During(t_B, t_D) \wedge During(t_B, t'_B) \\ & \wedge During(t_{cout,respond}, t_B) \wedge \\ & Meets(t_{cout,respond}, t_I) \wedge Meets(t_{cout,respond}, t'_I) \wedge Finishes(t'_I, t_I) \\ & \wedge Before(t_{cin,initiate}, t_{cout,respond}). \end{aligned}$$

The communicative act *respond* encapsulates the acts of composing a RESPOND message and sending the message to the initiating agent. If the responding agent believes it can perform the task, and yet it currently does not have a desire to do so, and does not believe it is performing the task, then it decides to negotiate. Note that the implicit assumptions discussed in Section 3.3.6 are at play here. Because of the cooperativeness of the agent, it intends to negotiate and intends to negotiate successfully. These two intentions motivate the responding agent to continue negotiating.

More_info When a responding agent (r) receives arguments from an initiating agent (i) for a requested task ρ, it processes the arguments to update evidence support for the task. If the support is still lacking, and the negotiation is on pace or the task is discrete, then it asks for more arguments.

$$\begin{aligned} & C_{in} : info(i, r, Do(r, \rho), t_{cin,info}) \wedge I : I_r(succeed(Negotiate(i, Do(r, \rho))), t'_I) \\ & \wedge I : I_r(update(\Gamma'_i), t''_I) \wedge \\ & \neg\exists D : D_r(Do(r, \rho), t_D) \wedge (B : B_r(\neg slow(Negotiate(i, Do(r, \rho))), t_B) \\ & \vee B : B_r(discrete(\rho), t'_B)) \\ & \qquad \Rightarrow C_{out} : more\_info(r, i, Do(r, \rho), t_{cout,more\_info}) \end{aligned}$$

where

$$\begin{aligned} & Meets(t_{cin,info}, t''_I) \wedge During(t_{cin,info}, t'_I) \wedge During(t''_I, t'_I) \\ & \wedge Meets(t''_I, t_D) \wedge During(t_D, t'_I) \\ & \wedge During(t_D, t_B) \wedge During(t_D, t'_B) \wedge During(t_{cout,more\_info}, t_D) \\ & \wedge Before(t_{cin,info}, t_{cout,more\_info}). \end{aligned}$$

Auxiliary to More_info The action *update* examines the arguments collected, Γ'_i during $t_{evidence}$ (Section 3.3.6) where $Finishes(t_{evidence}, t''_I)$, to see if the proposition $\Gamma'_i \cup \Gamma_r \models D : D_r(Do(r,\rho), t_D)$. If so, then $D : D_r(Do(r,\rho), t_D)$ where $Meets(t''_I, t_D)$.

First, the communicative act *info* encapsulates the actions of receiving and parsing an INFO-type message from the initiating agent, in which the $\langle contents \rangle$ holds the arguments $p \in \Gamma_i$ during t_{comm} and $p \notin \Gamma'_i$ during t_{comm} where $Before(t_{comm}, t_{cin,info})$. The responding agent then examines the arguments by invoking the predicate *update*. Since *update* is an action, it is self-terminating and $Equal(t''_I, [update(\Gamma'_i)])$. If the arguments are sufficient, then *update* results in $D : D_r(Do(r,\rho), t_D)$; otherwise, the negotiation continues. If at the meantime the agent believes that the pace of the negotiation is not slow or that the task is discrete, then it continues to ask for more information from the initiating agent. Note that in our protocol, a responding agent can only make a counter-offer when the requested task is non-discrete. There are two conditions that prompt a responding agent to counter-offer: (1) when the initiating agent does not have any more arguments (as discussed later), or (2) when the pace of the negotiation is slow. The predicate *slow* measures the pace of a negotiation. The predicates *update*, *slow*, and *discrete* are part of our real-time enabling functional predicates and will be discussed further in Section 3.4.

Agree 2 When a responding agent (r) receives arguments from an initiating agent (i) for a requested task ρ, it processes the arguments to update evidence support for the task. If the support is enough, then it agrees to perform ρ.

$$C_{in} : info(i, r, Do(r,\rho), t_{cin,info}) \wedge I : I_r(succeed(Negotiate(i, Do(r,\rho))), t'_I) \wedge$$
$$I : I_r(update(\Gamma'_i), t''_I) \wedge D : D_r(Do(r,\rho), t_D) \Rightarrow C_{out} : agree(r, i, Do(r,\rho), t_{cout,agree}) \wedge$$
$$I : I_i(\neg Negotiate(i, Do(r,\rho)), t'''_I) \wedge I : I_i(\neg succeed(Negotiate(i, Do(r,\rho))), t'''_I)$$
$$\wedge success$$

where

$$Meets(t_{cin,info}, t''_I) \wedge During(t_{cin,info}, t'_I) \wedge During(t''_I, t'_I) \wedge Meets(t''_I, t_D) \wedge$$
$$During(t_D, t'_I) \wedge During(t_{cout,agree}, t_D) \wedge During(t_{cout,agree}, t'''_I) \wedge Meets(t'_I, t'''_I).$$

This axiom is a counterpart of *More_info*. If it turns out that the arguments are sufficient, then the responding agent agrees to the request. It uses the communicative act *agree* to compose and send an AGREE-type message to the initiating agent. The negotiation ends with a *success* tag. Also, the agent also stops intending to negotiate and to negotiate successfully.

Counter 1 When a responding agent (r) receives arguments from an initiating agent (i) for a requested task ρ, it processes the arguments to update evidence support for the task. If the support is not enough, the pace of the negotiation is slow, and ρ involves non-discrete resource/task, then r makes a counter-offer (ρ').

$$C_{in} : info(i, r, Do(r, \rho), t_{cin,info}) \wedge I : I_r(succeed(Negotiate(i, Do(r, \rho))), t'_I) \wedge$$
$$I : I_r(update(\Gamma'_i), t''_I) \wedge \neg\exists D : D_r(Do(r, \rho), t_D) \wedge B :$$
$$B_r(slow(Negotiate(i, Do(r, \rho))), t_B) \wedge$$
$$B : B_r(\neg discrete(\rho), t'_B) \Rightarrow C_{out} : counter(r, i, Do(r, \rho), t_{cout,counter})$$

where

$$Meets(t_{cin,info}, t''_I) \wedge During(t_{cin,info}, t'_I) \wedge During(t''_I, t'_I) \wedge Meets(t''_I, t_D) \wedge$$
$$During(t_D, t'_I) \wedge Finishes(t_D, t_B) \wedge Finishes(t_D, t'_B) \wedge During(t_{cout,counter}, t_D).$$

The communicative act *counter* encapsulates the acts of finding a counter-offer (ρ'), composing a COUNTER-type message, and sending the message to the initiating agent. The counter-offer ρ' is stored in the $\langle contents \rangle$ of the message. This is a companion rule to *More_info* as discussed above. If the negotiation is off pace, then instead of asking for more information/arguments, the responding agent counter-offers. This is motivated by the intention of the agent to achieve a successful outcome to the negotiation. Note that, as mentioned earlier, when a task involves a discrete resource (see also Section 3.4.3), our protocol does not allow for a counter-offer. Thus, the *More_info* rule overwrites the *Counter 1* rule. Further, even though the responding agent makes a counter-offer, it does not have the desire to perform the task counter-offered. It only has the desire to do so after the initiating agent agrees to it. This is represented later in axiom *Success 3*.

Note that the motivation behind a counter-offer is to speed up the pace of the negotiation or as a last-ditch effort to salvage a failing negotiation. We do not perform counter-offer as part of the normal interaction – to evaluate and re-plan proposals at each negotiation step would have slowed down our negotiations and that is not applicable to a real-time problem.

Stop When a responding agent (r) is notified by an initiating agent (i) that it has no more arguments for a requested task ρ, and the task is discrete, the agent r stops the negotiation, and the negotiation fails.

$$C_{in} : info\_null(i, r, Do(r, \rho), t_{cin,info\_null}) \wedge I : I_r(succeed(Negotiate(i, Do(r, \rho))), t'_I) \wedge$$
$$I : I_r(update(\Gamma'_i(Do(r, \rho))), t''_I) \wedge \neg\exists D : D_r(Do(r, \rho), t_D) \wedge B : B_r(discrete(\rho), t'_B)$$
$$\Rightarrow C_{out} : stop(r, i, Do(r, \rho), t_{cout,stop}) \wedge I : I_r(\neg Negotiate(i, Do(r, \rho)), t'''_I) \wedge$$
$$I : I_r(\neg succeed(Negotiate(i, Do(r, \rho))), t'''_I) \wedge stop$$

where

$$Meets(t_{cin,info\_null}, t''_I) \wedge During(t_{cin,info\_null}, t'_I) \wedge During(t''_I, t'_I)$$
$$\wedge Meets(t''_I, t_D) \wedge During(t_D, t'_I) \wedge$$
$$Finishes(t_D, t_B) \wedge Finishes(t_D, t'_B) \wedge During(t_{cout,stop}, t_D) \wedge Meets(t'_I, t'''_I)$$
$$\wedge Starts(t_{cout,stop}, t'''_I).$$

The communicative act *info_null* is the encapsulation of receiving and parsing an INFO_NULL-type message from the initiating agent, while the communicative act *stop* encapsulates the actions of composing a STOP-type message and sending the message to the initiating agent. This is when the responding agent gives up on the negotiation, after being informed that no more arguments are on the way. As a result, it no longer intends to negotiate, and it opts out by informing the initiating agent as a responsible gesture.

The proposition *stop* indicates the failure of a negotiation because the responding agent is not convinced to perform the requested task.

Counter 2 When a responding agent (r) is notified by an initiating agent (i) that it has no more arguments for a requested task ρ, and the task is discrete, the agent r makes a counter-offer (ρ').

$$C_{in} : info\_null(i, r, Do(r, \rho), t_{cin,info\_null}) \wedge I : I_r(succeed(Negotiate(i, Do(r, \rho))), t'_I) \wedge$$
$$I : I_r(update(\Gamma'_i), t''_I) \wedge \neg\exists D : D_r(Do(r, \rho), t_D) \wedge B : B_r(\neg discrete(\rho), t'_B)$$
$$\Rightarrow C_{out} : counter(r, i, Do(r, \rho), t_{cout,counter})$$

where

$$Meets(t_{cin,info}, t''_I) \wedge During(t_{cin,info}, t'_I) \wedge During(t''_I, t'_I) \wedge Meets(t''_I, t_D) \wedge$$
$$During(t_D, t'_I) \wedge Finishes(t_D, t_B) \wedge Finishes(t_D, t'_B) \wedge During(t_{cout,counter}, t_D).$$

This axiom is a counterpart of *Stop*, and closely resembles *Counter 1*. Driven by the intention to succeed in the negotiation, the responding agent, after being notified of no more arguments coming in from the initiating agent, voluntarily makes a counter-offer if the task is non-discrete.

Failure 3R This axiom is similar to *Failure 3I*. See Appendix A.

Failure 4R This axiom is similar to *Failure 4I*. See Appendix A.

Abort R This axiom is similar to *Abort I*. See Appendix A.

Out_of_time R This axiom is similar to *Out_of_time I*. See Appendix A.

Success 3 When a responding agent (r) receives an AGREE message from an initiating agent (i) to its counter-offer (ρ'), the responding agent desires to perform the counter-offer, and the negotiation ends with success.

$$C_{in} : agree(i, r, Do(r, \rho), t_{cin,agree}) \wedge I : I_r(succeed(Negotiate(i, Do(r, \rho))), t'_I)$$
$$\Rightarrow D : D_r(Do(r, \rho'), t_D) \wedge I : I_i(\neg Negotiate(i, Do(r, \rho)), t''_I) \wedge$$
$$I : I_i(\neg succeed(Negotiate(i, Do(r, \rho))), t''_I) \wedge success$$

where $Finishes(t_{cin,agree}, t'_I) \wedge Meets(t'_I, t_D) \wedge Meets(t'_I, t''_I)$. The communicative act *agree* predicate encapsulates the acts of receiving and parsing an AGREE-type message, in which $\langle contents \rangle$ holds the information regarding ρ'. This rule says that if

the initiating agent agrees to the counter-offer, then the responding agent (1) desires to perform the counter-offered task and (2) intends no longer to continue with the negotiation regarding the originally requested task.

No_response R This is similar to *No_response I*. See Appendix A.

Figure 5 shows the time lines of the initiating agent's behavior when faced with an outright rejection or agreement. These are the simple cases of the axioms above. See for example that when the responding agent agrees to a request, it extends the desire to do the requested task.

Figure 6 shows the negotiation process, from the responding agent's point of view once it agrees to negotiate. Similar to Figure 4, Figure 6 allows us to explicitly describe the temporal relationships among the BDI states and the communicative acts. That description, in turn, allows us to guarantee the behavior of the negotiation and to fine-tune its efficiency. See Section 3.3.7. Note that when a responding agent has a desire to perform the requested task, it will monitor its negotiation progress, as triggered by the responding agent's intention to update its belief of whether the task requested should be agreed to. Also, when the task is non-discrete and the progress is slow, the responding agent will make a counter-offer.

3.4. Functional predicates

In this section, we describe the functional predicates mentioned in the previous section that present the logical framework for the rules of encounter between two agents. These predicates are the infrastructure to our real-time negotiation protocol. To simplify the discussion, we have touched upon 11 communicative acts such as *initiate*, *respond*, *no_go*, *agree*, etc. (discussed in Section 3.3) and six negotiation-related functional predicates: *slow*, *time, discrete, no_response*, *acceptable*, and *update*. Here we will elaborate further on the six predicates as they are an integral part that enables the real-time negotiation between the agents.

3.4.1. Slow This predicate takes the form of *slow*(*action*), and given an action (or a task), it measures the pace of the action and returns true or false. An action has two temporal intervals: the actual real-time interval, [*action*], and the planned/predicted interval, ‖*action*‖. Suppose that [*action*] has a duration between $t_{s,action}$ and $t_{f,action}$, and the set of states as a result of the action is $S_{action} = \{s_{0,action}, \ldots, s_{N,action}\}$.

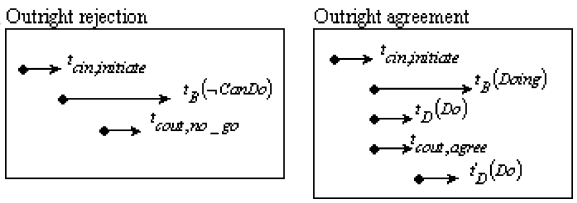

Figure 5. Time lines of the responding agent's behavior when faced with an outright rejection or agreement.

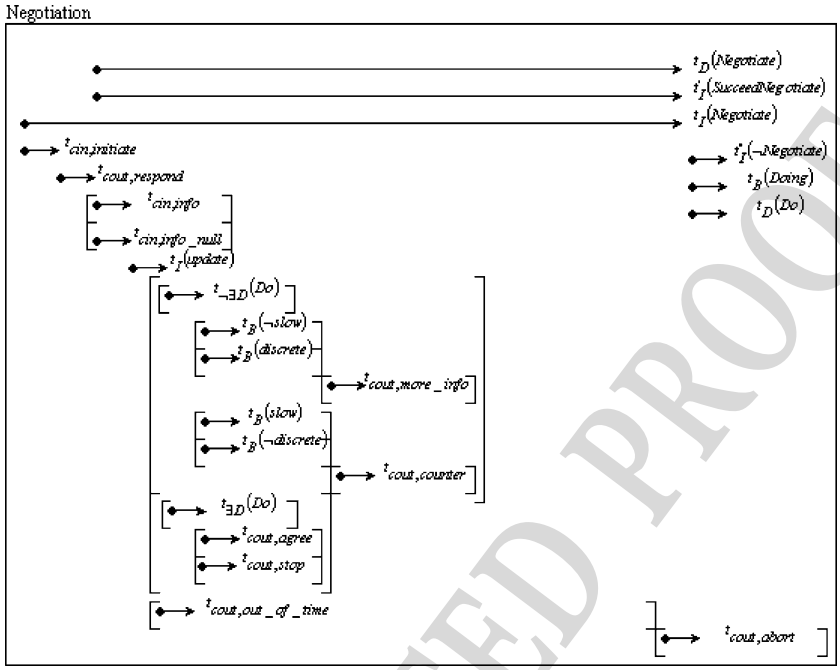

Figure 6. Time lines of the negotiation from the responding agent's point of view (a simplified version). Temporal intervals under brackets are options. Note that once the responding agent sends out a RESPOND message to the initiating agent, it is committed to negotiate successfully.

Each state, $s_{n,action}$, holds during an actual interval $[s_{n,action}]$ such that (1) $During([s_{n,action}],[action])$, (2) $Overlaps([s_{n,action}],[s_{n+1,action}])$, and (3) the temporal interval of two such states has a duration between the start of $s_{n,action}$ and the latest finish time of the two states, $[s_{n,action}\ s_{n+1,action}]$. Similarly, we can obtain $\|s_{n,action}\ s_{n+1,action}\|$

When $slow(action)$ is invoked, it retrieves the current state of the action, $s_{current,action} \in S_{action}$. If

$$\frac{[s_{0,action}\ s_{current,action}]}{[action]} > \frac{\|s_{0,action}\ s_{current,action}\|}{\|action\|}$$

is true, then $slow(action)$ returns true; otherwise, it returns false. Note that this predicate is binary since we use it to trigger a *counter_offer* act. A more general approach is to use a *degree* of slowness that would not only trigger a *counter_offer* but also dictate how conceding the responding agent should be.

3.4.2. Time This predicate takes the form of $time(action)$ and indicates whether an agent has run out of the allocated time for the *action*. Borrowing the notations from Section 3.4.1, suppose we have $[action]$ and $\|action\|$. When an agent invokes $time(action)$, the predicate measures the time elapsed so far, $t_{elapsed} = t_{current} - t_{s,action}$. If $t_{elapsed} \geq \|action\|$, then $time(action)$ returns true; otherwise it returns false.

3.4.3. Discrete A discrete task is when the task cannot be broken up or attenuated. For example, if the initiating agent asks the responding agent to turn on a sensor, then the responding agent may only respond with yes or no. However, if the initiating agent asks the responding agent to turn on a sensor in five seconds, then the responding agent, in addition to yes or no, may also counter with "yes, but in 10 seconds". The introduction of the time factor makes the task a non-discrete one, allowing the responding agent to counter-offer. In our negotiation protocol, a task ρ may be qualified by time and resource amount. The predicate *discrete* returns true if both qualities are absent, otherwise, it returns false. Note that we intentionally leave out the qualification of tasks in Section 3.3 to simplify our discussions.

3.4.4. No_response This predicate takes one argument – the agent from which the current agent is waiting for a message. In general, when an agent i invokes *no_response*(r) on another agent r, it is after agent i has performed a communicative act, C. Hence, $Before([C], [no\_response(r)])$. If agent i does not receive a response in the interval t_{window} (and $During(t_{window}, t_I)$ given that the intention I is to negotiate), then *no_response*(r) returns true; otherwise, it returns false.

3.4.5. Acceptable This predicate is only invoked by an initiating agent, I, and takes as argument a task. Suppose the agent i desires to achieve goal G. To achieve G, there is a set of subtasks $\vec{\rho} = \left\{\rho_1, \rho_2, \ldots, \rho_{|C_i(\vec{\rho})|}\right\}$, and as mentioned in Section 3.3.5, a coalition, $C_i(\vec{\rho})$, exists to distribute the subtasks among the coalition members. As a result,

$$D_i(G, t_D) \Rightarrow D_i(Do(C_i(\vec{\rho}), \vec{\rho}), t_D),$$

where $D_i(Do(C_i(\vec{\rho}), \vec{\rho}), t_D) = \left\{D_i(Do(r_1, \rho_1), t_{D1}), \ldots, D_i\left(Do\left(r_{|C_i(\vec{\rho})|}, \rho_{|C_i(\vec{\rho})|}\right), t_{D|C_i(\vec{\rho})|}\right)\right\}$. As the agent progresses in real time, G may become G' such that $D_i(G', t'_D) \Rightarrow D_i\left(Do\left(C_i(\vec{\rho}'), \vec{\rho}'\right), t'_D\right)$ where $During(t'_D, t_D)$. For example, an agent has received commitments for some negotiated resources, so it no longer desires the original set of resources if asked for. So, when the initiating agent receives a counter offer ρ' from the responding agent, it invokes *acceptable*(ρ') to compare the counter-offered task ρ' with the corresponding ρ'_r where $D_i\left(Do\left(r, \rho'_r\right), t'_D\right)$. If $\rho' \in \rho'_r$ then *acceptable*(ρ') returns true; otherwise, it returns false.

Whether a resource request is agreeable by a responding agent and whether a counter-offer is acceptable by an initiating agent depend on how the request is determined in the first place by the initiating agent. And this determination of which and the amount of resources to request from which responding agents is task allocation in our coalition. Note that in [50], we detail how task allocation is carried out in our coalition. Briefly, the initiating agent makes use of the potential utilities to carry out task allocations and assignments. Based on the overall potential utility of the initial coalition, the initiator may want to lower its demands to improve the chance of forming a coalition. By the same token, if the potential utility of a candidate is high, then the initiator may want to increase its demand with that candidate. We are currently investigating various algorithms such as greedy, lazy, worried, and weary. An initiating agent becomes greedy during a negotiation when (1) it tries to minimize its own rationalization and computing process, (2) it selects the

candidate with the higher overall utility values to approach hoping for a successful negotiation, (3) it cares mostly about high-priority tasks, (4) it tries to maximize its chance of getting a particular task done – by including sub-utilities in the focused utility evaluation, and (5) it hopes to shift its responsibility (partially) to the candidates via successful negotiations – expecting the candidates to spawn their own coalitions to help respond to the problem at hand. In a lazy algorithm, the initiator prefers to concentrate its effort on a few candidates, as it does not want to spend resources or time on too many. In a worried algorithm, the agent asks for more than it needs to ensure, that if some of the candidates refuse to help, it will get what it needs. This translates to insurance policies. Finally, in a weary algorithm, the initiator prefers not to upset a candidate – especially one that has a high uniqueness – by being over-demanding. This leads to demand caps that make sure that an additional demand does not hurt the negotiation.

3.4.6. Update This functional predicate is invoked only by a responding agent *r*, as previously mentioned in an auxiliary to the *More_info* axiom. See the discussion in Section 3.3.7.

4. Implementation

The driving application for our system is multi-sensor target tracking, a distributed resource allocation and constraint satisfaction problem. The objective is to track as many targets as possible and as accurately as possible using a network of sensors. Each sensor has a set of consumable resources, such as beam-seconds (the amount of time a sensor is active), battery power, and communication channels, that each sensor desires to utilize efficiently. Each sensor is at a fixed physical location and, as a target passes through its coverage area, it has to collaborate with neighboring sensors to triangulate their measurements to obtain an accurate estimate of the position and velocity of the target. As more targets appear in the environment, the sensors need to decide which ones to track, when to track them, and when not to track them, always being aware of the status and usage of sensor resources. Each sensor can at any time scan one of three sectors, each covering a 120-degree swath. Sensors are connected to a network of CPU platforms on which the agents controlling each sensor reside. The physical sensors are 9.35 GHz Doppler MTI radars that communicate using a 900 MHz wireless, RF transmitter. The agents (and sensors) must communicate over an eight-channel RF link, leading to potential channel jamming and lost messages. Our agents may reside on the same CPU platform or different platforms.

We are using a tracker module that receives radar measurements and returns to the agent the estimated position and velocity of a target. These estimates are improved with more measurements from different sensors within a short time interval, since this allows better target triangulation. The tracker software is not an agent, and simply implements a target tracking algorithm. The communication between the tracker module and an agents is via socket connections.

We have implemented the real-time, case-based negotiating agents described in the previous sections and tested them using real sensors and targets moving in a physical environment. The agents exhibit all of the behavior described: they use CBR to select and adapt a negotiation strategy, use a RTSS to request CPU resources and to have time and system awareness, negotiate for radar use, and learn the new negotiation strategies they have developed. See [50] for more details.

4.1. Agents and environments

We have implemented our multi-agent system in the C++ programming language. Each agent has $3+n$ threads: (1) a core thread that performs the reasoning, message checking, task handling chores of the agent and thus is always active, (2) a communication thread that is responsible for polling for incoming messages and sending out messages and thus is always active, (3) an execution thread that performs sensor-related tasks such as calibration, target searching, and tracking, and thus is sometimes active and sometimes dormant, and (4) n negotiation threads that each waits to be awaken to perform a negotiation and goes back to a dormant state after the negotiation is over. This setup allows an agent to carry out various lines of tasks concurrently. It also allows an agent to conduct multiple negotiations in parallel. Each negotiation thread can be an initiating thread or a responding thread, depending on the dynamic, real-time instructions given by the core thread.

The current implementation of our agents is able to (1) detect a target, (2) form a coalition, perform CBR to determine its negotiation strategy, initiate or respond to negotiations, (3) argue to persuade its partner to perform a task or reasons to whether to agree to perform a task, (4) monitor its own status such as its sensor, noise, tasks, and CPU resource usage, (5) interact with either a software simulation or the actual physical hardware setup, and (6) obtain real-time data from the operating system supporting its execution. More importantly, each agent is autonomous and can sense and react to real-time events in the environment. Moreover, there is no hierarchy within the multi-agent system – all agents are peers.

We have also implemented the complete real-time argumentative negotiation protocol in the negotiator module of an agent. With the formalisms encoded, the negotiator conducts a negotiation with high efficiency and autonomy. We have implemented all communicative acts: parsing messages and converting them to belief states and converting belief states and composing messages out of them. We have also implemented the belief, desire, and intention states as functions, procedures, and clauses. As for the temporal definitions and constraints of those states, we have implemented recursive mutexes (semaphore-like designs) to manage read/write accesses: when to acquire the value of a state, when to release a lock on the value of a state, when must a state be ready in order for a certain task to start, and so on. For example, an agent may be tracking a target and negotiating at the same time. While tracking, the agent may realize that the target is no longer visible. This directly affects the on-going negotiation since now the agent's sensor has become available, leading to a lower threshold, for example, of a counter-offer. But, the negotiator module of a negotiation thread cannot afford to constantly check the states of the

tracking. It does so occasionally and only when it is necessary, and can only be interrupted at certain points over the course of the negotiation. This is dictated by the temporal definitions of the states found in Section 3.

4.2. Real-time scheduling service

We have implemented a RTSS in the C programming language, on top of the KU Real-Time system (KURT) [53] that adds real-time functionality to Linux. First, the RTSS provides an interface between the agents and the system timers, allowing agents to (1) query the operating system about the current time; (2) ask the RTSS to notify them after the passage of certain length of time; and (3) ask the RTSS to ping them at fixed time intervals. This allows agents to know when to, for example, conclude a negotiation process or turn on a radar sector. Second, the agents may ask the RTSS to notify them when certain system-level events occur, such as process threads being activated, or communication messages going out or coming into the system. Third, the agents can ask the RTSS to allocate them a percentage of the CPU for each one of their threads (such as the ones controlling the radar and tracking or the ones used in negotiations) and to schedule this allocation within an interval of time. This RTSS allows an agent to monitor the progress of its own negotiations and the usage status of its allocated CPU resource.

4.3. Case-based argumentative negotiation

We have implemented the CBR Manager to maintain the case base of an agent. The implementation includes similarity-based retrieval, both difference- and outcome-driven adaptations, and the incremental and refinement learning. The CBR Manager retrieves a case for each negotiation. A case provides the appropriate negotiation strategy for the agent. For a responding agent, the negotiation strategy also includes the persuasion threshold. When the responding agent receives an argument from the initiating agent supporting its request, it measures the support of this argument using CLIPS[8] rules and compares the support collected so far against the persuasion threshold. We have implemented the entire negotiation protocol, as depicted in Figure 1, into each negotiation thread (the number of threads in an agent is static for a specific agent implementation. In our experiments we used two negotiation threads per agent). Each thread is capable of monitoring the pace of its own negotiation, retrieving messages via the communication thread, parsing incoming messages, making decisions and reasoning, composing an outgoing message and sending messages via the communication thread. Each thread is autonomous in a way that the core thread of the agent does not have to tell the thread how to conduct a negotiation once it has gotten underway.

Each thread forks off a child process that automatically invokes CLIPS. The communication between a negotiation thread and its child CLIPS process is through pipes. After receiving an acknowledge signal from the CLIPS child process, the negotiation thread informs the core thread that it is ready to accept a negotiation task and waits. When it finally receives an activation signal, the negotiation thread downloads the

relevant information regarding the negotiation task. From the information, the thread decides its identity – either an initiating thread or a responding thread. Then the negotiation thread negotiates following the negotiation protocol described in Section 3. Once the negotiation is done, the thread updates its status and waits for a signal from the core thread before resetting itself for the next negotiation task. Meanwhile, the core thread of the agent periodically checks the status of the active negotiation threads. If a negotiation is completed, the core thread downloads the updated data and signals the negotiation thread that it is okay to reset.

In our case-based strategy selection approach, a negotiation strategy dictates a set of tactics for a negotiation. For example, an initiating agent needs to know which arguments are more important to send over first to the responding agent. We use CBR to help us determine that. When an agent encounters a negotiation problem, it searches its casebase for the most similar case, in which the problem description in that case resembles the current negotiation problem. Then, based on the differences between the two problem descriptions, the CBR module of the agent performs an adaptation on the solution. The modified solution becomes the negotiation strategy (details are provided in [51]).

4.4. Real-time enabling functional predicates

In Section 3.4, we presented the logical model of our real-time enabling functional predicates. Here, we describe the implementation that, even though is domain- and application-specific, may serve as a useful example to other designs of the predicates. In this section, we also describe how our agents make a counter-offer in real-time. Note that in our implementation, we employ CBR to derive a negotiation strategy for an agent for each negotiation task. A case has a set of belief states (situated input parameters), a set of desires (a parametric negotiation strategy), and the outcome of the negotiation. In addition, in our agent design, a negotiation is handled by one of the negotiation threads that an agent dispatches. So, in the following, we will use the term "negotiation thread" quite often and make use of the belief and desire states.

4.4.1. Slow In a case, the desires include the number of negotiation steps allowed and the time allowed. That is, a negotiation thread desires to complete a particular negotiation in n iterations and s seconds. A small n means that fewer messages are exchanged and the negotiation may avoid incurring too much overhead cost per transmission. A small s means that the negotiation is to be completed in a short time.

Suppose we denote the number of negotiation steps allowed as $step_{allowed}$, the time allowed as $time_{allowed}$, the number of steps performed so far as $step_{sofar}$, and the time elapsed so far as $time_{sofar}$. We define the *slow* predicate for a responding agent α's negotiation (intending to achieve the desire for performing a requested task ρ) as

$$slow(I_\alpha(D_\alpha(Do(\alpha, \rho))), t) = \left(\frac{time_{sofar}}{step_{sofar}} > \frac{time_{allowed}}{step_{allowed}}\right)$$

This definition is an example of the logical model discussed in Section 3.4.1.

4.4.2. Time To implement this predicate, a negotiation thread registers its process ID with a real-time system-level service and makes use of a time-based notification mechanism. A negotiation asks the notification mechanism to signal the thread after s seconds. One unique characteristic of the mechanism is that the negotiation thread, after registration, may find out how much of the s seconds has elapsed after the notification was first registered (for example, 25, 50, 75, 100%) by consulting the notification flag: t_{flag}. The value of s is determined by the $time_{allowed}$ of the desire states of a case.

In our current design, we define the *time* predicate for an agent α's negotiation as

$$time(negotiation) = \left(t_{flag} < 1\right)$$

When $t_{flag} = 1$, that means the time elapsed has reached 100% of $time_{allowed}$.

4.4.3. Discrete First, we denote the set of discrete tasks Θ_{dis} and the set of non-discrete tasks Θ_{con}. Then we define the *discrete* predicate of a requested task ρ as

$$discrete(\rho) = \left(\rho_{request} \in \Theta_{dis}\right)$$

where $\rho_{request} \in \rho$ is part of the requested task.

4.4.4. No_response Our implementation is the following: After an agent sends out a message, it polls its message queue for a response before moving on. After $t_{polling}$ seconds, if the negotiation thread receives no messages from a particular negotiation partner, then *no_response* returns true. If there is a consistently typed message, then the negotiation thread reacts to it based on the negotiation protocol outlined in Section 3.3.

4.4.5. Acceptable This predicate is used only by an initiating agent when it receives a counter-offer from a responding agent and refers to non-discrete (continuous) tasks, θ_{con}. Let us denote a continuous task as $\vec{r} \in R_{con}$. There are three key parameters in $\rho_{con} : \rho_{con} = \{\rho_{con,name}, \rho_{con,res}, \rho_{con,amount}\}$ where $\rho_{con,name}$ is the name of the task, $\rho_{con,res}$ designates the resource involved in the task, and $\rho_{con,amount}$ indicates the amount of the resource involved.

In our design, when an agent α realizes $B : B_\alpha\left(\rho^{needed}_{con,amount}\right)$ where $\rho^{needed}_{con,amount}$ is non-zero, it initiates negotiations – each with a different $\rho^{requested}_{con,amount}$, to the coalition members – attempting to obtain enough $\rho_{con,amount}$ from the members to meet $\rho^{needed}_{con,amount}$, i.e., $\sum_{C_\alpha(\vec{\rho})} \rho^{requested}_{con,amount} \geq \rho^{needed}_{con,amount}$. At each agent cycle, the agent α updates its $B : B_\alpha\left(\rho^{needed}_{amount}\right)$ and $B : B_\alpha\left(\sum_{C_\alpha(\vec{\rho})} \rho^{requested}_{con,amount}\right)$. If $B : B_\alpha\left(\rho^{needed}_{con,amount} = 0\right)$ then it has achieved its target, and it can abort all current negotiations associated with that particular resource. The process checks (1) the current usage, (2) the anticipated usage, (3) the current allocation, and (4) the agreed additional allocation. As $\rho^{needed}_{con,amount}$ gets smaller, the remaining negotiations become less demanding in their requests, and vice versa. As each negotiation completes gradually, $\sum_{C_\alpha(\vec{\rho})} \rho^{requested}_{con,amount}$ changes.

Suppose that the kth negotiation thread of the agent α is negotiating to obtain $\rho^{requested}_{con,amount,k}$ from a responding agent and the responding agent has just counter-offered ρ'_k with an offered amount of $\rho'_{con,amount,k}$. Then the acceptability of the counter-offer ρ'_k is defined as

$$acceptable(\rho'_k) = \left(\sum_{C_\alpha(\vec{\rho})} \rho^{requested}_{con,amount} - \rho^{requested}_{con,amount,k} + \rho'_{con,amount,k} \geq \rho^{needed}_{con,amount} \right)$$

4.4.6. Update The *update* predicate is used when a responding agent receives information or arguments from an initiating agent. The objective of this predicate is to find out whether the evidence support for a requested task is convincing enough for the responding agent to perform it. One key parameter of a negotiation strategy of a responding agent, r, is the persuasion threshold, $T_{persuasion,Do(r,\rho)}$, for a requested task ρ. This is a value created by the responding agent r for ρ; to agree to $Do(r,\rho)$, the arguments sent by the initiating agent must provide for ρ that is greater than $T_{persuasion,\rho}$.

Suppose we denote the evidence support for $Do(r,\rho)$, with a persuasion threshold, $T_{persuasion,Do(r,\rho)}$, at temporal interval t as $Support(\rho, T_{persuasion,Do(r,\rho)}, t)$, and $Support(\rho, T_{persuasion,Do(r,\rho)}, time0) = 0$. The objective of the initiating agent is to obtain $Support(\rho, T_{persuasion,Do(r,\rho)}, t) \geq T_{persuasion,Do(r,\rho)}$ in order to convince the responding agent to perform the requested task. Thus we have the following axiom:

Axiom desire to do If a responding agent r is negotiating with an initiating agent i regarding a requested task ρ, and at temporal interval t, it has $Support(\rho, T_{persuasion,Do(r,\rho)}, t) \geq T_{persuasion,Do(r,\rho)}$ then $D : D_r(Do(r,\rho), t)$. (This supplements the *More_info*, *Agree 2*, *Counter 1*, *Stop*, and *Counter 2* axioms discussed in Section 3.3.)

When arguments are received by the responding agent, the support value changes based on the following definitions of an agent behavioral model:

(1) Agents that have cooperated before will tend to cooperate again.
(2) A responding agent is willing to trust an initiating agent's perception.
(3) An agent is more inclined to help another agent if that another agent has relied on the agent for help before.
(4) An agent is more inclined to help another agent if it knows that it is one of the few possible solutions to the requested task.
(5) An agent is more inclined to help another agent if it knows that that another agent is busy.

Note that the persuasion threshold and the evidence support work together to *imitate a joint intention*[9] between the two negotiating agents. On one hand, the responding agent is helpful and desires to help the initiating agent, but it also intends to help when it is worthwhile. This implies cooperativeness with a touch of selfishness in a local sense that translates into global optimization of resource allocation. To make sure that the requested task is worthwhile to do, the responding agent uses a persuasion threshold, derived from its past experience and its current status. On the other hand, the initiating agent intends that the responding agent help with its task.

It collects its belief states and sends over whatever it thinks are useful as arguments for its intention. These arguments modify an evidence support value. Therefore, Definition 4 (''An agent is more inclined to help another agent if that another agent has relied on the agent for help before.'') merges the two intentions, seen from two different perspectives, and the joint intention is to achieve a successful negotiation. This deviates from the model proposed by Cohen and Levesque [9, 10] but if we view *achieving a successful negotiation* as a *team action*, then both members of the team (the two negotiating agents) are jointly committed to completing the so-called team action, with the agent that performs the task believing that it performs the task while the other agent believes that the same performs the task as well.

We have implemented the *update* function as a CLIPS-based operation. Evaluation heuristics are coded as CLIPS rules and arguments received from an initiating agent are fed into the CLIPS process (associated with each negotiation thread) to recompute the evidence support for a requested task. The CLIPS process then sends back the updated evidence support to its negotiation thread and waits for another set of arguments. The negotiation thread and the CLIPS process communicate via a synchronized pipe connection. The negotiation thread compares the updated evidence support with its persuasion threshold. If the former is greater than or equal to the latter, then the negotiation thread agrees to the requested task, composes a message to notify the initiating agent of the deal, and completes its own negotiation. The core thread of the agent then schedules the requested task in its activity.

Here we list one example of the CLIPS rules used. The first rule indicates that if the initiating agent has been helping the responding agent in the past with a rate of 66.7%, then the evidence support is incremented by 0.05 of that rate in favor of the initiating agent with its current request.

```
(defruleworld-help-rate
(world(_helpRate?y&:(>?y0.6667)))
=>
(bind?*evidenceSupport*(+(*0.05?y)?*evidenceSupport*))
(bind?*world-help-rate-count*(+?*world-help-rate-count*1)))
```

In all, we have 15–20 rules in our CLIPS rulebase.

4.4.7. Counter offer When dealing with continuous resources, the responding agent has a persuasion function, modified by two parameters: (1) *kappa* – a willingness factor, and (2) *beta* – a conceding factor, and bounded by the maximum resource that it is willing to give up, $\rho^{max}_{con,amount}$. An agent can use any function[10] to express the continuous persuasion value; in our implementation, we examined two: a linear and an exponential persuasion function.

In our model, the linear persuasion function is:

$$\mathrm{P}_{linear}(\tau) = beta \cdot \tau + kappa$$

and, the exponential persuasion function is:

$$\mathrm{P}_{exp}(\tau) = kappa \cdot e^{\tau/beta}$$

The variable τ is the evidence support collected so far, i.e., $\tau = Support\left(\rho, T_{persuasion,Do(r,\rho)}, t\right) \geq T_{persuasian,Do(r,\rho)}$. So, in the beginning, at t_s when the support is zero, a responding agent is willing to give up $P_{linear}(0) = kappa$ or $P_{exp}(0) = kappa$. That is why *kappa* is called the willingness factor. Then for $\tau > 0$, the conceding rate depends on *beta*: the larger this value is, the more conceding the persuasion function is. The key differences between our persuasion functions and others are (1) the conceding and willingness factors are determined by past experiences, and adapted to fit the current situation, (2) each function applies to an evidence support based on arguments, (3) each function is implicitly bounded by $time_{allowed}$ for a negotiation, and (4) each function is bounded by $\rho^{max}_{con,amount}$, the maximum amount of a resource that the agent is willing to give up.

Thus, when a responding agent is about to make a counter-offer, it checks τ, and the counter-offer, $\rho'_{con,amount}$ is given as

$$\rho'_{con,amount} = \begin{cases} P(\tau) \\ \rho^{max}_{con,amount} \text{ if } P(\tau) > \rho^{max}_{con,amount} \end{cases}$$

5. Experiments and results We conducted a set of experiments with a varying number of real sensors and vehicles (as described in Section 4). The most complex experiment involved eight 9.35 GHz Doppler MTI radars and two targets moving in oval trajectories (Figure 7). During the experiments we used three different types of

COLOUR FIG.

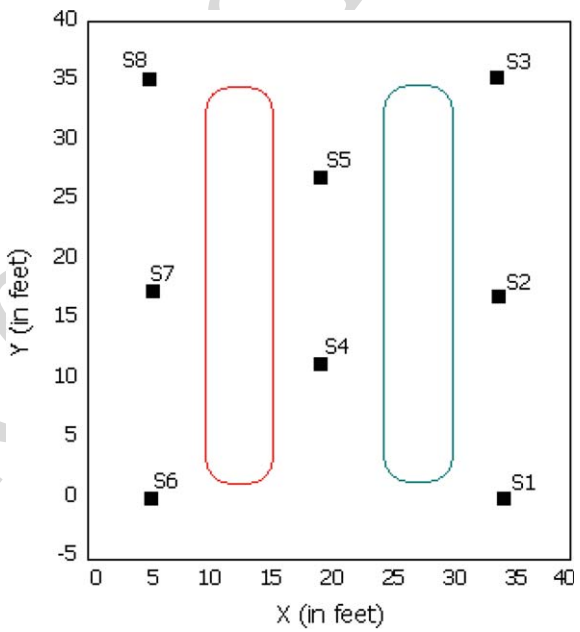

Figure 7. Experimental setup. The two ovals represent the trajectory of the two targets. The sensors are labeled as S1 to S8. The *X* and *Y*-axes are the dimensions of the area in which the targets travel and the sensors are located.

agents so as to compare the behavior and performance of other agent formalisms with that of the performance of negotiating, real-time agents described in this paper. Our hypothesis was that negotiating agents can track targets better since they can coordinate radar measurements and achieve better triangulation. The MTI sensors and the vehicles were simulated by the Radsim simulator version 2.08 [32]. Agents connect to Radsim and use a standard API to send control signals to their respective sensor platforms. Radsim, in turn, uses a model of the platforms' behavior to return, in the case of taking a measurement, a hypothetical value of amplitude and frequency based on the current locations of the targets. For each setup, we had three runs. Each run was run for 250,000 simulation ticks, representing 250 real-time seconds.

Our negotiating agents were compared to (1) simple, reactive ones, and (2) negotiating agents that use a single, static negotiation strategy instead of an adaptive, case-based one. The reactive agents operated as follows: When an agent detected a target, it started tracking it without attempting to form a coalition. Comparing our negotiating agents to reactive ones allowed us to establish a baseline performance in which the agents simply track without coordinating with other agents (as in the simple, reactive scenario) and study how coordinated tracking in a coalition (as in our negotiating agents) helped improve the tracking. When the agent no longer sensed the target, it returned to a "wait" state. The single-strategy negotiating agents used our negotiation protocol, but always used the same negotiation strategy. Comparing our negotiating agents between using a single, static strategy and using an adaptive, case-based one allowed us to investigate how agents equipped with learning capability are able to improve their negotiation and ultimately target-tracking efforts.

We used two metrics of quality for the performance of the three techniques. First, we used the number of times that an agent selected the correct sensor sector to turn on, indicating that it was correctly tracking the target. A higher percentage of correct sector selections would also lead to better triangulation and tracking, implying better collaboration among the agents. Second, we used the *quality* of the measurements made. A target is visible by a sensor sector for a period of time. During this time, its visibility to the sensor changes as the target moves closer or further away from the sensor. The *exact time* at which a sensor makes a measurement establishes the *quality of the measurement as a function of the target visibility*. In other words, the first metric indicates whether the agent selects the correct sector to turn on at the write time, and the second metric indicates whether the appropriate sensor is tracking the target.

To establish the optimal (or correct) sector that an agent should be looking at, we used the omniscient agent experiment performed by researchers at the University of South Carolina (http://www.cse.sc.edu/~vargasje/targetshare/). In their experiment the South Carolina researchers built a controller that knew the real location of the targets most of the time. The controller used that information to command a centralized agent to change the sectors of the sensors so that they pointed most of the time in the direction of the best sector for tracking the target. Their experiment determined the best baseline performance that a centralized, omniscient agent could achieve and provided us with the best "ground truth" available sector. Further, the negotiating agents with CBR did significantly better in tracking Target 1 than in

tracking Target 2. This is possibly due to resource contention for the sensors S4 and S5, where Target 1 was first engaged and acquired by the sensors, which caused Target 2 to be "off" in terms of the sensors' timing.

The results of this evaluation are summarized in the Table 1 below.

Our real-time negotiation protocol with CBR does much better in creating coalitions of the appropriate sectors to track a vehicle, especially for the left target.

Next, we used the *quality* of the measurements made as a proxy of the quality of the coalition formation and the task execution. The *exact time* at which a sensor makes a measurement establishes the *quality of the measurement as a function of the target visibility*. An example is given in Figure 8: The red arc represents the visibility of a target by a sector of a sensor (sensor named "node 1" and sector 1). The x-axis is time in milliseconds. The y-axis is the quality represented by the sensitivity of the target position to measurement errors. In practice average measurement errors are on the order of between 5 and 10 feet, so the vertical range of the chart roughly corresponds to the range of sensitivity over which a measurement has some theoretical usefulness for tracking purposes. Notice that for any given sensor most of the time measurements are of negligible value (for tracking); however, at any given time there exists a sensor which would produce valuable measurements. Also, the smaller the value of *quality*, the better, since it represents the expected average error. The vertical green lines are the times when the sector was measuring. Clearly, we want the measurements to coincide with the existence of a target (although some measurements will not, since the radar will occasionally be on sentry duty, searching for targets). In Figure 8, sector 1 of node 1 has made multiple measurements when the target was first visible starting around time 180, but then did not turn itself off and failed to start measuring the target that appeared around time 305. For *quality* of measurements we reward an agent for taking high-quality measurements of a target, and penalize it for measuring when there is no target.

In our experiments, different sensors had different quality totals based on their location relative to the target trajectories (for example, sensor 2 had the best performance because it could dedicate all its resources to one target only; sensors that had to track both targets had worse performance). Our results are presented both per sensor and as an average. The results of the quality of measurement are summarized in Table 2.

Our real-time negotiation protocol with CBR does much better in creating coalitions of the appropriate sectors to track a target, especially for the left target.

Table 1. Percentage of correctly selecting a radar sector for tracking a target for the three agent designs.

Agent designs	Correctly selecting radar sector for Target 1 (left target)	Correctly selecting radar sector for Target 2 (right target)
Negotiating agents with CBR	88.60%	75.69%
Negotiating agents with single strategy	68.20%	74.50%
Reactive agents	69.95%	70.72%

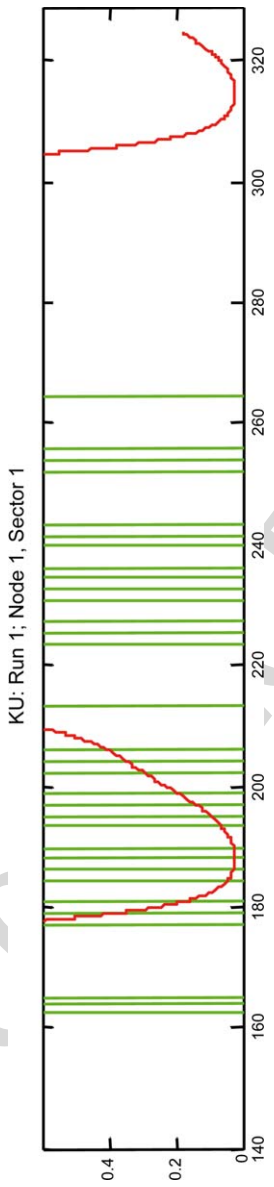

Figure 8. A graph explaining the concept of "quality of measurement" as a function of the visibility of a target and the time a measurement is taken. Target trajectories are represented by the arcs and the sensor measurements are represented by the vertical lines. The *x*-axis is time in milliseconds. The *y*-axis is the quality represented by the sensitivity of the target position to measurement errors.

Table 2. Average quality of each sensor in tracking targets for the three agent designs.

Agent designs	Average quality								
	Sensor1	Sensor2	Sensor3	Sensor4	Sensor5	Sensor6	Sensor7	Sensor8	All
Negotiating agents with CBR	190.3[11]	1.9	680.4	462.5	1800.1	6696.8	34.4	468.9	1322.5
Negotiating agents with single strategy	1159.0	2.3	932.6	1787.7	1390.3	4849.3	37.6	2463.9	1577.8
Reactive agents	497.2	2.6	949.0	2040.1	1767.1	4596.3	37.4	2129.3	1502.4

Note that our measure of "quality" is actually the expected average error. Thus, the smaller the value of quality, the better it is.

Overall, our technique scheduled and executed higher quality measurements. In two sensors, number 5 and number 6, we did worse. The number 5 sensor's performance is comparable to that of the reactive system, while sensor 6 does substantially worse than both other techniques. We suspect that these poor performances might be due to the initial casebases of the agents controlling these two sensors, and how the targets moved. Conceptually, sensor 6 corresponds to sensors 1, 2, and 8. These sensors with our CBR-powered negotiating agents were able to perform much better than with the other agent designs. Similarly, sensor 5 corresponds to sensor 4, which also had a better performance using the CBR-powered negotiating agent design. However, practically, it is possible that, due to how the two targets moved along the tracks, sensors 5 and 6 encounter situations not covered in the initial casebases and the agents do not have powerful-enough adaptations to learn to address those situations. We will investigate this further. In all other cases, though, our technique performed from exceptionally better (e.g. sensors 1, 4 and 8) to a little better (e.g. sensor 7), and did better on average.

One added measurement was the number of times a radar was turned on and measuring. Our technique averaged 63.1 target illuminations per radar per each 250-second experiment; the negotiating agents with a single strategy averaged 86.8; and the reactive agents averaged 89.1. In other words, our agents achieved higher quality measurements taken from the right radar sector at the right time, while *conserving* radar resources by not turning them on too often.

In summary, our results showed that in a dynamic, real-time environment of multi-sensor target tracking our proposed technique proved to be superior to a reactive agent architecture and a BDI architecture of negotiating agents that use a single negotiation strategy.

6. Related work

Agent-based negotiations involve information exchanges between two agents where the initiating agent desires to persuade the responding agent to accept a task. Two agents need to exchange information as each has only a partial set of the information or knowledge contained in the entire multi-agent environment, especially where

information cannot be updated and distributed quickly and accurately to all agents. Thus, when an initiating agent encounters a task that it believes that another agent can perform, it negotiates by informing that agent of the description of the task and its own constraints. In this way, each agent maintains a local information base and information is exchanged only when necessary.

Faratin et al. [15] presented a formal model of negotiation between autonomous agents. The authors defined *negotiation issues* such as price, volume, duration, quality, etc. For example, if an agent is trying to sell an item to another agent and both are negotiating about the price, then the price is the negotiation issue. The authors defined *negotiation tactics* as the set of functions that determine how to compute the value of an issue by considering a single criterion, including time, resources, previous offers and counter offers. There are time-dependent, resource-dependent, and behavior-dependent tactics such as polynomial, exponential, Boulware and Conceder functions. The authors ultimately defined a *negotiation strategy* as follows: First, for an issue, there is a weighted counter proposal, made up of a linear combination of the tactics that generates the value at the current time step of a negotiation, and the set of tactics is finite. Given a set of tactics, different types of negotiation behavior can be obtained by weighting the tactics in a different way and the weights are specified in a matrix. A negotiation strategy is then any function that is based on the matrix of the weights and the mental state of the agent. This definition provides a strong formal foundation for generating multi-tactic strategies and for modifying the negotiation strategy in real-time. However, the negotiation tactics are functional and thus prescribed. In our approach, we treat negotiation tactics as situational, temporal and even historical as our agents are able to learn from their past experiences and adapt old tactics to new situations.

Parsons et al. [13, 39, 40] proposed a comprehensive argumentative negotiation model that (1) used bridge rules for inference in different units [36] and (2) incorporated explicitly multi-context BDI agents [22, 43, 44] and the formalism described in [25] to construct arguments to evaluate proposals and counterproposals in negotiation. In particular, the authors made a strong case for sending over rules of an agent to another agent. This allows the other agent to know the actual reasoning process behind its counterpart's claims. In an agency of heterogeneous and competitive agents, this makes a lot of sense since it is logical to have agents that reason differently. Thus, naturally, to convince another agent, an agent lets its counterpart know about its reasoning steps. This approach promotes diversification and efficient *knowledge* (in addition to merely information) distribution within an agency. Of course, in a system of homogeneous agents in which all agents share the exactly same reasoning behavior, then such an argumentation-based negotiation model of Parsons and Jennings might not be necessary. The model is truly argumentative since agents not only argue about what evidence there is, but they also argue about how one reasons evidentially. Matos et al. [35] also investigated the use of genetic algorithms to evolve the above negotiation strategies. Strategies with high fitness values were chosen to survive to the next population generation. The fitness measure was a combination of gain (the seller's and buyer's scores) and cost (message exchanges). By the same token, there are different levels of acceptance, based on notions of argument defeat [28].

Amgoud et al. [4–6] further dealt with preferences between two arguments. First, an argument is undercut if and only if there exists an argument for the negation of an element of its support. Second, an argument arg_1 is preferred to another argument arg_2 according to *Pref* – a (partial or complete) preordering on the propositions – if and only if the level of the set of propositions in arg_1 is more referred then the corresponding set in arg_2. Using these two definitions, one is able to distinguish different types of relation between arguments. Suppose arg_1 and arg_2 are two arguments. Then, arg_2 strongly undercuts arg_1 if and only if arg_2 undercuts arg_1 and it is not the case that $arg_1 >>^{Pref} arg_2$, where $>>^{Pref}$ stands for the strict pre-order associated with *Pref*. If arg_2 undercuts arg_1 then arg_1 defends itself against arg_2 if and only if $arg_1 >>^{Pref} arg_2$. A set of arguments S defends arg_1 if there is some argument in S which strongly undercuts every argument arg_2 where arg_2 defends arg_1 and arg_1 cannot defend itself against arg_2. The set S of acceptable arguments of the argumentation system is the least fix-point of a set of propositions in the system that is defended by S. These definitions in turn guide the search for the set of arguments and counter-arguments (or defeaters) and provide the motivation for a dialogue. In our model, the preference of arguments is prescribed by the CBR system, and the dynamic part of this preference schemes lies with the prescribed *ordering* of this arguments. Our CBR system, based on its experience, ranks the order of the arguments to be communicated. Here, the ordering preference is based on the likelihood of an argument in persuading the other agent in the shortest time. In a way, this resembles the *degree of strength* of the argument in defending its associated request.

Kraus and Wilkenfeld [26] proposed a negotiation model on hostage crisis. The model is a modification of Rubinstein's model [38] of alternative offers which focuses on the passage of time and the preferences of the players for different agreements as well as for opting out of the negotiations. The authors assumed that agents cared only about the nature of the agreement or opting out, and the time at which the outcome was reached, and not about the sequence of offers and counter-offers that led to the agreement. In particular, no agent regrets either making an offer that was rejected or rejecting an offer. They also assumed disagreement is the worst outcome and the initiator gained over time and the receiver lost over time, since this was applied to hostage crisis. According to this model, if there is an agreement zone between the two players in the negotiation, an agreement will be reached in the first or the second iteration of the negotiation as the players either have complete knowledge of each other to begin with or have complete knowledge of each other after the first offer as time constraints are incorporated [24, 27]. The model is particularly useful when inter-agent communication has to be kept absolutely minimal (one or two interactions), individual agent knowledge is small, and less intelligent as a result. However, it does not generally apply to multi-agent negotiations where uncertain and incomplete knowledge is usually the case among autonomous agents.

Zlotkin and Rosenschein [46, 47, 59, 60] outlined a theoretical negotiation model for rational agents. Basically, the utility of an agent from a deal is the difference between the cost of achieving its goal alone and its expected part of the deal. A deal is *individual rational* if the utility of that deal to each partner of the deal is not negative, and *pareto-optimal* if there does not exist another deal that dominates it – there does not exist another deal that is better for one of the agents and not worse of

the other. A *negotiation set* is the set of all the deals that are both *individual rational* and *pareto-optimal*. In order for the negotiation set to be non-empty, a necessary condition is that there is no contradiction between the goals of the two agents. However, this condition is not sufficient since there may still be a conflict between the agents. A *conflict* is where any joint plan that satisfies the union of goals will cost one agent (or both) more than it would have spent achieving its own goal in isolation – i.e., no deal is individual rational. They presented a unified negotiation protocol (UNP) for conflict resolutions and agents taking partial cooperative steps. The protocol can be applied to two general agent scenarios. First, agents have been centrally designed to coexist in a single system and are predisposed toward cooperative activity – or in the least, there is some notion of global utility that the system is trying to maximize. Second, agents are self-serving, have their own utility functions and no global notion of utility. These agents have disparate goals and are individually motivated. Negotiation can be used to share the work associated with carrying out a joint plan (for the agents' mutual benefits), or to resolve outright conflicts arising from limited resources. When a conflict arises, the authors proposed a conflict resolution via coin toss based on weights. The authors further extended their model to task-oriented, state-oriented, and worth-oriented functions [61, 62, 63].

Lesser et al. [11, 12, 14] conducted research in which agent-based negotiations were essentially used for the collection and combination of various information pieces to derive meaningful and correct observations of the world. The approach stems from a partial global planning model [14], where an agent has the ability to (1) represent its own expected interpretation activities (of sensory data), (2) communicate about these expectations with others, (3) model the collective activities of multiple systems, (4) propose changes to one or more systems' interpretation activities to improve group performance, and (5) modify its planned local activities in accordance with the more coordinated proposal. Thus, the object of negotiation here is to combine information from several agents in order to reach an outcome for the entire group of agents. During negotiations, the goals, the long-term strategy and the rating of a plan are exchanged. Decker and Lesser [12] extended the work further to generalized partial global planning.

Based on the same partial global planning approach, Lander and Lesser [30, 31] introduced a framework called TAEMS that implements the cooperative (or negotiated) search and conflict resolution among heterogeneous, reusable, expert agents. The authors identified three negotiated-search strategies: (1) local search, (2) integration of local search, and (3) general negotiated search. A local search is performed individually by an agent within its current view of the shared solution space. This search includes an initiation, a critique – that describes the features of the proposal the agent agrees with and those it disagrees with [29], and a solution relaxation. In an integration of local search, solutions are critiqued by other agents, and the critique is used to determine local conflicts. The agent then performs relaxation (which lowers the utility of the solution to the agent) and computes the acceptability of the solution, based on the number of agents considering the solution to be acceptable. Finally, a general negotiated search is an opportunistic search augmented by the communication and assimilation of conflict information. Note that there is no

explicit *face-to-face* negotiation involved in the above approach. Agents exchange information through blackboards, and re-plan based on the critiques and conflicts. Then they re-post their solutions to the blackboard and the framework controller sends the information to every agent in the group. The solutions are refined until acceptable by all agents. However, the negotiated search in addressing inconsistencies such as relaxation is very important in agent-based negotiations, and the task group and commitment sharing is critical in negotiations as well.

Zeng and Sycara [58] proposed a negotiation model called Bazaar, powered by Bayesian belief networks. Given each response from the counterpart, an agent updates its beliefs (through the belief network) regarding the counterpart's reservation price. A supplier's reservation price, for example, is the price below which the supplier agent will not accept an offer. Then, the agent makes another offer based on the newly updated reservation price. The unique characteristic of this work is its use of Bayesian belief network, which allows modeling of constraints and hypotheses formally. However, one has to come up with the initial conditional probabilities to build the belief networks.

Kraus et al. [25] described a logical model for multi-agent argumentative negotiations and an implementation called the Automated Negotiation Agent (ANA). Given the mental states of the agents in their beliefs, desires, intentions, and goals, the authors defined argumentation as an interactive process emerging from exchanges between agents to persuade each other and bring about a change in intentions. There are six different argument types such as threats, promises of a future reward, appeals to past promises, etc. To decide which arguments to use, an agent uses negotiation meta-rules. For example, if the counterpart is a memoryless agent, then do not choose the argument type "appeal to past promise". Next, the agent ranked all generated arguments in terms of their strengths. For example, a threat is stronger than a promise of a future reward. Finally, the agent evaluates a request to decide whether to accept or reject it based on whether there is a contradiction, its convincing factor and its acceptance value. This logical model provides a comprehensive and sound foundation for competitive, selfish (to a certain degree), and argumentative agents. In our research, however, our agents are cooperative and, due to time constraints, cannot afford to be *over*-argumentative. In addition, our agents can neither pose a threat (since agents are cooperative) nor offer a promise of a future reward (since no agents in our application can guarantee its future state in a dynamic environment). Moreover, our protocol explicitly defines when certain states have to be true and for how long. This makes the real-time implementations feasible.

Our use of mental states and temporal logic shared significant similarity to the SharedPlan model of Grosz and Kraus [18, 19, 20] with the key difference being in the resolution of the temporal constraints that we refine in our negotiation model. As described in [20], the SharedPlan formalization provides mental-state specifications of both SharedPlans and individual plans. Shared plans are constructed by groups of collaborating agents and include subsidiary SharedPlans formed by subgroups as well as subsidiary individual plans formed by individual participants in the group activity. The formalization distinguishes between complete plans – those in which all the requisite beliefs and intentions have been established – and partial plans. In addition to the propositional attitude of intending to do an action, it introduces the

attitude of intending that a proposition hold. Stemming from this formalization, Grosz and Kraus [19] identified two axioms.

First, an agent can only have an intention-that toward propositions it believes are possible. The axiomization does not allow an agent to intend-that an impossible proposition hold. That is, $(\forall G, T_i)Int.Th(G, prop, T_i, T_{prop}, IC_{prop}) \Rightarrow \neg Bel(G, \neg prop, T_i)$ where if G is an agent that intends-that (for a duration of T_i) proposition *prop* to be true for duration T_{prop}, within the intention context of IC_{prop}, then G does not believe that the proposition is not possible during the duration of the agent's intention. Similar to our model, this SharedPlans model uses a variant of intentions, bounded by time to generate other mental states. In SharedPlans, the intention-that stems from the mutual belief of agents to make sure that the constraints for doing a task will hold. Similarly, we treat intentions as derived from beliefs and desires in our model. In addition, our agents also believe that a proposition is possible if they attempt to achieve that proposition during the negotiation process. The key difference is that in our model, due to the consideration of real-time delays, we allow for a mental state (e.g., *Bel*) – that has initially caused another mental state (e.g., *Int*) to hold – to become false before the intention is completely acted out.

Second, if an agent believes that the intended proposition *prop* is possible but does not currently believe it holds and knows some actions it can take to cause proposition *prop* to hold, then the agent must consider doing at least one of these actions. So, the second axiom in the SharedPlans model says the following. If an agent is uncertain about whether proposition *prop* holds and believes there are some actions any of which might lead to *prop* holding, then it must either (a) intend to do, or (b) be actively considering doing, one of these actions, or (c) it must have considered all of the actions and determined that it could not do any of them. Our model adopts a similar practice. However, the mental state of our agent in its intention is subtly different. In our model, when an agent intends to achieve a *prop*, it carries out a series of actions towards achieving that *prop*. These actions may lead to a modification of the original intention, and may lead to the termination of the original intention. In the face of conflicts that cannot be resolved, our agents *negotiate* to change the original intention so that an alternative *prop* can be achieved.

Gmytrasiewicz and Durfee [17] proposed a Recursive Modeling Method (RMM) approach to multi-agent coordination under time constraints, and one of their applications was anti-air defense. Even though agents do not negotiate explicitly, they make decisions based on their individual modeling of each other, and subsequent modeling of how each other models others.

Our domain of application has certain similarities with the distributed vehicle monitoring testbed (DVMT) (among many publications on the topic: [33, 37]) which was used as a tool to investigate distributed problem solving networks. The application is vehicle monitoring where each agent receives data only from its own individual sensor. The sensor coverage areas of the agents overlap. So, the task becomes one of modeling agent interpretations to ensure local and global consistencies. Hence, in a way, an agent is able to produce an interpretation by itself and needs to synthesize its own with other agents'. However, in our problem, an agent needs to cooperate with other agents just to obtain an interpretation of where the

target is. Hence, the DVMT problem focuses more on well-timed sensor fusion while ours more on coordination.

Distributed sensor networks have also recently received attention where paradigms and strategies from the multi-agent systems are used to solve the problem of allocating resources within such networks [34]. In such networks, the sensors may be heterogeneous in terms of what and when they could sense, the quality of their measurements, and the coverage area in which they operate. There are approaches based on negotiation such as dynamic mediation, argumentation, and satisficing search. There are also approaches based on hierarchical team organization for large-scale network arbitration and approaches employing distributed constraint satisfaction techniques dealing with both soft and hard constraints and peer-based optimization. A version of our work is reported in [34], focusing on a coalition formation architecture that learns [50].

7. Conclusions

This paper describes a real-time case-based negotiation model and its logical negotiation protocol. We first defined the characteristics of our agents and the cooperative environment that they work in. Next, we outlined a set of real-time constrained design guidelines and introduced our argumentative negotiation protocol. We described the protocol in a state transition diagram and as a logical framework. To better represent the axioms used to guide the rules of encounter between agents, we used the multi-context BDI framework and temporal logic to define *CanDo*, *Do*, and *Doing* predicates that link agent behaviors to negotiation motivations. We also defined logically motivations for a negotiation from the different viewpoints of the initiating and responding agents. Moreover, we utilized a set of communicative acts to handle a received message and to send out a message. We further described in detail two sets of axioms that govern the negotiation behavior between two agents. The axioms are real-time constrained, facilitated by a suite of functional predicates. The functional predicates are infrastructural tools that enable a negotiation process to monitor the pace of its negotiation, to determine whether it is on time, to determine whether and how to counter-offer, to analyze the acceptability of a counter-offer, and to evaluate evidence support for a requested task. These predicates are real-time and reflective. Then, we turned our attention to our case-based negotiation model. We also outlined a set of guidelines for the design of the model, particularly for enabling real-time negotiations. In addition, we designed and implemented a real-time logical framework and a set of real-time functional predicates.

Subsequently, we applied the model to the distributed sensor network domain in which multiple sensors controlled by different agents are required to collaborate to track targets. Our experiments showed that our model was able to select the appropriate sensor sectors and perform good-quality measurements better than purely reactive agents and negotiating agents with only a single negotiation strategy.

As for our future work, we aim to extend our model to consider multi-agent systems with heterogeneous agents, different types of resources, and multiple

concurrent negotiation issues. In Sections 3.3.7 and 3.3.8, we have hinted that our model assumes a multi-agent with agents of overlapping capabilities, such that when an initiating agent encounters a negotiation failure, it is able to approach another neighbor for help. In some heterogeneous multi-agent systems, this may not be possible. Thus, if an agent is determined to solve a task and if there is only one neighbor that has that capability, then the agent has no choice but to continue to negotiate with the neighbor. In a way, our proposed model is general enough to support the above scenario. However, the specific mechanism is not entirely clear. For example, our model will require some extensions, including a *notion of persistence* and a ''re-planning'' mechanism to overhaul the task description and allocation. The persistence notion will still have to trade-off time constraints and task accomplishments. We will also look into different types of resources: highly constrained, sharable, consumable, and otherwise. We will investigate the role of these negotiations in coalition formation – for example, if a coalition is deemed to fail due to a failed negotiation, should the initiating agent still continue with the remaining, ongoing negotiations? Finally, we will also study how real-time negotiations based on our model perform in noisy and uncertain environments. For example, if there is uncertainty in the requested task or there is message loss in the communication, the negotiation protocol has to be flexible enough such that the negotiating agents will factor uncertainty into their reasoning, and be more cautious in its communication.

8. Acknowledgments

The work described in this paper is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-2-0502. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, or the U.S. Government. The authors also would like to thank the anonymous reviewers for their comments that improved the paper.

Appendix A. Similar Axioms

Failure 2 When an initiating agent (i) receives a STOP message from a responding agent (r), it believes that the responding agent r does not desire to perform the requested task ρ and thus stops negotiating with r to perform the task, and the negotiation fails.

$$C_{in}:stop\big(r,i,Do(r,\rho),t_{cin,stop}\big)\wedge I:I_i\big(succeed(Negotiate(r,Do(r,\rho))),t'_I\big)\Rightarrow$$
$$B:B_i(\neg CanDo(r,\rho),t_B)\wedge D:D_i(\neg Do(r,\rho),t_D)\wedge I:I_i\big(\neg Negotiate(r,Do(r,\rho)),t''_I\big)\wedge$$
$$I:I_i(\neg succeed(Negotiate(r,Do(r,\rho))),t'''_I)\wedge rejected$$

where $During(t_{cin,stop}, t'_I) \wedge Meets(t'_I, t_B) \wedge Meets(t'_I, t_D) \wedge Meets(t'_I, t''_I)$. The communicative act *stop* is the encapsulation of receiving and parsing a STOP message. This rule is similar to *Failure 1*. In addition, it also states that the agent intends to not negotiate. In our current design we do not differentiate between an outright failure (failure type 1) and an opt-out failure (failure type 2) – both end with a *rejected* tag.

Failure 3I When an initiating agent (*i*) receives an ABORT message from a responding agent (*r*), it believes that the responding agent *r* no longer desires to perform the requested task ρ and thus stops negotiating with *r* to perform the task, and the negotiation fails.

$$C_{in}: abort(r, i, Do(r,\rho), t_{cin,abort}) \wedge I: I_i(succeed(Negotiate(r, Do(r,\rho))), t'_I) \Rightarrow$$
$$B: B_i(\neg CanDo(r,\rho), t_B) \wedge D: D_i(\neg Do(r,\rho), t_D) \wedge I: I_i(\neg Negotiate(r, Do(r,\rho)), t''_I) \wedge$$
$$I: I_i(\neg succeed(Negotiate(r, Do(r,\rho))), t''_I) \wedge abort$$

where $During(t_{cin,abort}, t'_I) \wedge Meets(t'_I, t_B) \wedge Meets(t'_I, t_D) \wedge Meets(t'_I, t''_I)$. The communicative act *abort* is the encapsulation of receiving and parsing an ABORT message. This rule is similar to *Failure 2*.

Failure 4I When an initiating agent (*i*) receives an OUT_OF_TIME message from a responding agent (*r*), it believes that the responding agent *r* no longer desires to perform the requested task ρ and thus stops negotiating with *r* to perform the task, and the negotiation fails.

$$C_{in}: out\_of\_time(r, i, Do(r,\rho), t_{cin,out\_of\_time}) \wedge I: I_i(succeed(Negotiate(r, Do(r,\rho))), t'_I) \Rightarrow$$
$$B: B_i(\neg CanDo(r,\rho), t_B) \wedge D: D_i(\neg Do(r,\rho), t_D) \wedge I: I_i(\neg Negotiate(r, Do(r,\rho)), t''_I) \wedge$$
$$I: I_i(\neg succeed(Negotiate(r, Do(r,\rho))), t''_I) \wedge out\_of\_time$$

where $During(t_{cin,out\_of\_time}, t'_I) \wedge Meets(t'_I, t_B) \wedge Meets(t'_I, t_D) \wedge Meets(t'_I, t''_I)$. The communicative act *out_of_time* is the encapsulation of receiving and parsing an ABORT message. This rule is similar to *Failure 2*.

Failure 3R When a responding agent (*r*) receives an ABORT message from an initiating agent (*i*), it believes that the initiating agent *i* no longer desires to negotiate and thus stop negotiating with *i* to perform the task, and the negotiation fails.

$$C_{in}: abort(i, r, Do(r,\rho), t_{cin,abort}) \wedge I: I_r(succeed(Negotiate(i, Do(r,\rho))), t'_I) \Rightarrow$$
$$I: I_r(\neg Negotiate(i, Do(r,\rho)), t''_I) \wedge I: I_r(\neg succeed(Negotiate(i, Do(r,\rho))), t''_I) \wedge abort$$

where $During(t_{cin,abort}, t'_I) \wedge Meets(t'_I, t''_I)$. The communicative act *abort* is the encapsulation of receiving and parsing an ABORT message. This axiom is similar to *Failure 3I*.

Failure 4R When a responding agent (*r*) receives an OUT_OF_TIME message from an initiating agent (*i*), it believes that the initiating agent *i* no longer desires to negotiate and thus stop negotiating with *i* to perform the task, and the negotiation fails.

$$C_{in}:out\_of\_time\big(i,r,Do(r,\rho),t_{cin,out\_of\_time}\big)\wedge I:I_r\big(succeed(Negotiate(i,Do(r,\rho))),t'_I\big)\Rightarrow$$
$$I:I_i\big(\neg Negotiate(i,Do(r,\rho)),t''_I\big)\wedge I:I_i\big(\neg succeed(Negotiate(i,Do(r,\rho))),t''_I\big)\wedge out\_of\_time$$

where $During\big(t_{cin,out\_of\_time},t'_I\big)\wedge Meets\big(t'_I,t''_I\big)$. This axiom is similar to *Failure 4I*.

Abort R When a responding agent (r) no longer intends to negotiate with an initiating agent (i) to perform a requested task ρ, it aborts the negotiation.

$$I:I_r\big(\neg Negotiate(i,Do(r,\rho)),t'_I\big)\Rightarrow C_{out}:abort\big(r,i,Do(r,\rho),t_{cout,abort}\big)\wedge$$
$$I:I_r\big(\neg succeed(Negotiate(i,Do(r,\rho))),t''_I\big)\wedge abort$$

where $During\big(t_{cout,abort},t'_I\big)\wedge Meets\big(t_{cout,abort},t''_I\big)\wedge Finishes\big(t''_I,t'_I\big)$. The communicative act *abort* is the encapsulation of composing and sending an ABORT message. It is similar to *Abort I*.

Out_of_time R When a responding agent (r) runs out of its allocated time for the negotiation with an initiating agent (i) to perform a requested task ρ, it aborts the negotiation.

$$B:B_r(\neg time(Negotiate(i,Do(r,\rho))),t_B)\wedge I:I_r\big(succeed(Negotiate(i,Do(r,\rho))),t'_I\big)\Rightarrow$$
$$C_{out}:out\_of\_time\big(r,i,Do(r,\rho),t_{cout,out\_of\_time}\big)\wedge I:I_r\big(\neg Negotiate(i,Do(r,\rho)),t''_I\big)\wedge$$
$$I:I_r\big(\neg succeed(Negotiate(i,Do(r,\rho))),t''_I\big)\wedge out\_of\_time$$

where
$During\big(t_B,t'_I\big)\wedge Finishes\big(t_{cout,out\_of\_time},t_B\big)\wedge Meets\big(t_{cout,out\_of\_time},t''_I\big)\wedge Meets\big(t'_I,t''_I\big)$.
This axiom is similar to *Out_of_time I*.

No_response R When a responding agent (r) detects receives no response from an initiating agent (i) during a negotiation, then it unilaterally quits the negotiation with a failure.

$$B:B_r(no\_response(i),t_B)\wedge I:I_r\big(succeed(Negotiate(i,Do(r,\rho))),t'_I\big)\Rightarrow$$
$$I:I_r\big(\neg Negotiate(i,Do(r,\rho)),t''_I\big)\wedge I:I_r\big(\neg succeed(Negotiate(i,Do(r,\rho))),t''_I\big)\wedge$$
$$channel\_jammed$$

where $During\big(t_B,t'_I\big)\wedge Meets\big(t'_I,t''_I\big)$. The *no_response* predicate is one of our real-time enabling functional predicates to be discussed in Section 3.4. This axiom allows an agent to bail out of a negotiation when the negotiation partner fails to respond.

Notes

1. A "good-enough, soon-enough" solution is sometimes known as a "satisficing" one [34] – a solution that satisfies the minimum problem requirements and the time constraints.
2. Since our agents must be able to perform multiple, concurrent negotiations, it is important for them to be multithreaded.

3. Take a simplified example, if a tracking coalition must be formed before a target enters a coverage zone that includes the projected path of the target and involves at least three different sensors, and if the negotiation runs longer than the allowable time, then the negotiation is considered to have lost its utility – even if a deal is reached, the coalition is no longer useful for tracking the target. This motivates our design to consider real-time issues significantly.
4. A "time step" can be any interval of time that is reasonable for an application. For target tracking, for example, a reasonable time step is in the order of tens of milliseconds.
5. This could be accomplished through a domain knowledge base that each agent has at creation, or a centralized broker or a mediating agent.
6. The utility-based ranking is detailed in [51, 52] and will not be repeated here.
7. This assumes that the agent knows of neighbors of similar capabilities such that it is able to approach a second neighbor for help when the first approached neighbor refuses to help.
8. See, for example, http://www.ghg.net/clips/CLIPS.html for a discussion on CLIPS.
9. Note that the workings of the persuasion threshold and evidence do not define to a joint intention in a formal sense. However, we see these two parameters as the keys to make possible for two negotiating agents to come to a deal with which both are satisfied, leading to an implicit joint intention in the end (if a deal is indeed reached).
10. There have been a variety of persuasion functions used in previous work in negotiation. Lander and Lesser [30, 31] used linear functions of local utility values over contract prices. Zlotkin and Rosenschein [61–63] suggested non-linear and exponential worth (or utility) functions and task-based, predefined worth functions. In [15], polynomial, exponential, Boulware tactics and Conceder are used.
11. An explanation of these numbers may be in order: Some sensors do not see a target often. During their idle time they measure the environment to see if a target is there. Such sentry duty is penalized since it results in no quality measurements. So, the actual values in the table fluctuate greatly across sensors. The important aspect is the comparison of the behavior of the same sensor using different techniques.

References

1. J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, 1983.
2. J. F. Allen, "Time and time again: the many ways to represent time," *Int. J. Intell. Syst.*, vol. 6, no. 4, pp. 341–355, 1991.
3. J. F. Allen and G. Ferguson, "Actions and events in interval temporal logic," *J. Logic Comput. Spl. Issue Actions Proces.*, vol. 4, no. 5, pp. 531–579, 1994.
4. L. Amgoud, S. Parsons, and N. Maudet, "Arguments, dialogue, and negotiation," in *Proceedings of the 14th European Conference on Artificial Intelligence*, Berlin, Germany, 2000.
5. L. Amgoud, N. Maudet, and S. Parsons, "Modelling dialogues using argumentation," in *Proceedings of the 4th Conference on Multi-Agent Systems (ICMAS'00)*, Boston, MA, USA, 2000.
6. L. Amgoud and S. Parsons, "Agent dialogues with conflicting preferences," in *Proceedings of the Workshop on Agent Theories, Architectures and Languages (ATAL'01)*, Seattle, WA, USA, 2001.
7. A. Bolotov and M. Fisher, "A clausal resolution method for CTL branching time temporal logic," *J. Exp. Theor. Artif. Intell.*, vol. 11, (1999), pp. 77–93, 1999.
8. F. Brazier and J. Treur, "Compositional modelling of reflective agents," in *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'96)*, Banff, Alberta, Canada, 1996.
9. P. R. Cohen and H. J. Levesque, "Intention is choice with commitment," *Artif. Intell.*, vol. 42, no. 2–3, pp. 213–261, 1990.
10. P. R. Cohen and H. J. Levesque, "Confirmation and joint action," in *Proceedings of IJCAI-91*, Sydney, New South Wales, Australia, 1991, pp. 951–959.
11. K. Decker and V. Lesser, "A one-shot dynamic coordination algorithm for distributed sensor networks," in *Proceedings of AAAI-93*, 1993, pp. 210–216.

12. K. Decker and V. Lesser, "Designing a family of coordination algorithms," in *Proceedings of ICMAS-95*, San Francisco, CA, 1995, pp. 73–80.
13. J. E. Doran, S. Franklin, S. N. R. Jennings, and T. J. Norman, "On cooperation in multi-agent systems," *Knowl. Eng. Rev.*, vol. 12, no. 3, pp. 309–314, 1997.
14. E. H. Durfee and V. Lesser, "Partial global planning: a coordination framework for distributed hypothesis formation," *IEEE Trans. Syst. Man Cybernet.*, vol. 21, no. 5, pp. 1167–1183, 1991.
15. P. Faratin, C. Sierra, and J. R. Jennings, "Negotiation decision functions for autonomous agents," *Int. J. Robot. Autonomous Syst.*, vol. 24, no. 3–4, pp. 159–182, 1998.
16. J. R. Galliers, "A strategic framework for multi-agent cooperative dialogue," in *Proceedings of ECAI'88*, Munich, Germany, 1998, pp. 415–420.
17. P. J. Gmytrasiewicz and E. H. Durfee, "Rational coordination in multi-agent environments," *Autonomous Agents and Multiagent Systems*, vol. 3, no. 4, pp. 319–350, 2000.
18. B. Grosz and S. Kraus, "Collaborative plans for group activities," in *Proceedings of the 1993 International Joint Conference on Artificial Intelligence (IJCAI-93)*, San Mateo, CA, 1993, pp. 367–373.
19. B. Grosz and S. Kraus, "Collaborative plans for complex group action," *Artif. Intell.*, vol. 86, no. 2, pp. 269–357, 1996.
20. B. J. Grosz and S. Kraus, "The evolution of SharedPlans," in Rao, A. and M. Wooldridge (eds.), *Foundations and Theories of Rational Agency*, Kluwer Academic Publishing, 1998.
21. H. Jakobovits and D. Vermeir, "Dialectic semantics for argumentation frameworks," in *Proceedings of the Seventh International Conference on Artificial Intelligence and Law*, 1999, pp. 53–62.
22. N. R. Jennings, "Controlling cooperative problem solving in industrial multi-agent systems using joint intentions," *Artif. Intell.*, vol. 75, (1993), pp. 195–240, 1993.
23. J. Kolodner*Case-Based Reasoning*.Morgan Kaufmann, San Mateo, CA, 1993.
24. S. Kraus, "Beliefs, time, and incomplete information in multiple encounter negotiations among autonomous agents," *Ann. Math. Artif. Intell.*, vol. 20, no. 1–4, pp. 111–159, 1997.
25. S. Kraus, K. Sycara, and A. Evenchik, "Reaching agreements through argumentation: a logical model and implementation," *Artif. Intell. J.*, vol. 104, no. 1–2, pp. 1–69, 1998.
26. S. Kraus and J. Wilkenfeld, "A strategic negotiations model with applications to an international crisis," *IEEE Trans. Syst. Man Cybernet.*, vol. 23, no. 1, pp. 313–323, 1993.
27. S. Kraus, J. Wilkenfeld, and G. Zlotkin, "Multiagent negotiation under time constraints," *Artif. Intell.*, vol. 75, (1995), pp. 297–345, 1995.
28. P. Krause, S. Ambler, M. Elvang-Gøransson, and J. Fox, "A logic of argumentation for reasoning under uncertainty," *Comput. Intell.*, vol. 11, (1995), pp. 113–131, 1995.
29. B. Lâasri, H. Lâasri, S. Lander, and V. Lesser, "A generic model for intelligent negotiating agents," *Int. J. Intell. Coop. Inform. Syst.*, vol. 1, no. 2, pp. 291–317, 1992.
30. S. Lander and V. Lesser, "Customizing distributed search among agents with heterogeneous knowledge," in *Proceedings of CIKM-92*, Baltimore, MD, 1992, pp. 335–344.
31. S. Lander and V. Lesser, "Understanding the role of negotiation in distributed search among heterogeneous agents," in *Proceedings of IJCAI-93*, Chambéry, France, 1992, pp. 438–444.
32. J. Lawton "The Radsim simulator". V. Lesser, C. L. Ortiz Jr., M. Tambe (eds.), *Distributed Sensor Networks: A Multiagent Perspective*. Kluwer Academic Publishers, The Netherlands, 2003, pp. 11—20.
33. V. Lesser and D. Corkill, "The distributed vehicle monitoring testbed: a tool for investigating distributed problem solving networks," *AI Magazine*, vol. 4, no. 3, pp. 15–33, 1983.
34. V. Lesser, C. L. Ortiz Jr., M. Tambe (eds.) *Distributed Sensor Networks: A Multiagent Perspective*. Kluwer, Norwell MA.
35. N. Matos, C. Sierra, and J. R. Jennings, "Negotiation strategies: an evolutionary approach," in *Proceedings of International Conference on Multiagent Systems (ICMAS)*, Paris, France, 1998, pp.182–189.
36. P. Noriega and C. Sierra, "Towards layered dialogical agents," in *Proceedings of ECAI Workshop on ATAL'96*, Budapest, Hungary, 1996, pp. 157–171.
37. C. Norman, V. Lesser, and Q. Long, "Distributed Sensor Interpretation: Modeling Agent Interpretations in DRESUN," in *University of Massachusetts Technical Report*, UMCS 93-75, 1993.
38. M. J. Osborne and A. Rubinstein*A Course in Game Theory*.MIT Press, Cambridge, MA, 1994.

39. S. Parsons and N. R. Jennings, "Negotiation through argumentation—a preliminary report," in *Proceedings of ICMAS-96*, Kyoto, Japan, 1996, pp. 267–274.
40. S. Parsons, C. Sierra, and N. R. Jennings, "Agents that reason and negotiate by arguing," *J. Logic Comput.*, vol. 8, no. 3, pp. 261–292, 1998.
41. H. V. D. Parunak, A. D. Baker, and S. J. Clark, "The AARIA agent architecture: an example of requirements-driven agent-based system design," in *Proceedings of the 1st International Conference on Autonomous Agents (ICMAS'97)*, 1997, pp. 482–483.
42. H. Prakken, "From logic to dialectics in legal argument," in *Proceedings of the Fifth International Conference on Artificial Intelligence and Law*, 1995, pp. 165–174.
43. A. Rao and M. Georgeff, "Modeling rational agents within a BDI-architecture," in *Proceedings of the* 2nd *International Conference on Principles of Knowledge Representation and Reasoning*, 1991, pp. 473–484.
44. A. Rao and M. Georgeff, "BDI agents: from theory to practice," in *Proceedings of ICMAS-95*, San Francisco, CA, 1995, pp. 312–319.
45. A. Rao and M. Georgeff, "Decision procedures for BDI logics," *J. Logic Comput.*, vol. 8, no. 3, pp. 293–342, 1998.
46. J. S. Rosenschein and G. Zlotkin*Rules of Encounter*.MIT Press, Cambridge, MA, 1994.
47. J. S. Rosenschein and G. Zlotkin, "Designing conventions for automated negotiation.," *AI Magazine*, vol. 15, no. 3, pp. 29–46, 1994.
48. T. W. Sandholm and V. R. Lesser, "Coalition formation among bounded rational agents," in *Proceedings of IJCAI-95*, Montreal, Canada, 1995, pp. 662–669.
49. M. Singh, "Developing formal specifications to coordinate heterogeneous autonomous agents," in *Proceedings of the 3rd International Conference on Multiagent Systems (ICMAS'98)*, 1998, pp. 261–268.
50. L.-K. Soh, H. Sevay, and C. Tsatsoulis, "A satisficing, learning, and negotiated coalition formation architecture," in V. Lesser, M. Tambe and C. Ortiz (eds.), *Distributed Sensor Networks: A Multiagent Perspective*, Kluwer Publishing, Chapter 7, 2003, pp.109–138.
51. L.-K. Soh and C. Tsatsoulis, "Reflective negotiating agents for real-time multisensor target tracking," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'01)*, August 6–11, Seattle, WA, 2001, pp. 1121–1127.
52. L.-K. Soh and C. Tsatsoulis, "Agent-based argumentative negotiations with case-based reasoning," in *Working Notes of the AAAI Fall Symposium Series on Negotiation Methods for Autonomous Cooperative Systems*, November 1–4, North Falmouth, MA, 2001, pp. 16–25.
53. B. Srinivasan, S. Pather, R. Hill, F. Ansari, and D. Niehaus, "A firm real-time system implementation using commercial off-the shelf hardware and free software," in *Proceedings of RTAS-98*, June Denver, CO, 1998, pp. 112–119.
54. K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa, "The RETSINA MAS infrastructure," Technical Report CMU-RI-TR-01-05, Robotics Institute Technical Report, Carnegie Mellon, 2001.
55. M. Tambe, "Towards flexible teamwork," *J. Artif. Intell. Res.*, vol. 7, (1997), pp. 83–124, 1997.
56. S. A. Vere, "Planning in time: windows and durations for activities and goals," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 5, no. 3, pp. 246–267, 1983.
57. M. Wooldridge and N. Jennings, "Intelligent agents: theory and practice," *Knowl. Eng. Rev.*, vol. 10, no. 2, pp. 114–152, 1995.
58. D. Zeng and K. Sycara, "Bayesian learning in negotiation," *Int. J. Human-Comput. Stud.*, vol. 48, (1998), pp. 125–141, 1998.
59. G. Zlotkin and J. S. Rosenschein, "Negotiation and task sharing among autonomous agents in cooperative domains," in *Proceedings of IJCAI-89*, Detroit, MI, August, 1989, pp. 912–917.
60. G. Zlotkin and J. S. Rosenschein, "Cooperation and conflict resolution via negotiation among autonomous agents in noncooperative domains," *IEEE Trans. Syst. Man Cybernet. Spl. Issue Distributed Artif. Intell.*, vol. 21, no. 6, pp. 1317–1324, 1991.
61. G. Zlotkin and J. S. Rosenschein, "Mechanism design for automated negotiation, and its application to task oriented domains," *Artif. Intell.*, vol. 86, no. 2, pp. 195–244, 1996.
62. G. Zlotkin and J. S. Rosenschein, "Mechanisms for automated negotiation in state oriented domains," *J. Artif. Intell. Res.*, vol. 5, (1996), pp. 163–238, 1996.
63. G. Zlotkin and J. S. Rosenschein, "Compromise in negotiation: exploiting worth functions over states," *Artif. Intell.*, vol. 84, no. 1–2, pp. 151–176, 1996.