

Knowledge-Based Human-Agent Teamwork for Distributed Training

Dianxiang Xu, Richard A. Volz, Michael S. Miller, and Jesse Plymale

Abstract — This paper presents a knowledge-based approach to human-agent mixed teams for distributed team training in CAST-DDD. CAST-DDD is a marriage between the multi-agent architecture CAST and the command-and-control simulation tool DDD, where CAST agents can replace some or all members on a DDD team. We explore the MALLET language in CAST to capture DDD teamwork knowledge that involve both humans and agents. To allow for adjustable autonomy of human team members, we provide a repetitive choice construct for embedding non-deterministic operations in human-agent team processes. To offer a visible picture of teamwork status, team processes are visualized and tracked via an extended formalism of Predicate/Transition nets. In addition, we describe different communication and coordination methods for members of a human-agent mixed team as well as how agents reason about the dynamic, partially observable environment of the DDD simulation.

Index Terms— intelligent agents, multi-agent systems, human-agent team, teamwork, team training

1. INTRODUCTION

Teamwork has become the most widely-accepted metaphor for capturing the nature of cooperation among multiple individuals (agents and/or humans) [1]. The advantage of teamwork results from a synergy among the team members, in which they combine their skills and resources and establish common communication protocols and decision-making procedures. Effective teamwork often relies on the acquisition of shared mental model, which is a common picture of the overall team goals, processes, and how each role fits in [2]. Due to the impact of mental models on team performance, fostering the development of shared mental models has been the target of practical methods for training teams [3, 4]. These team-training methods are of great importance not only for researchers to conduct empirical study on the key factors of effective teamwork, but also for modern industry to improve organizational productivity.

Multi-agent systems offer a potential for training teams that are geographically distributed. Intelligent software agents as automated teammates of human users on a team can make team training more flexible and cost-effective. With the help

of such partner agents, any number of trainees (typically smaller than the team size) can be organized and trained as a team in geographically distributed teamwork settings.

In addition to the traditional human-computer/agent interaction issue, however, the mixture of human users and partner agents raises significant challenges with respect to the notion of shared mental models for effective teamwork. A shared mental model may consist of shared ontology, goals, beliefs, team structure, and team process. Use of multiple agents for teamwork simulation often assumes a computational representation or approximation of shared mental models, especially team processes, such as Shared Plans [5, 6], or Team-Oriented Programming [7-9]. With the introduction of humans in the loop, characterizing the computational shared mental model or team process of a human-agent mixed team becomes more challenging. This is primarily due to full autonomy and non-determinism of human decision-making.

This paper, based on the work in [10], elaborates on a knowledge-based approach to human-agent mixed teams for distributed team training in CAST-DDD. CAST-DDD is a marriage between the multi-agent architecture CAST [11, 12] and the Command-and-Control (C2) simulation software DDD [13]. In CAST-DDD, software agents can replace any number of teammates on a DDD team. To do so, an explicit representation of expected human behaviors in human-agent team processes is highly desirable for several reasons [10]:

- A team process often requires individual team members to do certain actions at specific points of time. It may not always be feasible to describe a team process without specifying the responsibilities of some of the team members.
- Capturing the overall picture of a team process is important for teamwork design. The process for a human-agent mixed team can be independent of whether a role is played by an agent or human.
- For training purposes, the inclusion of human behaviors in human-agent team processes facilitates the representation of expert models, which specify what human trainees on a team are expected to do.

To capture teamwork knowledge that involves both humans and agents, CAST-DDD also extends the MALLET language with a repetitive choice construct for embedding non-deterministic operations in human-agent team processes [10]. This allows for adjustable autonomy of humans on a human-agent team. In this paper, we also exploit an extended formalism of Predicate/Transition (PrT) nets [14] to visualize

Dianxiang Xu is with the Department of Computer Science, North Dakota State University, Fargo, ND 58105, USA (e-mail: dianxiang.xu@ndsu.edu).

Richard A. Volz, Professor Emeritus, Department of Computer Science, Texas A&M University, College Station, TX 77843, USA (e-mail: volz@cs.tamu.edu).

Michael S. Miller and Jesse Plymale are with Department of Computer Science, Texas A&M University, College Station, TX 77843, USA (email: {mmiller,jwp2654}@cs.tamu.edu)

and track team processes so as to offer team members a visible picture of team status. In addition, we describe different communication and coordination methods for a human-agent mixed team as well as how agents reason about the dynamic, partially observable environment of the DDD simulation.

The rest of this paper is organized as follows. Section 2 reviews related work. To facilitate our discussion, Section 3 gives an introduction to the CAST and the DDD. Section 4 presents the CAST-DDD architecture. Section 5 describes how to specify human-agent teamwork knowledge in MALLETT and how to visualize and track team processes via extended PrT nets. Section 6 shows how agents reason about the dynamic, partially observable environment. In Section 7, we discuss how humans and agents communicate and coordinate with each other to accomplish team tasks. Section 8 concludes this paper.

2. RELATED WORK

Several computational models of teamwork have been proposed for the design of cooperative behavior among multiple software agents, such as BDI (Beliefs, Desires, and Intentions) model [15], joint intention theory [16, 17], shared plan model [5, 6], and team-oriented programming [7-9], etc. These models and relevant systems (e.g. STEAM [18, 19], GRATE* [17], SWARM Air Mission Teams [20] and CAST [11, 12]) primarily fit into the context of agent teams, or using agents as assistants of humans. In most cases that involve humans, a human and some agent comprise a single team member per se.

The closest work to ours is Steve agents [21, 22], which are virtual humans in a virtual reality world for team training. The virtual humans can serve as instructors for individual students and substitute for missing team members, allowing students to practice team tasks without requiring all their human teammates. Agents' tasks are represented by partially ordered plans. Each partially ordered plan consists of a set of actions, causal links, and ordering constraints. According to explicit speech acts in the task descriptions, humans and agents communicate through spoken dialogue. More recently, Steve agents have been empowered with negotiation and delegation capabilities [22]. While CAST-DDD and Steve agents share overall objectives, an essential difference is that Steve agents are not constructed in a team-oriented way in that they do not have an explicit representation of human-agent team structure and process, which are the key components of shared mental models. Given a team task, each Steve agent independently uses its task knowledge to complete individual task model. Another difference is that, for communication and coordination, CAST-DDD teams use operators, emails, and communication performatives, other than spoken dialogue.

The Teamcore [23] project has provided an agent integration architecture that enables a heterogeneous set of distributed agents from different research groups to work together to solve more complex problems. This architecture has also been applied to assist human collaboration by

automating many of routine coordination tasks. The key ideas include the use of proxies that are capable of general teamwork reasoning and locating agents that match specified requirements. Although both software agents and humans may be involved in the architecture, the architecture and the language for team-oriented programming do not appear to support explicit description of team structure and process of a human-agent mixed team.

COLLAGEN [24, 25] models dialogue, a form of joint activity, between a user and an agent based on the shared plan model. The agent and the user can be viewed as a human-agent mixed team, but the difference in the context of team structure and team process is obvious. Nevertheless, it provides a potential tool for augmenting the interaction between humans and their partner/coaching agents in CAST-DDD.

Heinze et al [26] have presented a case study on modifying agent-only teams to human-agent teams. To include human-in-the-loop in the existing maritime patrol and surveillance simulation system, where intelligent agents were originally designed for modeling all human components, user interfaces were developed to allow air force personnel to replace agents that previously modeled them. The application was used for exploration, evaluation and development of tactics and procedures rather than for training. In contrast, our work is to add intelligent CAST agents to existing simulation systems, like the DDD, designed for humans. As indicated by their experience, this is more complex than adding humans to a system designed for intelligent agents [26].

The team-planning environment MokSAF [27] allows two or more commanders to interact with one another to plan routes in a terrain. Each commander needs to plan a route from a start point to a shared rendezvous point. The individual planning is done by either an automatic route-planning agent, or joint work between a naive route-planning agent and the commander. The agent analyzes the route drawn by the commander and helps the commander refine the existing plan. Then the commanders must evaluate their plans from a team perspective and iteratively modify these plans until an acceptable team solution is developed. Commanders and their route-planning agents are called a human-agent team. Strictly speaking, the team supported by MokSAF is a human team, where each team member is assisted with an agent.

In addition, research on human-agent teamwork in the area of human-computer interaction often refers to the teamwork (or more precisely, group work) between interface agents and human users that involves substantial coordination (e.g. [28]). Coordination, although important for teamwork, does not necessarily mean teamwork in our context. Our research is based on the idea that teamwork is characterized by the notion of shared mental model. Our focus is on human-agent mixed teams where agents are automated peers or teammates of humans, and more specifically on explicit representation of human-agent team structure and process. In this sense, there is little literature comparable to our work.

3. CAST AND DDD: AN OVERVIEW

To facilitate our discussion on human-agent mixed teamwork in CAST-DDD, this section gives a brief review on both the CAST and the DDD.

3.1 CAST: Collaborative Agents for Teamwork

CAST is a multi-agent architecture for simulating teamwork [10, 11, 12]. It provides a prefix notation-based language, MALLET, for specifying team structure (membership, roles, and capabilities of individuals on the team) and teamwork process (i.e. plans for the team to perform the tasks). Agent capabilities are basic actions, or operators, of which agents are capable. Operators are specified in terms of preconditions and post-conditions. They need to be implemented in the specific application domain (e.g. DDD). Operators are also the basis for the hierarchical construction of team plans in the sense that they are atomic actions in a plan hierarchy. Individual or team plans are formed by operators, sub-plans, or arbitrary sequential, parallel, contingent, and iterative combinations.

The MALLET parser translates teamwork knowledge into extended PrT nets, an internal representation of agents' shared mental model. This model keeps evolving in the course of plan execution, and agents make use of this evolving model to determine on-the-fly how individual actions fit together. Although each agent on a team has a copy of the PrT nets that represent the team plans, agents only perform their own part and coordinate with others by communicating on the control tokens when executing the team plans. In addition, CAST uses the logic language JARE [29] to specify agents' knowledge and beliefs about their environment and teammates.

3.2 DDD: Distributed Dynamic Decision-Making

DDD is a client-server simulation tool for capturing the essential elements of a wide variety of C2 tasks in a dynamic teamwork environment [13]. It allows the experimenter to vary team structure, access to information, and control of resources. The recent generations of DDD provide an extensive set of capabilities for implementing complex, synthetic C2 team tasks. According to the pre-specified scenario, the DDD simulation sever coordinates all clients (or decision makers - DMs in DDD terminology). For example, the server notifies each simulation client when a new track is coming up at the designated time. The scenario definition language is used to configure various parameters of terrains, structure of decision makers, communication channels, resources (e.g. air bases and such assets or sub-platforms as fighters, AWACS craft - Airborne Warning and Control Systems, tankers, and helicopters), and tracks (also called targets). It allows for random generation of some track parameters, such as appearing time, velocity, and maneuver data of tracks. Each DM can issue a number of commands to operate on air base, assets, and tracks. Each operation is sent to the server. The server in turn broadcasts a message to other clients, which update relevant information for the clients.

Fig. 1 shows a typical DDD task screen (similar to that in [13]), which consists of four geographic quadrants. The

centermost 12×12 and 4×4 grids represent a restricted and highly restricted area, respectively. Neutral territory is the area outside the 12×12 grid. Each DM is responsible for one quadrant with a home base. DM2 and DM4 (DM1 and DM3) are in charge of the northern (or southern) area. The team's goal is to defend the restricted and highly restricted areas by monitoring the geographic space, identifying the friendly or unfriendly nature of all tracks, and destroying unfriendly tracks. As such, each DM has three different scores: personal, group, and team.

Each DM's base or launched asset has a detection ring radius and an identification ring radius (each asset also has other ring radiuses, e.g. the attack radius within which the asset can be used to attack a less powerful target). The DM can detect the presence or absence of any track within the detection ring, and discern the friendly or unfriendly nature of a track within the identification ring. Any track outside the detection ring was invisible to the DM. Each DM has also control of various types of vehicles, including AWACS crafts, tankers, helicopters, and jets. Each of these sub-platforms varies in range of vision, speed of movement, duration of operability, and weapons capacity. For instance, tankers are often defined as the most powerful yet the slowest vehicles. A detailed description about the DDD can be found in [13].

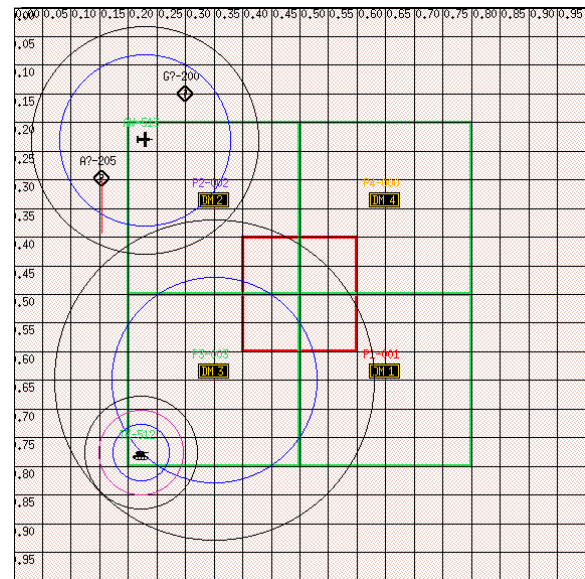


Figure 1. The DDD task screen

4. CAST-DDD

In CAST-DDD, agents can replace some or all of the DDD DMs and work as teammates of the rest human DMs. Fig.2 shows the CAST-DDD architecture. Agents in the original CAST architecture make decisions through the agent kernel based on their teamwork knowledge specified in MALLET, and individual knowledge represented in JARE.

In the CAST-DDD architecture, agents can perform the same DDD commands as human players, e.g. identifying a track. These commands are basic operators for construction of

individual and/or team plans. When an agent issues such a command, the CAST Agent Kernel forwards it to the DDD Domain Actor through the CAST Domain Actor. Upon receiving the command, the DDD Domain Actor applies it to the DDD simulation as if a human user had issued the corresponding command.

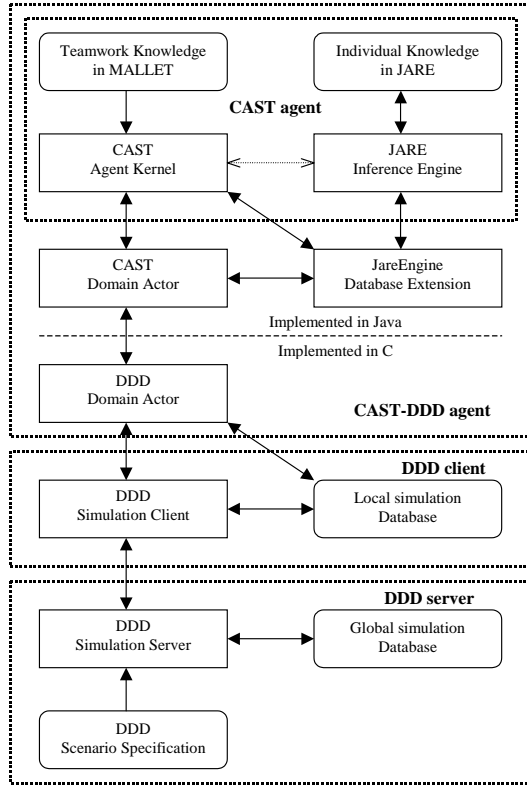


Figure 2. The CAST-DDD architecture [10]

As will be discussed in Section 6, CAST-DDD agents have their individual knowledge about the simulation environment. Since the DDD environment is dynamic (e.g. targets may appear without prior knowledge to agents or players, targets are moving at different velocities and directions), agents do not maintain the information that is frequently changing (e.g. aircraft positions). When needed, agents obtain the current state of relevant information through the JareEngine Database Extension. It communicates with the CAST Domain Actor, which sends requests via the socket to the DDD Domain Actor. The DDD Domain Actor then queries the Local Simulation Database and returns the results to the CAST Domain Actor and the JareEngine Database Extension. Since the CAST and the DDD are implemented in Java and C, respectively, sockets are used for the communication of commands and data between the CAST agents and the DDD simulation environment. Different ways of agent communication will be discussed in Section 7.

5. HUMAN-AGENT MIXED TEAMWORK IN CAST-DDD

In this section, we first discuss how to specify the structure and process of a human-agent mixed team, then describe the

visualization and tracking of team processes based on extended PrT nets.

5.1 Specifying Human-Agent Team Structure

A team structure consists of membership and capabilities of team members, either agents or humans. For example, the MALLET specification, `(team DDDteam (DM1 DM2 DM3 DM4))`, defines DDDteam as a team of four members, DM1, DM2, DM3, and DM4. We can also specify which team members are played by software agents. For instance, the specification `(agents DM1 DM2 DM3)` indicates that team members DM1, DM2, and DM3 are played by software agents. DM4 by default is a human team member.

The capabilities of each agent or human team member are further specified in terms of the basic actions of which they are capable. In CAST-DDD, these actions refer to the commands available to the DDD players, including launching assets, transferring assets to teammates, identifying a track, pursuing a track, returning to base, etc. These DDD commands are abstracted as MALLET operators. Operators are specified by their preconditions and post-conditions (effects). They bridge the gap between team structure, team plans, and the simulation environment in the sense that:

- Capabilities as part of team structure are defined by a relation between agents and sets of operators;
- Operator invocation is the basic construct for building hierarchical individual and team plans; and
- Operators are implemented in the simulation environment.

The following MALLET specification describes the *launch* operator for launching an asset.

```
(ioper launch (?asset)
  (precond (owner self ?asset))
  (effect (launched ?asset)))
)
```

For a team member to issue the *launch* command, it (i.e. `self`) has to be the owner of the asset. This constraint forms the precondition of the *launch* command. After the team member has successfully performed the *launch* command, the status of the asset becomes launched, which is defined as the effect. The actual effect of executing a command, however, is determined by the specific implementation of the operator. The specified effect may include only part of the actual post-conditions of an operator (sometimes, it is hard to define the complete post-conditions). Nevertheless, there are three considerations on using effects for operator specification:

- Operator effects provide a resource for team members to reason about information flow among teammates, based on which the proactive information exchange is made possible in the CAST. For instance, according to the effect of *identify*, a DDD team member may obtain the friendly or unfriendly nature of a specific target from the teammate who has performed the operator *identify* on the target. In this case, the teammate is the provider of the information.
- As a part of an abstraction that separates specification from implementation details, the effect

specification, together with the precondition specification, helps to describe system design in a more understandable way.

- Operators and pre-defined sub-plans may be used to do dynamic planning for some complex tasks, which usually requires the complete specifications of preconditions and post-conditions.

Consider the following operator *moveto*, which is corresponding to the DDD command for moving a launched asset to *location* (?x, ?y).

```
(ioper moveto (?asset ?x ?y)
  (precond (owner self ?asset)
            (launched ?asset))
)
```

The *moveto* operation does not guarantee that *?asset* will be finally located at (?x ?y) because *?asset* may run out of fuel, in which case *?asset* will return to home base automatically. To guarantee that the asset is to be located at the destination location, a process of returning-to-base, re-launching, and re-locating may be used to deal with the cases when the asset is running low on fuel. As a matter of fact, for the purposes of plan abstraction and reuse, we have defined a number of frequently used processes like this as basic sub-plans.

Now we can define the capabilities of a team member in terms of operators of which the team member is capable. Although in general team members may have different capabilities, DDD team members often share capabilities (e.g. all the commands available in DDD) but have different resources (e.g. assets) and workload (e.g. number of targets). The following specification says that agents DM1, DM2, and DM3 are capable of such operations as *moveto*, *transfer*, *launch*, *returntobase*, *identify*, *attack*, *fusion*, *pursue*, etc [10].

```
(capability (DM1 DM2 DM3)
  (moveto transfer launch
   returntobase identify attack
   fusion pursue transferinfo)
)
```

Consider DM4, which is played by a human. If the capabilities of DM4 are not specified, the human player by default can make full use of the DDD commands. However, for the following specification, DM4, even though played by a human, can only issue the commands listed in the specification (i.e., *moveto*, *transfer*, *launch*, *returntobase*).

```
(capability (DM4)
  (moveto transfer launch returntobase))
```

5.2 Encoding Human-Agent Team Process

A team process refers to the procedure of how team members (agents and/or humans) will achieve their common goal. This is specified as team and individual plans, where the DDD commands are primitive operators. For example, when a team member has found, or been told by teammates, that an unidentified target is moving towards its area of responsibility, the member might launch an asset, move it toward the target, identify the target, attack the target if unfriendly and not more powerful, and then return the asset to its base. This procedure involving a single member can be formalized by the following two individual plans *defend* and *handle-target*, where *handle-target* is a sub-plan of *defend*:

```
(plan defend (?craft ?target ?x ?y)
  (pre-cond (myasset self ?craft ?base))
  (process
    (seq
      (do (launch ?craft ?base))
      (do (moveto ?craft ?x ?y))
      (while (cond(not(id-range ?craft ?target)))
        (do (pursue ?craft ?target)))
      (do (identify ?craft ?target))
      (do (handle-target ?craft ?target))
      (do (returntobase ?craft ?base))
    )
  )
)

(plan handle-target (?craft ?target)
  (process
    (if (cond ((foe ?target)
              (morepowerful ?craft ?target)))
      (seq
        (while (cond(not(attack-range?
                        craft ?target)))
          (do (pursue ?craft ?target))
        )
        (do (attack ?craft ?target))
      )
      (do (transferinfo ?target))
    )
  )
)
```

The team member executing the *defend* plan uses asset *?craft* to accomplish the task. After launched, the asset is directed to the designated position (?x, ?y). While *?target* is beyond the identification range, *?craft* will pursue the target. The team member will identify the target once it is within the identification range of *?craft*. After the target is handled, the team member will return the asset to its base. Target handling is fulfilled by the *handle-target* sub-plan as follows: if the target is unfriendly and the asset is more powerful, the team member will attack the unfriendly target when the target reaches the attack range of its aircraft. If the target is friendly or the target is more powerful, the team member will transfer the information on the target to its teammates. When the *defend* plan is being executed, the CAST Agent Kernel sends the corresponding command to the DDD for each invocation of operators in the plan.

CAST-DDD supports two styles of human-agent teamwork:

- human-agent teams with loosely-coupled team processes; and
- human-agent teams with tightly-coupled team processes.

In the first approach, most of team plans are defined as individual ones and team members are relatively independent of each other in the team process. Cooperation primarily relies on explicit communication. Except for the mixture of humans and agents, this bears similarity to general multi-agent systems. The second involves substantial coordination in team-level plans. Both styles are useful for describing team-training scripts.

In a loosely coupled team process, each agent has their individual plans and cooperates with agent or human teammates via communication (communication will be discussed in Section 7). Nevertheless, we may still use team plans to describe the overall picture of a team process. The

following team plan describes that agents DM1, DM2, DM3 perform their individual plans independently, yet in parallel.

```
(plan teamtask-1
  (process
    (par
      (do DM1 (patrol AW 0.80 0.80))
      (do DM2 (defend JT 200 0.20 0.20))
      (do DM3 (defend HE 205 0.20 0.80))
    )
  )
)
```

The *teamtask-1* plan specifies that DM1 launches an AWACS (AW) craft to patrol its area of responsibility, whereas DM2 and DM3 launch their jet (JT) and helicopter (HE) to defend their areas of responsibility. 200 and 205 are identifiers of targets that have appeared on the screen. Consider DM4 as a human team member operating independently except for communication with partner agents. Since DM4 does not have any individual plan, he/she is fully autonomous and can do whatever the DDD client interface allows its user to do.

We may also define MALLETT plans for a human team member. Such plans often indicate an expert model for suggesting what the human player should do under the given situations. A coaching agent can provide instructions and hints, validate user commands against the plan steps, and provide feedback regarding whether the user's command is valid. This offers a trial-and-error style for training beginners step by step in training scenarios. Consider the following plan:

```
(plan teamtask-2
  (process
    (par
      (do DM1 (patrol AW 0.80 0.80))
      (do DM2 (defend JT 200 0.20 0.20))
      (do DM3 (defend HE 205 0.20 0.80))
      (do DM4 (defend HE 210 0.80 0.20))
    )
  )
)
```

DM4's actions should be those defined by plan invocation (do DM4 (defend HE 210 0.80 0.20)), although it is possible for DM4 to try any command the DDD client interface provides. DM4's only valid operation for the first step of (defend HE 210 0.80 0.20) is launching a helicopter according to the *defend* plan. DM4's operation will not take effect until it is consistent with the corresponding step in the plan. As such, the autonomy of a human team member would be limited if plans for this member are strictly procedural.

A parallel structure that involves different individuals refers to concurrent performance of individual plans. If a parallel structure of individual actions for the same team member (agent or human), the team member has the freedom to choose the ordering of actions and can only do one thing at a time (similar to the partial order semantics of concurrency). Consider the following parallel structure in an individual plan performed by DM4:

```
(par
  (do (moveto AW 0.80 0.20))
  (do (moveto TK 0.75 0.25))
  (do (moveto HE 0.70 0.30))
)
```

DM4 can direct the vehicles (AWACS, tanker, and helicopter) to specified positions in the order of his/her own choice.

We can also describe tightly-couple team process, i.e., team process with more coordination among teammates. Consider a divisional scenario in the CAST-DDD [10], where DM3 possesses all the 4 AWACS crafts. Suppose DM3 transfers an AWACS craft to each teammate at the beginning of the game. Then each team member will use an AWACS craft to monitor his/her borders of the restricted area. Before the game is finished, each of DM3's teammates will return the transferred AWACS craft to DM3. Therefore, DM3 needs to coordinate with each teammate for the transfer of an AWACS craft, and each teammate is not supposed to use the AWACS craft until DM3 has finished the entire transfer process. This scenario implies two basic forms of coordination:

- Operations like *transfer* require coordination between multiple team members; and
- Sequential steps involve multiple team members.

These coordination activities are fulfilled by the implementation of operators (e.g. *transfer*) and the CAST Agent Kernel, respectively. The following team plan describes the above scenario [10]:

```
(plan teampatrol
  (process
    (seq
      (do DM3 (transfer AW1 DM1))
      (do DM3 (transfer AW2 DM2))
      (do DM3 (transfer AW4 DM4))
    )
    (par
      (do DM1 (monitorAT AW1 0.8 0.8))
      (do DM2 (monitorAT AW2 0.2 0.2))
      (do DM3 (monitorAT AW3 0.2 0.8))
      (do DM4 (monitorAT AW4 0.8 0.2))
    )
    (do othertasks)
  )
)
```

The *monitorAT* sub-plan consists of several sequential steps, including launching an asset, moving the asset through a series of specified positions, and identifying tracks within the identification ranges, etc. Each team member cannot enter the parallel structure to execute their individual *monitorAT* plan until DM3 has finished the transfer of AWACS crafts. For simplicity, other tasks are abstracted into the *othertasks* sub-plan, the doer of which is by default the team that is executing the *teampatrol* plan. The above plan could remain the same regardless of whether a team member is played by an agent or a human [10].

5.3 Adjusting Human's Autonomy in Team Process

Generally speaking, humans and software agents on a team do have different behaviors. In CAST-DDD, human team members have to use the DDD client interface to perform actions manually, whereas software agents issue DDD commands automatically according their plans. On the other hand, specifying procedural plans may leave few choices for a human team member, which imposes a strong restriction on the human's autonomy. It is highly desirable that humans on a human-agent team could have adjustable autonomy and make more non-deterministic decisions on their own.

Consider the above *teampatrol* plan. Suppose DM4, as a

human team member, needs to keep using a certain number or all of the DDD commands before returning the transferred AWACS to DM3 (i.e. within plan *othertasks*). There are many choices for each single step, and it is up to the human to make each choice. For instance, the human may first launch any asset of his/her own. After launching an asset, there are more choices: in addition to launching another asset, the human may either move the launched asset towards any position, or pursue one of many tracks, or identify one of many tracks, and so on.

In our approach, we represent such a process by using a repetitive choice construct:

```
(do DM4
  (while (cond (not (launched ?anyasset)))
    (choice
      (launch ?craft1 base4)
      (moveto ?craft2 ?x ?y)
      (pursue ?craft3 ?target1)
      (identify ?craft4 ?target2)
    )
  )
)
```

where all variables are unbound. Suppose DM4 has not launched any asset before the execution of this process. For the first step, DM4 can do nothing but launching an asset from base4. All other choices (*moveto*, *identify*, and *pursue*) are disabled because their preconditions (e.g. the use of a launched aircraft) are not satisfied. After DM4 has launched an asset, these choices become available.

The repetitive choice construct in CAST-DDD is designed for embedding non-deterministic operations in team processes. The BNF-style notation is shown below [10].

```
(do [<doer>]
  (while (cond <condition>)
    (choice
      (<operator-invocation>)*
    )
  )))
```

where *<doer>* is a human team member. If *<doer>* is omitted when the embedding plan is an individual one, the doer is the one that is executing the embedding plan. For an agent, the above construct means that, for each step when the loop condition evaluates true, *<doer>* will attempt the choices in order until one is found that succeeds. If the *<doer>* is a human, the doer can choose any valid operator (whose precondition holds at current situation) and determine the values for free variables, if any, in the operator.

5.4 Visualizing and Tracking Human-Agent Team Process

Petri nets are a well-established formalism for modeling concurrency, synchronization, and non-determinism etc in distributed systems [30]. Covert and McNelis have modeled team decision making in Petri nets [31], and experienced the difficulty in constructing and maintaining complex Petri net models of teams due to limited expressiveness of traditional Petri nets. As a formalism of high-level Petri nets, however, PrT nets have been successfully applied to model and verify multi-agent behaviors [32, 33]. Different from the modeling and verification approach that uses PrT nets as a front-end tool, we have extended PrT nets to be an internal

representation for capturing evolving status of team processes. Specifically, the PrT nets for a team are generated automatically in terms of teamwork specification in MALLETT. The execution of team plans is captured by token games. Thanks to the graphical notation of PrT nets, we have implemented a tool for visualizing and tracking team processes. When humans are introduced into the loop of agent teamwork, the visualization offers human team members a visible picture of what their teammates are doing, which we believe is particularly useful for team training.

The major aspects of our extension to PrT nets include:

- hierarchical representation that is corresponding to the plan hierarchy in MALLETT,
- interaction with individual knowledge base. Specifically, transitions firings are evaluated and performed according to the corresponding knowledge base, and
- communication with other nets. The communication among agents may change the marking of a PrT net.

Hence, an extended PrT net consists of:

- Predicates (i.e. first order places). A predicate may represent control information or correspond to a predicate in knowledge bases or preconditions/effects of operators/plans;
- Transitions. A transition may represent an abstraction of another PrT net (i.e. an invocation of some sub-plan in MALLETT) or a basic operation (i.e. an invocation of some operator in MALLETT). Each transition is associated with a doer or a list of doers (i.e. a team).
- Labeled arcs between predicates and transitions. An arc label is a tuple of constants and variables, which represents the parameters of the corresponding predicate. An arc from a predicate to a transition indicates the fact that the predicate is a precondition of the transition firing. An arc from a transition to a predicate means that the predicate is an effect of the transition firing.
- Logical inscription formulae of transitions. An inscription is a precondition of transition firing, and evaluated in terms of team member's knowledge;
- Variable bindings for the corresponding plan invocation;
- Current marking, which represents current token distribution in predicates. Each token is a tuple of constants, representing either a fact, a belief, or a piece of control information (called control token).

Fig. 3 shows the PrT net generated for the *teampatrol* plan described in subsection 5.2. Each team member involved in plan *teampatrol* would have a copy of the PrT net. Invocations of operators and sub-plans (*transfer*, *monitorAT*, and *othertasks*) are represented by transitions. For an individual plan like *monitorAT*, the doer may expand the corresponding transition into another PrT net. For a team plan like

othertasks, each team member involved in the team plan may expand the transition. Transition t_2 denotes the start of the plan execution. It is fireable only when the precondition of the plan is satisfied. Predicates p_{14} , p_{15} , and p_{16} synchronize the sequential steps among multiple team members. Transitions t_6 and t_7 coordinate the start and end of the parallel branches, respectively. In the course of plan execution, each member performs their own part and coordinates with others via communication of control tokens. Consider the first step (`do DM3 (transfer AW1 DM1)`). Firing transition o_3 in the PrT net for DM2 does not mean DM2 is performing action (`transfer AW1 DM1`). Instead, it means DM2 is told that DM3 has finished this action and is ready to do the next.

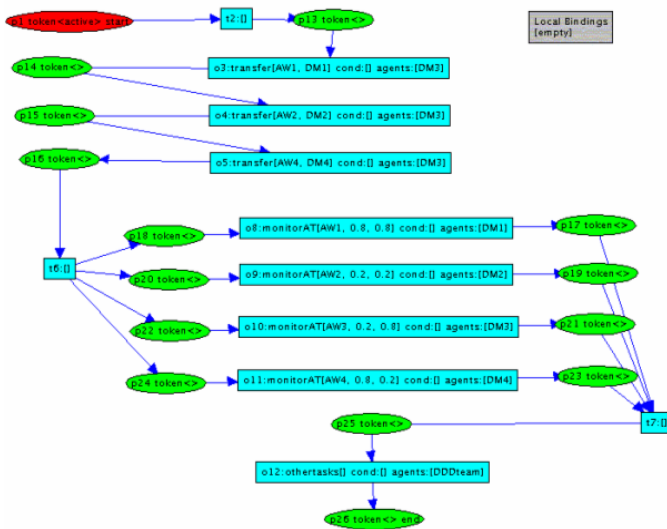


Figure 3. The PrT net generated for *teampatrol*

6. REASONING ABOUT THE DYNAMIC ENVIRONMENT

Just as human DDD users keep observing what is happening on the screen in order to get information for decision-making, CAST-DDD agents also need the up-to-date state of the simulation environment. Specifically, individual knowledge about the environment is required to evaluate the preconditions of operators and plans, and the conditions for contingent and repetitive statements (e.g. *if* and *while*) in the plan processes. For example, an agent executing the *defense* plan specified in the previous section must be able to evaluate the truth value of condition (`id-range ?craft ?target`).

The environment for CAST-DDD agents is dynamic (i.e. the targets as well as the agents' assets may be changing their positions, directions and speeds) and partially observable (i.e. an agent cannot observe a target unless the target is within the detection range of its base or launched assets). To deal with such an environment, we use the 'pull' technique for agents to obtain the current state of needed, observable information (e.g. the position of an observable target) from the environment. This is more efficient than the 'push' style, which may keep agents updated on all information in each simulation cycle.

Since CAST-DDD agents use JARE as the knowledge representation language, we specify the dynamic information by a predefined number of predicates. When such a predicate needs to be evaluated, the JareEngine Database Extension communicates with the DDD Domain Actor via the CAST Domain Actor to get the up-to-date values. Note that, these predefined predicates can be used to define inference rules in agents' knowledge base. The following are two rules for determining if a target is within the identification range of an asset, and if an asset is powerful enough to destroy a target, respectively.

```
(rule (id-range ?asset ?target)
  (position ?asset ?x1 ?y1)
  (position ?target ?x2 ?y2)
  (radar ID-range ?asset ?id-radius)
  (> (+ (* (- x1 x2) (- x1 x2))
      (* (- y1 y2) (- y1 y2))))
  (* ?id-radius ?id-radius))
(rule (morepowerful ?asset ?target)
  (strength ?asset ?power1)
  (strength ?target ?power2)
  (>= ?power1 ?power2))
```

where predicates *position*, *radar*, and *strength* represent information on the simulation environment. When conditions *id-range* and *morepowerful* in a MALLET specification are evaluated, the JareEngine Database Extension will get corresponding data about position, radar, and strength from the DDD Domain Actor.

7. COMMUNICATION FOR AGENT TEAMWORK

Communication is a critical element for effective teamwork. The CAST-DDD provides several ways for team members to communicate:

- Communication performatives as operators. Some operators in the CAST-DDD actually involve communication and coordination among teammates. For example, *transfer* and *transferinfo* allow team members to transfer assets and information on targets to their teammates. This type of communication and coordination is realized through the DDD commands. Generally, it is possible to abstract other communication actions as operators, and implement them in the simulation domain.
- Free form messages. They are primarily used for the communication between human team members.
- Formatted email messages: team members may send their team members email-like messages with predefined formats of performatives, which form a simple agent communication language, like KQML [34] or FIPA ACL [35]. Formatted messages are implemented based on the free-form messages in the DDD. They are suitable for both agent-agent communication and human-agent communication.
- Proactive information exchange: (inherited from CAST). Based on the analysis of information needs indicated by the preconditions of plans and operators in MALLET specification, agents can proactively provide information that is useful for other agents to

make decisions. For example, when an agent has identified a target that is in the area of a teammate, the agent may proactively notify the teammate the nature of the target.

Communication operators as well as formatted email messages (particularly outgoing messages) can be embedded in the specification of team process. In addition, a top-level plan may be defined as a parallel task for processing asynchronous messages from teammates. The following specifies a plan scheme for DM to handle various incoming messages:

```
(plan DM-comm
  (process
    (par
      (while (not (done))
        (do (handleTransferRequest)))
      (while (not (done))
        (do (handleTransferInfoRequest)))
      (while (not (done))
        (do (handleFormattedMessages)))
      ...
    )))
```

Each parallel branch handles one type of messages, e.g. requests for transferring an asset, requests for transferring information on some target, or formatted messages. A *while* loop is used to deal with all messages of the same type. The following plan specifies a simple way for responding to the messages of transfer requests. If the agent does own the requested asset and the asset is not occupied yet, the agent may agree to transfer the asset, otherwise decline the request.

```
(plan handleTransferRequest
  (precond (transferreq ?sender ?asset))
  (process
    (if (cond ((owner self ?asset)
              (free ?asset))))
      (do (transfer ?sender ?asset))
      (do (declinetransfer ?sender ?asset)))
  )))
```

It is worth mentioning that CAST_DDD agents may respond to their teammates and environment in a reactive way. The precondition `(transferreq ?sender ?asset)` of the *handleTransferRequest* plan may be viewed as a trigger for events of requesting asset transfer.

8. CONCLUSIONS

We have presented human-agent mixed teamwork for distributed team training in CAST-DDD, where software agents are automated teammates of human decision makers. An explicit representation has been explored to capture team structure and process that involves both humans and software agents. Although this work is based on the domain-specific simulation tool DDD, the approach is applicable to the integration of CAST with other similar simulation systems.

In the CAST-DDD experimentation, mixed human-agent teams have played a few DDD scenarios that were used for previous human-based experiments [36]. Input from AWACS subject matter experts was incorporated into our construction of basic expert models or teamwork plans. These plans consisted of patrolling for the purposes of detecting, identifying, and transmitting identification of friendly and hostile craft that appear in the simulation. Also, agents

supported attacking hostile threats in their control zones and refueling operations for friendly assets. The mixture of humans and agents, where subordinate or minor roles in performing teamwork plans were replaced by agents, allowed for teamwork or team training activities with fewer humans involved. The agents had the performance to run in a mixed human-agent team of up to five total teammates.

ACKNOWLEDGMENT

This work was supported in part by AFOSR (MURI) grant #F49620-00-1-0326. The DDD is the property of Aptima, Inc. (www.aptima.com). The authors thank the Aptima staff for making the DDD available to support the research presented in this paper. The authors thank other MURI group members, Dr. John Yen, Dr. Thomas R. Ioerger, Dr. Jianwen Yin, et al., for discussions. In particular, Dr. Ioerger and Mr. Joseph Sims initiated the idea of hooking into the DDD simulation by a separate program. The authors also thank Drs. John R. Hollenbeck and Daniel R. Ilgen at the Michigan State University for their support with the MSU-DDD and related documentation.

REFERENCES

- [1] P. R. Cohen and H. J. Levesque, "Teamwork," *Nous*, vol. 25, no. 4, pp. 487-512, 1991.
- [2] E. Blickensderfer, J. A. Cannon-Bowers, and E. Salas, "Theoretical Bases for Team Self-Correction: Fostering Shared Mental Models," in *Advances in Interdisciplinary Studies of Work Teams*, vol. 4, S. Beyerlein, Ed. Greenwich, CT: JAI Press, 1997, pp. 249-279.
- [3] W. B. Rouse, J. A. Cannon-Bowers, and E. Salas, "The Role of Mental Models in Team Performance in Complex Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 6, pp. 1296-1308, 1992.
- [4] J. A. Cannon-Bowers and E. Salas, "A Framework for Developing Team Performance Measures in Training," in *Team Performance Assessment and Measurement: Theory, Research and Applications*, C. Prince, Ed. Hillsdale, NJ: Lawrence Erlbaum Associates, 1997, pp. 45-62.
- [5] B. Grosz and S. Kraus, "Collaborative Plans for Complex Group Actions," *Artificial Intelligence*, vol. 86, no. 2, pp. 269-357, 1996.
- [6] B. Grosz and S. Kraus, "The Evolution of Shared Plans," in *Foundations and Theories of Rational Agency*, M. Wooldridge, Ed., 1998.
- [7] G. Tidhar, "Team-Oriented Programming: Social Structures," *Australian Artificial Intelligence Institute Technical Notes*, no. 47, 1993.
- [8] D. Kinny, M. Ljungberg, A. S. Rao, E. A. Sonenberg, G. Tidhar, and E. Werner, "Planned Team Activity," *Proceedings of the fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW'92)*, 1992.
- [9] D. V. Pynadath, M. Tambe, N. Chauvat, and L. Cavedon, "Toward Team-Oriented Programming," *Proceedings of the Sixth Int'l Workshop on Agent Theories, Architectures, and Languages (ATAL'99)*, 1999.
- [10] D. Xu, R. A. Volz, and M. S. Miller, and J. Plymale, "Human-Agent Teamwork for Distributed Team Training", In *Proc. of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI03)*, pp. 602-607, IEEE Computer Society, Nov. 2003.
- [11] J. Yen, J. Yin, T. R. Ioerger, M. S. Miller, D. Xu, and R. A. Volz, "CAST: Collaborative Agents for Simulating Teamwork," *Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI2001)*, Seattle, WA, 2001.
- [12] J. Yin, M. S. Miller, T. R. Ioerger, J. Yen, and R. A. Volz, "A Knowledge-Based Approach for Designing Intelligent Team Training Systems," *Proc. of the Fourth International Conference on Autonomous Agents*, Barcelona, Spain, 2000.
- [13] D. L. Kleinman, P. W. Young, and G. Higgins, "The DDD-III: A Tool for Empirical Research in Adaptive Organizations," *Proceedings of the*

1996 Command and Control Research and Technology Symposium, Monterey, CA, 1996.

- [14] H. J. Genrich, "Predicate/Transitions Nets," in *Petri Nets: Central Models and Their Properties: Advances in Petri Nets, LNCS vol. 254*, Springer-Verlag, 1987, pp. 207-247.
- [15] A. S. Rao and M. Georgeff, "BDI Agents: From Theory to Practice," First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, 1995.
- [16] P. R. Cohen and H. J. Levesque, "Intention Is Choice with Commitment," *Artificial Intelligence*, vol. 42, no. 3, pp. 213-261, 1990.
- [17] N. R. Jennings, "Controlling Cooperative Problem-Solving in Industrial Multiagent Systems Using Joint Intentions," *Artificial Intelligence*, vol. 75, no. 2, pp. 195-240, 1995.
- [18] M. Tambe, "Agent Architectures for Flexible, Practical Teamwork," National Conference on Artificial Intelligence (AAAI-97), 1997.
- [19] M. Tambe, "Towards Flexible Teamwork," *Journal of Artificial Intelligence Research*, vol. 7, no. 1, pp. 83-124, 1997.
- [20] G. Tidhar, C. Heinze, and M. Selvestrel, "Flying Together: Modeling Air Mission Teams," *Journal of Applied Intelligence*, vol. 8, no. 3, pp. 195-218, 1998.
- [21] J. Rickel and W. L. Johnson, "Virtual Humans for Team Training in Virtual Reality," *Proc. of Ninth International Conference on AI in Education*, 1999.
- [22] D. Traum, J. Rickel, J. Gratch, and S. Marsella, "Negotiation over Tasks in Hybrid Human-Agent Teams for Simulation-Based Training," *Proc. Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2003.
- [23] D. V. Pynadath and M. Tambe, "An Automated Teamwork Infrastructure for Heterogeneous Software Agents and Humans," *Journal of Autonomous Agents and Multi-Agent Systems*, 2003.
- [24] C. Rich and C. L. Sidner, "COLLAGEN: When Agents Collaborate with People," *Proc. of the Int'l Conference on Autonomous Agents*, 1997.
- [25] C. Rich and C. L. Sidner, "COLLAGEN: A Collaboration Manager for Software Interface Agents," *User Modeling and User-Adapted Interaction*, vol. 8, no. 3-4, pp. 315-350, 1998.
- [26] C. Heinze, S. Goss, T. Josefsson, K. Bennett, S. Waugh, I. Lloyd, G. Murray, and J. Oldfield, "Interchanging Agents and Humans in Military Simulation," *AI Magazine*, 2002.
- [27] T. L. Lenox, T. R. Payne, S. Hahn, M. Lewis, and K. Sycara, "MokSAF: How Should We Support Teamwork in Human-Agent Teams?," Carnegie Mellon University, Technical Report, CMU-RI-TR-99-32 September 1999.
- [28] H. Lieberman and T. Selker, "Agents for the User Interface," in *Handbook of Software Agents*, J. M. Bradshaw, Ed. Cambridge: AAAI/The MIT Press, 2002.
- [29] T. R. Ioerger, "JARE MENU," Department of Computer Science, Texas A&M University, 2001.
- [30] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the Institute of Electrical and Electronics Engineers*, vol. 77, no. 4, pp. 541-580, 1989.
- [31] M. D. Coover and K. McNelis, "Team Decision Making and Performance: A Review and Proposed Modeling Approach Employing Petri Nets," in *Teams: Their Training and Performance*, E. Salas, Ed.: Ablex Pub Corp, 1992, pp. 247-280.
- [32] D. Xu, R. A. Volz, T. R. Ioerger, and J. Yen, "Modeling and Analyzing Multi-Agent Behaviors Using Predicate/Transition Nets," *International Journal of Software Engineering and Knowledge Engineering*, vol. 13, no. 1, pp. 103-124, 2003.
- [33] D. Xu, J. Yin, Y. Deng, and J. Ding, "A Formal Architectural Model for Logical Agent Mobility," *IEEE Transactions on Software Engineering*, vol. 29, no. 1, pp. 31-45, 2003.
- [34] T. Finin, Y. Labrou, and J. Mayfield, "KQML as an Agent Communication Language," in *Software Agents*, J. M. Bradshaw, Ed. Menlo Park, CA: AAAI Press, 1997.
- [35] FIPA, <http://www.fipa.org>.
- [36] C. O. Porter, J. R. Hollenbeck, D. R. Ilgen, A. P. J. Ellis, B. J. West, and H. Moon, "Backing Up Behaviors in Teams: The Role of Personality and Legitimacy of Need," *Journal of Applied Psychology*.



Dianxiang Xu received the B.S., M.S., and Ph.D. degrees in Computer Science from Nanjing University, China in 1989, 1992, and 1995, respectively. He is assistant professor of computer science at North Dakota State University, USA. From August 2000 to July 2003, he was research assistant professor and engineer in the Computer Science Department at Texas A&M University. From May 1999 to August 2000, he was a research associate at the School of Computer Science, Florida International University. Prior to that, he was associate professor in the Department of Computer Science and Technology, Nanjing University. His research interests are in the areas of software engineering, software security, software agents, and applied formal methods. He is a senior member of the IEEE.



Richard A. Volz received his B.S., M.S., and Ph.D. degrees in Electrical Engineering from Northwestern University in 1960, 1961 and 1964 respectively. He is now semi-retired. Prior to retirement, he was the Royce E. Wisenbaker Professor of Engineering in the Computer Science Department at Texas A&M University. He served as Department Head from 1988 to

1997, and resumed an active research career after that. Prior to joining Texas A&M University, Dr. Volz was founding Director the Robotics Research Laboratory and Professor of Electrical Engineering and Computer Science at the University of Michigan. He has served on five federal advisory boards: 1) the Air Force Scientific Advisory Board, 2) the Ada Board, 3) The Aerospace Safety Advisory Panel, a Congressional oversight committee on NASA, 4) the NASA Space Station Advisory Panel, and 5) the NASA Center of Excellence in Information Technology Advisory Panel. He has received the Decoration for Exceptional Civilian Service from the U.S. Air Force, two Special Service Awards and the Public Service Award from NASA, and the Millennium Medal and Robotics & Automation Society Distinguished Service Award from IEEE. He is the author or co-author of over 175 research papers, has led over \$15,000,000 in funded research projects, and is currently President of the IEEE Robotics and Automation Society. He is an IEEE Fellow.



Michael S. Miller received his B.S., M.S., and Ph.D. degrees in Computer Science from Texas A&M University, University of Houston at Clear Lake, and Texas A&M University in 1990, 1997, and 2006, respectively. He is currently an assistant research engineer in the Computer Science Department at Texas A&M University. He was commissioned in the U.S. Army as an officer in 1990. He worked for 6 years as a NASA contractor. His research interests include artificial intelligence in education, intelligent tutoring systems, multi-agent systems, and software engineering.

Jesse Plymale is a graduate of the Computer Science Department at Texas A&M University.