

Human-Agent Teamwork for Distributed Team Training

Dianxiang Xu

*Department of Computer Science
North Dakota State University
Fargo, ND 58105
dianxiang.xu@ndsu.nodak.edu*

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

*Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
{volz,mmiller,jwp2654}@cs.tamu.edu*

Abstract

This paper presents an approach to human-agent mixed teams for distributed team training, demonstrated through the integration of the domain independent multi-agent architecture CAST with the military command-and-control simulation software DDD (Distributed Dynamic Decision-making). We extend the CAST architecture in a way such that agents can replace any number of teammates on a DDD team. To capture the overall picture of teamwork, we explore the MALLETT language in CAST for the specification of team structures and processes that involve both humans and agents.

Keywords: multi-agent systems, teamwork, human-agent teams, team training.

1. Introduction

Computer-based simulation has been a popular approach to individual as well as team training [1]. For example, Porter et al. [2] have studied backing-up behaviors in teams with the aid of the DDD (Distributed Dynamic Decision-making) [3], a computer software system for simulating command-and-control (C2) tasks. The advances in multi-agent systems offer a natural, promising way for improving distributed team training, which allows for geographical distribution of team organization. Specifically, multi-agent teamwork has become a major concern in the past decade. Most of existing multi-agent teamwork systems, however, support either agent teams or human teams [4, 5]. Some agents for so-called human-agent teams [6, 7, 8] are essentially developed as software assistants of human users from the human-agent interaction perspective, not as automated peers or teammates of human users from the shared mental model perspective. Such teamwork simulation systems for team training require physical organization of teams before team training can be conducted. We argue that agents as automated teammates of human users can make team training more flexible and effective.

This paper presents an approach to human-agent mixed teams for distributed team training, demonstrated through the integration of the domain independent multi-agent architecture CAST for teamwork simulation [9, 10] with the C2-oriented simulation software DDD. The central ideas underlying CAST include a computational representation for approximating shared mental models for agent teams and automatic generation of communication and coordination in terms of pre-specified teamwork knowledge. To enable CAST agents to act as teammates of DDD human decision makers, these ideas need significant renovation. Specifically, how does one include humans in the representation of shared mental model, particularly team process and how does one deal with human-agent communication and coordination for teamwork? In this paper, we extend the CAST architecture so that agents can replace any number of teammates on a DDD team. To capture the overall picture of teamwork process, we explore the MALLETT language in CAST for the specification of team processes that involve both humans and agents.

The rest of this paper is organized as follows. Section 2 gives an introduction to CAST and DDD. Section 3 discusses the architecture for integrating CAST agents into the DDD. Section 4 describes how teamwork knowledge for DDD tasks can be specified. Section 5 concludes this paper.

2. CAST and DDD: An Overview

2.1 CAST

The domain independent multi-agent architecture CAST was designed to simulate effective teamwork by capturing the knowledge about team structure and teamwork process. A team structure specifies membership, roles, and capabilities of individuals on the team, whereas a teamwork process specifies plans for the team to accomplish its goals. The common prior knowledge about the team structure and process enables

the team members to establish an evolving shared mental model, represented by a variant formalism of Predicate/Transition (PrT) nets. Based on this computational model, agents may reason about the team state and the need of their teammates and adopt a communication strategy for determining whether and when they should assist teammates regarding their information needs.

CAST provides MALLET (Multi-Agent Logic Language for Encoding Teamwork), a language for specifying membership, roles, agent capabilities, goals, tasks, operators, and team/individual plans. Agent capabilities are defined in terms of operators. Plans are a procedural description of teamwork processes, i.e. how team members will achieve the goals or perform the tasks. An individual or team plan may consist of invocations of operators, sub-plans, or arbitrary combinations using various constructs such as sequential, parallel, contingent, iterative, etc. The MALLET parser compiles team plans into PrT nets, which are an internal representation of agents' shared mental model about the status of the team process.

In addition to teamwork knowledge, CAST agents also have individual domain knowledge and beliefs about their environment and teammates, represented in the logic language JARE (Java Automatic Reasoning Engine) [11]. Logical conditions and constraints in a MALLET specification are evaluated by the backward-chaining inference engine of JARE on the individual knowledge base. Based on the team structure, the evolving team process, and the agents' individual knowledge, the CAST Agent Kernel makes decisions regarding what task is assigned, what the next action is, whether communication with other agents is needed, etc.

2.2 DDD

DDD is a distributed multi-person simulation and software tool for understanding C2 issues in a dynamic teamwork environment. It provides an extensive set of capabilities for implementing complex, synthetic C2 team tasks. Each DDD player (or DM) is supported by a simulation client, which communicates with the DDD simulation server via sockets. The simulation server is in global control of the game and coordinates all clients according to the pre-specified scenario. Through a graphical user interface, a DDD client enables its human user to operate on air bases, assets, and tracks via a variety of commands, which include launching, moving and returning an asset, identifying the friendly or unfriendly nature of a track that is within the identification range of the owner's asset, pursuing a track, attacking unfriendly tracks within the attack range, and transferring assets or sending messages to teammates.

Our work is based on the MSU version of DDD and its simplified defense scenarios, and should also be easily

portable to other versions. Team members (decision makers) are assigned to four geographic quadrants. The centermost 4x4 grid marked off in red represents a highly restricted area. The 12x12 grid demarcated in green represents a restricted area. The area outside this restricted area is neutral territory. Each decision-maker has a home base inside their quadrant. DM2 and DM4 are in charge of the northern area, whereas DM1 and DM3 are in charge of the southern area. The team's goal is to keep unfriendly vehicles from moving into the restricted and highly restricted areas. The team's task is to monitor the geographic space, identify all tracks in terms of their nature (friendly or unfriendly), and then destroy unfriendly tracks threatening the restricted area.

3. CAST-DDD: The Architecture for Human-Agent Mixed Teams

The integration of CAST with DDD, called CAST-DDD, enables CAST agents to replace some or all of the players in a DDD simulation task. These agents can communicate and coordinate with their human teammates. Each human user may be associated with a coaching agent, which provides operational instructions, strategy, explanations, etc. (However, coaching agents are beyond the scope of this paper.) The architecture of CAST-DDD is shown in Fig. 1.

In CAST-DDD, one of the basic ideas is that agents must be able to perform the same commands DDD provides its human players, e.g. launching an AWACS craft (Air Warning and Control Systems) from the owner's base, pursuing an enemy aircraft, etc. We define these commands as basic operators of which agents are capable and use them to construct complex individual and/or team plans in MALLET. Once an agent decides to perform such an operation in the course of executing an individual or team plan, the CAST Agent Kernel sends the operating command to the CAST Domain Actor, which in turn communicates via a socket with the DDD Domain Actor. The DDD Domain Actor applies the command to the DDD simulation, achieving the same effect as if the corresponding command were issued by a human user.

In order for an agent to participate in a DDD teamwork simulation, the agent must make decisions according to its individual knowledge about the DDD simulation domain. For example, before the agent can attack a target with one of its assets, it has to make sure the target is already within the attack range of the asset and the asset is not less powerful than the target. Determining the truth-values of these constraints needs several pieces of information about the asset and the target, such as positions, strength levels, the attack range, etc. These constraints are often represented in

preconditions of operators and plans, logical conditions of contingent statements in the processes of plans, and even inference rules in agent's knowledge base. Considering the dynamic nature of the DDD environment, DDD-CAST agents do not store in their individual knowledge base the environment information that is frequently changing (e.g. aircraft positions). Instead, we extend JARE and its inference engine so that agents can obtain the current state of relevant information as needed in the course of decision-making. This is achieved by the JareEngine Database Extension, which predefines a set of predicates that correspond to the domain information in the DDD simulation database. The predefined predicates are used to define conditions in MALLET specifications and inference rules in agent's individual knowledge base. When referenced, these predicates invoke routines to query the simulation to obtain the needed information.

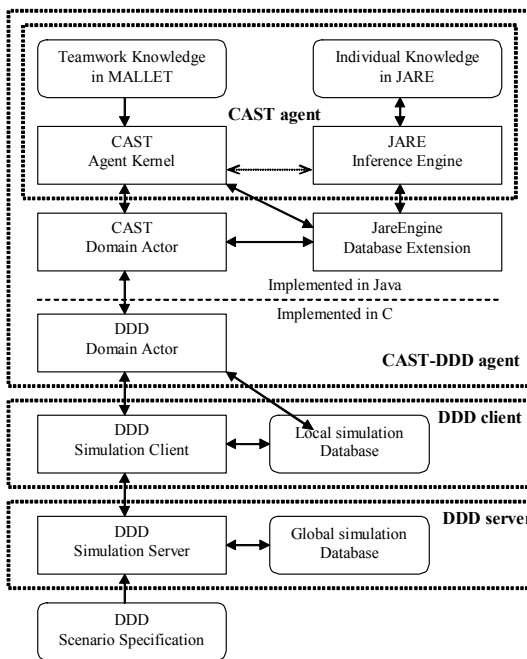


Figure 1. The CAST-DDD architecture

Since communication is critical for teamwork, CAST-DDD provides several ways for team members to communicate: 1) Operators as performatives. Operators such as *transfer* actually involve communication and coordination among team members; 2) Free form e-mail messages. They are primarily used by human team members; 3) Formatted email messages: team members may send others emails with predefined formats of performatives; 4) Proactive information exchange inherited from CAST. Based on the analysis of information needs in terms of preconditions of plans and operators, agents may proactively provide information that is useful for their teammates to make decisions.

4. Human-Agent Teams in CAST-DDD

4.1 Agent Capabilities and Team Structure

Agent capabilities are basic actions of which agents are capable. These basic actions are abstracted as operators specified by preconditions and effects. In CAST, operators bridge the gap between team structure, team plans, and the simulation domain in the sense that 1) capabilities as part of team structure are defined by a relation between agents and sets of operators, 2) operator invocation is the basic construct for building hierarchical individual and team plans, and 3) operators are implemented in the simulation domain.

We define each DDD command as an operator. Agents are capable of performing all operators that DDD provides its human users. For instance, the following individual operator *identify* corresponds to the DDD command a human would use for directing an asset to identify a target.

```
(ioper identify (?asset ?target)
  (precond (owner self ?asset) (launched ?asset)
    (id-range ?asset ?target))
  (effect (status ?target ?nature))
)
```

The precondition specifies that, to identify a target, the agent issuing the command (represented by the meta variable *self*) should be the owner of the launched asset, and the target should be within the identification range of the asset. For any team member to perform operator *identify*, this precondition must be believed. The effect specifies that the doer will know the friendly or unfriendly nature of the target upon successful execution of the command. Since each operator in CAST-DDD is implemented in the domain simulation, the actual effect of executing an operator is determined by the specific implementation. The specified effect does not necessarily include all actual post-conditions of an operator.

As noted above, we may define agent capabilities by associating each agent with a set of operators. Generally, different team members may have different capabilities. In the DDD, however, the team members share the same capabilities, though they may have different resources. The following MALLET example specifies that agents *DM1*, *DM2*, and *DM3* are capable of the operations listed.

```
(capability (DM1 DM2 DM3)
  (moveto transfer launch returnto base identify
    attack fusion pursue transferinfo ... )
)
```

In addition, we should define the membership of a human-agent team, including how many members are on the team and which member is played by an agent or human. This is often the first step in specifying a human-

agent team. The following specification defines DDDteam as a team of four members, *DM1*, *DM2*, *DM3*, and *DM4*.

```
(team DDDteam (DM1 DM2 DM3 DM4))
(agents DM1 DM2 DM3)
```

where *DM1*, *DM2*, and *DM3* are software agents, and *DM4* by default is a human team member who will use the DDD's client interface. For purposes of having a team of mixed humans and software agents, there is no need to define in MALLETT the capabilities of a human team member, such as *DM4*; however, if one were creating a more complex mixed system in which there was an agent coach, it might be advantageous to specify the operations of which the human is supposed to be capable.

4.2 Human-Agent Team Process

CAST-DDD supports team processes that involve both software agents and humans. In addition to supporting the trial-and-error style of learning, our considerations about including humans in the description of team processes include:

- 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 team processes without specifying the responsibilities of some team members, and
- Capturing the overall picture of teamwork 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.

A team process represented by individual and team plans describes the procedure of how team members (agents or humans) will perform the team task. Take, for example, the following formalized procedure involving a single team member:

```
(plan defense (?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))
      (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))
        )
      (do (returntobase ?craft ?base))
    )
  )))
```

The preceding code describes the actions an agent might take when it has discovered, or been told by its teammates, that an unidentified target is moving towards its area of responsibility; in response, the agent could launch its own asset, move the asset toward the moving target, identify the target, attack the target if unfriendly, and then return to its base.

The pre-condition, (*myasset self ?craft ?base*), means that the agent executing the plan uses one (i.e. *?craft*) of its assets to perform the plan; if none are available, the precondition evaluates false in terms of unification, and the agent cannot execute the plan. After the asset has been launched, the agent moves the aircraft to a designated position. If *?target* is not within the range in which it can be identified, *?craft* is directed to pursue it until it can be identified. Once the target is within the identification range, the agent is able to identify the target. If the target is unfriendly and the agent's aircraft is more powerful, the agent will attack the unfriendly target when the target reaches the attack range; if not, identification information will be transferred to other team members. Finally, the asset returns to base.

One basic style of human-agent team processes in CAST-DDD involves each agent having their individual plans and cooperating with agent or human teammates via communication. Individual plans are coupled into team plans, which specify the overall picture of a team process. The following team plan describes that agents *DM1*, *DM2*, *DM3* perform their individual plans in parallel.

```
(plan c2task-1
  (process
    (par
      (do DM1 (defense JT 205 0.80 0.80))
      (do DM2 (defense HE 200 0.20 0.20))
      (do DM3 (patrol AW 0.2 0.8))
    )
  )))
```

This plan specifies that *DM1* and *DM2* launch their jet and helicopter to defend their areas of responsibility, whereas *DM3* launches an AWACS craft to patrol its area of responsibility. 205 and 200 are the identifiers of targets that have appeared on the screen. Though for clarity and simplicity, specific target ids are given in our examples, more generally, one would have more complex plans in which the presence of targets would be discovered through asset sensors.

As in the previous example, suppose *DM4* is a human team member, who operates 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. For training purposes, however, MALLETT plans may be defined for a human team member. In this case, the plans specify the steps that the human is expected to do, and often indicate an expert model for the specific situation. A coaching agent may be used to provide

instructions and hints, validate user commands against the plan steps, and provide feedback regarding whether the user's command is valid. This provides a trial-and-error style for training beginners step by step in training scenarios. For instance, if we add (*do DM4 (defense TK 206 0.80 0.20)*) as a new parallel branch into the above example, *DM4's* actions should those defined by plan invocation (*do DM4 (defense TK 206 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 (*defense TK 206 0.80 0.20*) is launching a tank (*TK*) according to the previous specification of plan *defense*. *DM4's* operation will not take effect until it is consistent with the corresponding step in the plan.

Given a parallel structure of individual actions in a plan for single human team member, the human has freedom to choose the ordering of actions and can only do one thing at a time (similar to the partial order semantics of concurrency). Generally, specifying strictly procedural plans for a human may limit the user's autonomy.

Another style of human-agent team processes involves increased interaction and coordination among teammates. Consider a divisional scenario in CAST-DDD, where only *DM3* possesses 4 AWACS craft. Suppose the strategy to be adopted is that, at the beginning of the game, *DM3* transfers an AWACS craft to each teammate and each team member uses the AWACS craft it receives to monitor its borders of the restricted area. Before the game is finished, each of *DM3's* teammates will return their AWACS craft to *DM3*. In this case, *DM3* has 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 whole transfer. This simple procedure implies two basic forms of coordination: 1) operations like *transfer* that require the coordination between multiple team members, and 2) Sequential steps that involve multiple team members. The implementation of *transfer* and the CAST Agent Kernel will be responsible for the coordination activities. The first part of the strategy can be described in the following team plan:

```
(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))
      )
    )
  )
)
```

Where sub-plan *monitorAT* contains several sequential steps, such as launching an asset, moving the asset through a series of specified positions, identifying tracks within the identification ranges, etc. Each team member cannot enter the parallel structure to execute their *monitorAt* plan until *DM3* has finished the transfer of the AWACS craft. The above plan description could remain the same regardless of whether a team member is played by an agent or a human.

However, agents and humans in CAST-DDD do behave differently. Human team members have to use the DDD client interface to perform the actions manually. In the above example, *DM4* as a human team member can use any command that the DDD client interface provides before it returns the AWACS craft to *DM3*. How can we specify such activities for *DM4*? In essence, this is the same issue of overcoming the limitation on the users' autonomy mentioned earlier. Generally, 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 one of owned assets. 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. Such a process can be represented as follows:

```
(do DM4
  (while (cond (not (launched ?anyasset)))
    (choice
      (launch ?craft1 base4)
      (moveto ?craft2 ?x ?y)
      (identify ?craft3 ?target1)
      (pursue ?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 by their preconditions (e.g. the use of a launched aircraft) and may become available if their preconditions are satisfied after *DM4* has launched an asset. In general, the repetitive *choice* construct is used for embedding non-deterministic operations in team process. Its BNF-style notation is as follows.

```
(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. When the <doer> is a human, however, the interpretation is that the human can choose any choice from the list of valid operators. An operator is said to be valid if its precondition holds at current situation. If a free variable appears in the arguments of some operator, a human <doer> may also determine the value of the free variable (In contrast, the binding of a free variable for a software agent essentially depends on the ordering of relevant facts and rules in the knowledge base, because JARE is similar to Prolog). For humans, we further extend the semantics of choice to mean that if no operator is specified, the choice list includes all operators with free arguments. In this case, a human <doer> can select any of the valid options that the simulation provides. Given such a repetitive *choice* statement in some team process for training purpose, a coaching agent, if assigned to the human team member, may provide instructions and hints and validate each user's action against the valid choices.

5. Conclusion

We have presented an approach to human-agent mixed teams for distributed team training. The unique features of this work are the use of software agents as automated teammates of human decision makers, and an explicit representation of team structure and process that involves both humans and software agents. Although the approach is demonstrated through the integration of CAST with a specific system (DDD), we believe it is portable to similar distributed simulation systems.

CAST-DDD has undergone initial experimentation. Mixed human-agent teams have played a few simple scenarios. We have used previous human-based experiments to build some scenarios and incorporated input from AWACS expert into the construction of human-agent plans. These plans consist of patrolling for the purposes of detecting, identifying, and transmitting identification of friendly and hostile craft that appear in the simulation. Also agents can support attacking hostile threats in their control zones and refueling for friendly assets.

Our long-term objective is to provide the psychologists on our MURI project group, among other interested parties, with CAST-DDD as a tool for empirical study on team training. A coaching framework exists, but we are currently exploring supporting multiple approaches to monitoring and assisting the trainee. We also plan to design protocols of team training and parameters of team performance analysis.

Acknowledgment

This work was supported in part by AFOSR (MURI) grant #F49620-00-1-0326. 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. Dr. Thomas R. Ioerger and Joseph Sims developed 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 help.

References

- [1] 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.
- [2] 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*, vol. 1, 2002.
- [3] 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.
- [4] N. R. Jennings, "Controlling Cooperative Problem-Solving in Industrial Multiagent Systems Using Joint Intentions," *Artificial Intelligence*, vol. 75, pp. 195-240, 1995.
- [5] M. Tambe, "Towards Flexible Teamwork," *Journal of Artificial Intelligence Research*, vol. 7, no. 1, pp. 83-124, 1997.
- [6] P. Stone and M. Veloso, "Task Decomposition and Dynamic Role Assignment for Real-Time Strategic Teamwork," *Artificial Intelligence*, vol. 100, no. 2, pp. 241-273, 1999.
- [7] C. Rich and C. L. Sidner, "COLLAGEN: When Agents Collaborate with People," Proc. of the International Conference on Autonomous Agents, 1997.
- [8] 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.
- [9] J. Yen, J. Yin, T. R. Ioerger, M. S. Miller, D. Xu, and R. A. Volz, "CAST: Collaborative Agents for Simulating Teamwork," Proc. of IJCAI'2001, pp. 1135-1142, 2001.
- [10] 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.
- [11] T. R. Ioerger, "JARE MENU," 2001.