Collaborative Agents for C2 Teamwork Simulation

Dianxiang Xu, Michael S. Miller, Richard A. Volz, and Thomas R. Ioerger
Department of Computer Science
301 H.R. Bright Building
Texas A&M University
College Station, TX 77843-3112
{xudian, mmiller, volz, ioerger}@cs.tamu.edu

Abstract – Existing team training software often requires that trainees be organized as physical teams and the members of the same team be trained at the same time. To demonstrate that team training software can be made more flexible, this paper presents an approach to incorporating software agents into the distributed command-and-control (C2) simulation software DDD, which supports only human players. Based on the multi-agent architecture CAST, the software agents are designed as a team to perform C2 tasks in the DDD. By associating the agents with the same basic capabilities as the DDD provides its human users, we illustrate how to specify the teamwork knowledge for the agent team, and how to make agents efficiently reason about the dynamic, partially observable environment. Different methods of communication and coordination among agents are also briefly described.

Keywords: agents, multi-agent systems, teamwork, communication, command-and-control

1. Introduction

Teamwork has gained increasing attention in the areas of multi-agent systems [1-8], cognitive psychology [9-11], and team training [12-14]. The notion of 'shared mental model' [9], which is one of the psychological findings about teamwork, has been applied as a basic principle for the design of multi-agent teamwork [4]. Meanwhile, distributed software systems have also been playing an important role in psychological research on team performance and team training. For example, Porter et al [10] have studied helping behaviors of teammates with the aid of the DDD (Distributed Dynamic Decision-making) [15], a computer software system for simulating command-and-control (C2) tasks.

However, existing simulation software for team training, like the DDD, often supports only human players or trainees. This requires the organization of physical teams before team training can be conducted. On the other hand, existing multi-agent systems are seldom suitable for psychological study of team training because they support agent-only teams. It is naturally desirable that a multi-agent system could support human-agent mixed teams, where agents are peers or teammates of human users (this is essentially different from those agent systems, where agents are primarily assistants of human users). Such a system may significantly improve cost-effectiveness and flexibility of practical team training.

In this paper, we present an approach to incorporating software agents into the DDD so that the agents can replace the human players to perform teamwork. The agent team is designed based on the domain-independent multi-agent architecture CAST [4, 16]. We show how to extend the CAST architecture so that the agents can interact with the DDD in an effective way. We also demonstrate how to specify the teamwork knowledge for the agents to perform C2 tasks in the DDD and how the agents reason about their individual knowledge about the dynamic, partially observable environment of the DDD simulation. Different methods of communication and coordination among agents are also briefly described.

The rest of this paper is organized as follows. To facilitate our discussion, section 2 gives a brief introduction to the DDD. Section 3 discusses the architectural issue of integrating CAST agents into the DDD. Section 4 describes how to specify teamwork knowledge for the agents to perform C2 tasks in the DDD simulation, and how CAST agents can make use of the simulation information on the dynamic, partially observable environment. In Section 5, we discuss how agents communicate and coordinate with each other in order to accomplish team tasks. Section 6 concludes this paper with some directions of future work.

2. DDD: An Overview

The DDD is a distributed multi-person simulation and software tool for understanding C2 issues in a dynamic teamwork environment. The DDD was designed to capture the essential elements of a wide variety of C2 team tasks, and allow the experimenter to vary team structure, access to information, and control of resources [13, 15]. The recent generations of DDD provide an extensive set of capabilities for implementing complex, synthetic C2 team tasks. One of the DDD generations, MSU-DDD, together with a group of training and testing C2 scenarios, has been used by cognitive psychologists at Michigan State University for the study on helping behaviors and team performance [10].

Technically, each DDD player is supported by a simulation client, which communicates with the DDD simulation server via sockets. The simulation sever is in global control of the game, and coordinates all clients. For example, the server notifies each simulation client when a new track is coming up according to the scenario specification. Through a graphical user interface, a DDD client provides its human user a number of commands (operations) that can be used to perform C2 tasks, such as launching, moving and returning an asset (e.g. fighter, AWACS, tanker and helicopter), identifying friendly or unfriendly nature of a track that is within the identification range of the owner’s asset, pursuing a track, and attacking unfriendly track that is within the attack range, and transferring assets or sending messages to teammates. Each operation or message issued by a human player is sent to the server. The server in turn broadcasts the message to other clients, which update and display relevant information for the players.

Our work is based on the MSU-DDD and its simplified air defense scenarios, though it should be easily portable to other versions of DDD. Fig. 1 shows a typical task screen (similar to that in [10]). Team members (or decision makers or DMs in DDD terminology) are assigned to the four geographic quadrants, respectively. The centermost 4×4 grid marked off in red represents a highly restricted area. The 12×12 grid demarcated in green represents a restricted area. The area outside this restricted area is neutral territory. 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 (also called targets in this paper) threatening the restricted area.

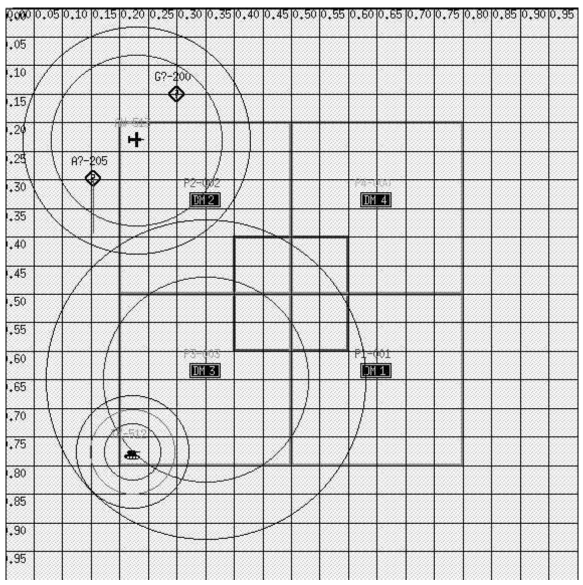

Figure 1. The MSU-DDD task screen

Each DM's base has a detection ring radius and an identification ring radius. 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 (also called sub-platforms, or assets), including AWACS, tanks, helicopters, and jets. Each of these sub-platforms varies in range of vision, speed of movement, duration of operability, and weapons capacity. A more detailed description can be found in [15].

3. Integrating Agents into DDD

The agents we have developed for DDD tasks are based on the CAST, a multi-agent architecture for modeling effective teamwork by capturing team structures and teamwork processes. The common prior knowledge about the team structures and processes enables the team members to develop an overlapping shared mental model, which is the source for a team member to reason about the states and the need of his/her teammates. With such a model, agents on a team can anticipate the actions and expectations of others (e.g. by knowing others' roles, capabilities, and commitments), and initiate proactive information exchange (knowing who to ask for information, or providing information proactively just when it is needed by someone else to accomplish their task).

To formally specify the teamwork knowledge, the CAST provides the MALLET (Multi-Agent Logic Language for Encoding Teamwork) language to define team organizations, roles and capabilities of agents, goals, tasks, operators, and team/individual plans. Operators, as schemes of basic atomic actions of agents are specified by preconditions and post-conditions. Plans are essentially the procedural description of teamwork processes, i.e. how they will achieve the goals or perform the tasks. Teamwork processes consist of invocations of atomic actions (operators), or arbitrary combinations using various constructs such as sequential, parallel, contingent, or iteration.

In addition to the teamwork level knowledge represented in MALLET, CAST agents also have individual domain knowledge and beliefs about their environment and teammates, represented in the logic rule based language JARE. Information on CAST, MALLET and JARE can be found in [4, 16]. Briefly, logical conditions and constraints expressed in MALLET are evaluated by invoking the backward-chaining inference engine of JARE on the individual knowledge base. Decisions such as what is the next action and whether communication with other agents is needed are made by the CAST Agent Kernel in terms of the teamwork knowledge and the agents' individual knowledge.

The integration of the CAST with the DDD, called CAST-DDD, enables CAST agents to replace human players in a DDD simulation task. These agents are able to communicate and coordinate with each other. The architecture of the CAST-DDD is shown in Fig. 2.

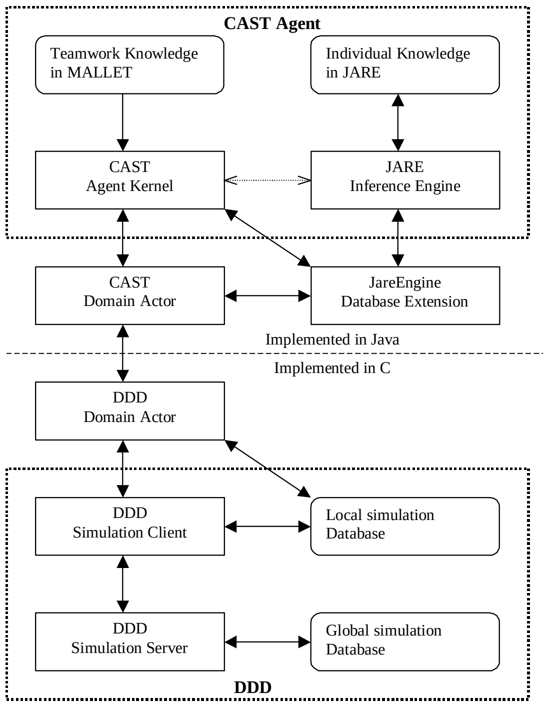

Figure 2. The CAST-DDD architecture

One of the basic ideas about the integration is that CAST agents must be able to perform the same commands as the DDD provides its human players, e.g. for launching an AWACS from the owner's base, pursuing an unfriendly track, etc. We define these commands as the basic operators that agents are capable of and use them to construct complex individual and/or team plans in a MALLET teamwork specification. Once an agent has decided 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 sends the command to the DDD simulation, achieving the same effect as if the corresponding command had been issued by a human player.

In order for an agent to perform a DDD simulation task in a teamwork setting, the agent must make decisions according to its individual knowledge about the DDD simulation domain.

For example, before the agent can attack an unfriendly target with one of its asset, 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 typically represented in preconditions of operators and plans, logical conditions of contingent statements in the processes of plans, and even inference rules in the agent's knowledge base (refer to next section for more details). Considering the dynamic nature of a DDD environment (e.g. targets may appear without prior knowledge to agents or players, targets are moving at different velocities, etc.), the agents do not store the environment information in their individual knowledge bases. Instead, we extend the JARE knowledge representation language and the inference engine so that the agents get only the current state of relevant information needed in the course of decision-making at the time they need it. In other words, the agents don't keep track of everything that is happening in the environment. 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. These predefined predicates can be used to define conditions in MALLET specifications and inference rules in agents' individual knowledge base.

The JareEngine Database Extension obtains environment information through the CAST Domain Actor, which sends requests via the socket to the DDD Domain Actor. The DDD Domain Actor then retrieves 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.

4. Agent Team for C2 Tasks

This section discusses two of the key issues about using intelligent agents as a virtual team for C2 tasks in the DDD simulation. These issues are: 1) the specification of teamwork knowledge (i.e. team structure and team process) for the agent team to perform given C2 tasks, and 2) the use of simulation information for individual agents to make decisions.

4.1 Specifying C2 teamwork knowledge

To enable software agents to be team members of DDD team tasks, agents should be able to perform the same commands as the DDD provides its human users. In the CAST-DDD, these commands are defined as corresponding operators. Some examples are listed as follows:

```
(ioper moveto (?asset ?x ?y))
(ioper identify (?asset ?target))
(ioper transfer (?asset ?to))
(ioper launch (?asset ?base))
(ioper returntobase (?asset ?base))
(ioper attack (?with ?target))
(ioper fusion (?with ?target))
(ioper pursue (?target ?with))
(ioper transferinfo (?asset ?to))
```

The operators are then used to specify team structures (team organization and capabilities etc), as well as team processes (plans). For example, we may have the following MALLET specification of team structure:

```
(team DDDteam (DM0 DM1 DM2 DM3 DM4))
(capability (DM1 DM2 DM3 DM4)
    (moveto transfer launch returntobase
     identify attack fusion pursue transferinfo
    ))
```

This means that team *DDDteam* consists of five agents, namely *DM0*, *DM1*, *DM2*, *DM3* and *DM4*, and *DM1-DM4* are capable of performing the listed operations. Here, *DM0*'s capability is not defined because it is an observer, as in the MSU-DDD.

As mentioned earlier, team processes, which describe the procedure of how agents will perform the team task, are captured by plans in the MALLET specification. In the CAST-DDD, the DDD commands, i.e. MALLET operators, are basic actions from which complex MALLET plans are constructed. For example, when an agent has found or been told by its teammates that an unidentified target is moving towards its area of responsibility, the agent would 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. This procedure can be formalized by the following MALLET plan:

```
(plan defense (?who ?craft ?target ?x ?y)
    (pre-cond (myasset ?who ?craft ?base))
    (process
         (seq
              (do ?who (launch ?craft ?base))
              (while (cond (not (launched ?craft)))
                (do ?who (await))
              )
              (do ?who (moveto ?craft ?x ?y))
              (while (cond (not
                       (id-range ?craft ?target)))
                   (do ?who (await))
              )
              (do ?who (identify ?craft ?target))
              (if  (cond (foe ?target)
                     (morepowerful ?craft ?target) )
                (seq
                   (while (cond (not (attack-range
                                   ?craft ?target)))
                                 (do ?who (await))
                   )
                   (do ?who (attack ?craft ?target))
                 )
                (do ?who (transferinfo ?target ?dm))
              )
              (do ?who (returntobase ?craft ?base))
         )))
```

The pre-condition of the above plan, *(myasset ?who ?craft ?base)*, means that the agent (*?who*) executing the plan uses one (i.e. *?craft*) of its assets (if none, the agent cannot execute the plan) to perform the plan. Since it takes some time to launch an aircraft from its base (which is determined by the DDD simulation), the agent has to wait until the aircraft is completely launched before the agent can move the aircraft to a designated position (while waiting, the agent may do other things concurrently, though). Once the target reaches the identification range of the aircraft, 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 of its aircraft. If the target is friendly or the target is more powerful, the agent will transfer the information on the target to the teammates. During the plan execution, the CAST Agent Kernel sends a corresponding command to the DDD for each invocation of operators in the above plan.

In MALLET, agents are allowed to perform parallel and concurrent tasks. The following is a parallel plan with a number of branches.

```
(plan c2task
   (process
      (par
       (do DM1 (defense DM1 JT 205 0.80 0.80))
       (do DM1 (defense DM1 HE 206 0.75 0.75))
       (do DM2 (defense DM2 JT 200 0.20 0.20))
       (do DM2 (defense DM2 TK 201 0.25 0.25))
       (do DM3 (patrol DM3 AW ?base3 0.2 0.8))
       (do DM4 (patrol DM4 AW ?base4 0.8 0.2))
      )))
```

This plan specifies that *DM1* and *DM2* launch their jet and helicopter to defend their areas of responsibility, whereas *DM3* and *DM4* launch their AWACS craft to patrol their areas of responsibility.

We can also specify strategic or tactical C2 knowledge in MALLET. For example, in one of the divisional MSU-DDD scenarios, each agent owns a jet, a helicopter, a tanker, and an AWACS. One possible strategy for playing the game is that, to keep alert on the coming waves of enemy attacks, each agent or player launches his/her AWACS to the border of his/her partner's area of responsibility. This could be specified by a similar structure of the above parallel plan.

4.2 Reasoning about the environment

Besides teamwork-level knowledge, the CAST-DDD agents also need individual knowledge, particularly on the environment, for the process of decision-making. Specifically, such individual knowledge is required to evaluate preconditions of operators and plans, and conditions for contingent and repetitive statements (e.g. if and while) in the processes of plans. For example, an agent that is executing plan *defense* specified in last subsection has to be able to evaluate the truth value of condition *(id-range ?craft ?target)*.

The environment for the CAST-DDD agents is dynamic (i.e. the targets as well as the agents' assets are changing their positions, and even 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 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. For example, the following is a rule for determining if a target is within the identification range of an asset.

```
(rule (id-range ?asset ?target)
    (position ?asset ?x1 ?y1)
    (position ?target ?x2 ?y2)
    (radar ?asset ?radius)
    (>  (+ (* (- x1 x2) (- x1 x2))
           (* (- y1 y2) (- y1 y2))
        )
        (* ?radius ?radius)
    )
)
```

where *(radar ?asset ?radius)* means that *?asset* can identify a target within the range of *?radius*.

5. Agent Communication

Communication is a critical element for coordination and cooperation in effective teamwork. The CAST-DDD provides three different ways of communication and coordination:

- Transfer of assets and information: agents can transfer their assets as well as information on targets to other agents. This type of communication and coordination is realized through the DDD commands, such as ‘transfer info’ and ‘transfer asset’, which are naturally defined as an individual operators in MALLET specification
- Formatted messages: agents may send other agents email-like messages with predefined formats (similar to a simple discourse language [17]). This is based on the free-form messages in DDD, which are useful for the communication among human players. However, the CAST-DDD agents cannot understand the messages that do not comply with predefined formats.
- 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 provides information that is useful for other agents to make decisions.

6. Conclusion

We have presented how to incorporate CAST agents into the distributed C2 simulation software DDD to replace human players to perform teamwork. The basic strategy has been implemented, though not all operators and predicates are yet complete, and only preliminary testing has been performed.

Our long-term objective is to develop software agents as teammates of human players as well as coaches. The use of agents as human partners on a team makes team training more flexible in that any number (typically smaller than the required team size) of trainees can be trained. This requires that MALLET be sufficiently expressive for the specification of the teamwork knowledge for human-agent mixed teams. For example, what roles are played by human trainees and what roles are played by agents, and how the MALLET supports plans that involve team training. On the other hand, the use of agents as coaches that provide immediate online instructions and suggestions can improve the effectiveness of team training. The design of such coach agents needs acquisition of more domain knowledge from C2 experts (e.g. what are good strategies for decision makers on a team to cooperate under different C2 situations).

Acknowledgment

This work was supported in part by AFOSR under MURI grant #F49620-00-1-0326. The DDD software is the property of Aptima, Inc. (www.aptima.com) and has been developed over the last 15 years with research and development funds from the Office of Naval Research and the Air Force Research Laboratory. The authors thank the Aptima staff for making the DDD available to support the research presented in this paper. The authors also thank Drs. John R. Hollenbeck and Daniel R. Ilgen at the Michigan State University for their support.

References

[1] P. R. Cohen and H. J. Levesque, "Teamwork," *Nous*, vol. 25, 487-512, 1991.

[2] M. Tambe, "Agent Architectures for Flexible, Practical Teamwork," National Conference on Artificial Intelligence (AAAI-97), 22-28, 1997.

[3] M. Tambe, "Towards Flexible Teamwork," *Journal of Artificial Intelligence Research*, vol. 7, 83-124, 1997.

[4] 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 (IJCAI'2001), Seattle, WA, 1135-1142, 2001.

[5] N. Jennings, "Controlling cooperative problem solving in industrial multi-agent systems using joint intentions," *Artificial Intelligence*, vol. 75, 195-240, 1995.

[6] 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), 226-256, 1992.

[7] P. Stone and M. Veloso, "Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork," *Artificial Intelligence*, vol. 110, 241-273, 1999.

[8] P. R. Cohen and H. J. Levesque, "Intention Is Choice With Commitment," *Artificial Intelligence*, vol. 42, 213-261, 1990.

[9] E. Blickensderfer, J. A. Cannon-Bowers, and E. Salas, "Theoretical Bases for Team Self-Correction: Fostering Shared Mental Models," *Advances Interdisciplinary Studies of Work Teams*, vol. 4, M. Beyerlein, D. Johnson, and S. Beyerlein, Eds. Greenwich, CT: JAI Press, 249-279, 1997.

[10] 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. I, 2002.

[11] M. T. Brannick, C. Prince, and e. al., "The Measurement of Team Process," *Human Factors*, vol. 37, 641-651, 1995.

[12] 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*, M. T. Brannick, E. Salas, and C. Prince, Eds. Hillsdale, NJ: Lawrence Erlbaum Associates, 45-62, 1997.

[13] D. Serfaty, E. E. Entin, and J. Johnston, "Team Adaptation and Coordination Training," in *Decision Making Under Stress: Implications for Training and Simulation*, A. Cannon-Bowers and E. Salas, Eds. Washington D.C.: Apa Press, 170-184, 1998.

[14] R. J. Stout, E. Salas, and J. E. Fowkes, "Enhancing Teamwork in Complex Environments Through Team Training," *Journal of Group Psychotherapy, Psychodrama & Sociometry*, vol. 49, 163-186, 1997.

[15] D. L. Kleinman, P. W. Young, and G. Higgins, "The DDD-III: A Tool for Empirical Research in Adaptive Organizations," Proc. of the 1996 Command and Control Research and Technology Symposium, Monterey, CA, 1996.

[16] 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, 427-434, 2000.

[17] B. Grosz and C. Sidner, "Plans for a discourse," in *Intentions in Communication*, J. Cohen, J. Morgan, and M. E. Pollack, Eds. Cambridge, MA: MIT Press, 417-444, 1990.