Defence Science Journal, Vol. 57, No. 6, November 2007, pp. 853-864

Modelling and Simulation of Tactical Team Behaviour

Sanjay Bisht, Aparna Malhotra, and S.B. Taneja
Institute for Systems Studies and Analyses, Delhi-110 054

ABSTRACT

Realistic military simulations are needed for analysis, planning, and training. Intelligent agent technology is a valuable software concept with the potential of being widely used in military simulation applications. They provide a powerful abstraction mechanism required for designing simulations of complex and dynamic battlefields. Their ability to model the tactical decision-making behaviour of simulated battlefield entities gives them an edge over other techniques. During battlefield simulation, these entities generally represent individualistic behaviour, taking operational order from higher control and executing relevant plans. However, since a complex battlefield scenario typically involves thousands of entities, their coordinated team behaviour should also be considered to make the simulation more realistic. This paper demonstrates the use of intelligent agent-based team behaviour modelling concepts in simulating the armoured tanks in a tactical masking scenario.

Keywords: Wargames, battlefield simulation, intelligent agents, team-oriented modelling, military training military simulation, modelling and simulation, decision making

1. INTRODUCTION

Battlefields are simulated so as to serve as a tool for training, analysis, tactical decision-making, doctrine proving, and evaluating weapon systems. Object-oriented methodologies and concepts[1,2] are generally used to model and develop such complex simulations since these provide quality maintainable software products. However, these are unable to effectively model the tactics of entities involved in complex battlefield systems, which require better problem decomposition, more powerful abstraction mechanism and better representation of organisational hierarchy. Intelligent software agents[3-7] represent an abstraction mechanism that encapsulates behaviour activation, provides rich interactions and has self-invocation capability. Such an abstraction mechanism has the potential to model tactical decision-making behaviour of battlefield entities because such problems map easily into agent-based concepts. Another advantage of using agent technology for tactical modelling is that these encourage separation of simulation logic (designed using object-oriented methodology) from the tactics representation. This separation of concern maximises the opportunity for reuse and the ability of simulations to absorb new entity models with minimal effort.

For modelling the battlefield entities using intelligent agent technology, the problem is formulated in terms of multiple, interacting, autonomous agents that have a particular objective (or explicit goals or desires) to achieve and preprogrammed with a set of plans. Plans are designed to achieve a particular goal under particular circumstances. Plans are executed when their invocation matches the defined conditions

Received 04 August 2006, revised 22 June 2007

provided that their context is appropriate. Plans are reasoned about using a complex reasoning engine. The agent must also have the intention to achieve these goals under varying circumstances. An intention represents the commitment of the agent to achieve a particular goal by progressing along a particular path that leads to the goal. Plans are used in combination with a particular goal to form an intention. The battlefield entity (represented by an) will make decisions according to its current beliefs (or perception) of the state of the battlefield. This reasoning behaviour has been borrowed from the theoretical belief-desire-intention (BDI) model of artificial intelligence in which agents have a view of the world (beliefs), certain goals they wish to achieve (desires), and they form plans (intentions) to act on these using their accumulated experience.

The tactical and reactive behaviour of battlefield entities such as tanks[8] has been modelled using JACK™ Intelligent Agent framework[8]. The paper gives details of how the BDI agent architecture has been extended to model tank, damager and detector agents using JACK™ Intelligent Agent as the implementation paradigm. The paper also discusses the issues and challenges of tactical behaviour modelling in the context of wargame simulation and strength of intelligent agent technology to overcome these.

In the above-mentioned study, the command and control organisational hierarchy, which plays a dominant role in any battlefield and its simulation has been ignored. In a typical complex battlefield scenario involving various arms like infantry, armour, artillery, mechanised forces, etc, there are thousands of entities following strict command and control. Apart from displaying individualistic behaviour (i.e., taking operational order from higher control and executing relevant plans), the entities also display coordinated team behaviour in certain situations (e.g. masking, recce and outflanking, etc.). In such situations, the actions of these individual entities are affected by their interaction with their peers and the state of the environment. While simulating this behaviour, often a top-level view is sufficient and we may not need the level of fidelity that results when each and every entity composing a team is simulated individually. For example, it may be sufficient to know that a regiment has moved and suffered a certain percentage of casualties instead of knowing the status of all the constituent troops and tanks. To avoid addressing the individual entities independently, their coordinated team behaviour[9] must be considered.

Team behaviour can be displayed by grouping the entities as teams using a number of different organisational structures. In such a structure, the supervisory units in the command hierarchy will delegate roles and responsibilities to groups of subordinate units. These units (sub-teams) can contain other teams as well. This results in a powerful organisational structure capable of modelling hierarchical as well as non-hierarchical battlefield interactions. A group of entities (team) is assigned some common goals and a broad specification of tasks (joint plans) to be carried out to achieve the mission. The tasks are broken down into sub-tasks and responsibilities are delegated to each entity (team-member) according to its capability. Thus, the actions performed by the entity that is part of a team are not determined only by its own individual state, but also by the joint state of the team. These actions are performed by executing the applicable and relevant plans.

In battlefield simulations and wargames, such coordinated team behaviour can be modelled using team-oriented modelling which is an extension of intelligent agents. In authors' previous work[8] they used agents to represent individual tanks. As an extension, they had modelled troops of tanks as teams of agents and implemented their coordinated team behaviour. They had applied these concepts on an armour masking scenario using JACK Teams, which is a powerful team-oriented programming tool. JACK Teams[10,11] is an extension to an existing Java-based multi-agent framework, JACK™ Intelligent Agents[12-16] that supports the well-established BDI agent architecture[17,18]. This paper focuses on the modelling and implementation details of the team agents.

2. EXTENDING AGENT CONCEPTS FOR TEAM-ORIENTED BEHAVIOUR

The meaning of teamwork is cooperative effort by the members of a team to achieve a common goal. Hence, team-oriented tactical modelling involves the formation of teams of battlefield entities and

their coordination to achieve common goals. This current situation and goal is seen from the abstract point of view of the team as a whole[19]. The BDI agents have beliefs, desires (goals) and intentions (plans). Extending this concept further, the team will have joint beliefs, joint goals and joint plans, which lead individuals or sub-teams in the team to intend to do their share (role) of the team activity. Hence, the individual team members will have individual beliefs, goals, plans, intentions, and mutual beliefs about the battlefield environment and about each other's actions. This may necessitate communication with other members of the team also.

Team-oriented concepts are implemented through an agent-based team reasoning entity that encapsulates coordinated team behaviour. In this model, although team members act in coordination by being given goals according to the specification, they are individually responsible for determining how to satisfy those goals. Hence, a complex task is decomposed into smaller manageable tasks.

In team-oriented modelling, all the agent concepts still hold, only tasks are associated with roles. Role is a very important concept that constraints an individual or a sub-team to undertake certain activities in service of the joint intention. The team declaration specifies which roles the team itself may perform for other teams and which roles it offers to other sub-teams to perform. Different team members will perform different roles to fulfil the joint intention, or the teamplan (which dictates the steps directing each sub-team to achieve specific goals). After formation, team members are referred within a plan by their role and associated to specific agents. This aids in specifying coordinated team behaviour in terms of individual agent collaborations. The specification includes: what the team is capable of doing (i.e., what roles it can fulfil); which roles it offers to other team members (agents) to fill; which components are needed to form a particular type of team; when/whether the team is willing to take on a particular role within another team; and the coordinated behaviour among the team members.

These concepts of teams requiring roles and team performing roles provide a framework where group behaviours and individual behaviours can be clearly separated. Group behaviour is specified in terms of the roles that are required to achieve the desired behaviour. This behaviour is specified independent of the actual teams performing the roles. However, the team has access to its sub-teams through the role container so it is able to perform reasoning based on the actual team membership, when necessary. The team specification determines what each member does, and it also handles failure of members to achieve their goals. Team members act in coordination by being given goals according to the specification, and they are themselves responsible for determining how to specify those goals.

3. APPLICATION: SIMULATION OF ARMOUR MASKING SCENARIO

Team work is an essential requisite for success in any organisational activity. In any battle, it is essential that all the participating arms, groups, and units must be integrated into closely knit teams and must fight in coordination. For this purpose, a detailed knowledge of the organisation, capabilities, limitations and tactics of each other is an essential prerequisite. The required coordination can be achieved in battlefield simulations by modelling the team behaviour. As a step forward in this direction, the ease of applying team concepts in tactical simulations is demonstrated by modelling an armour-masking scenario. Inset 2 in Appendix 1 refers to a brief introduction of tactical terms involved. Before detailing the team-oriented modelling of this scenario, a textual narrative of the tactics involved is presented below.

In this application, a scenario in which a combat group (CG) of combat command (CC) has been assigned the task of capturing an objective. (Fig.1). Combat group starts from forward assembly area and sends a Recce troop (one section). This troop detects some enemy and informs the combat group commander. Combat group then sends two troops for masking operation so that main armour may move swiftly to the objective. Simultaneously, the masking team keeps on engaging enemy in enemy zone until the main armour moves out of enemy range. This masking team thereafter re-joins the main armour and moves on towards the objective.

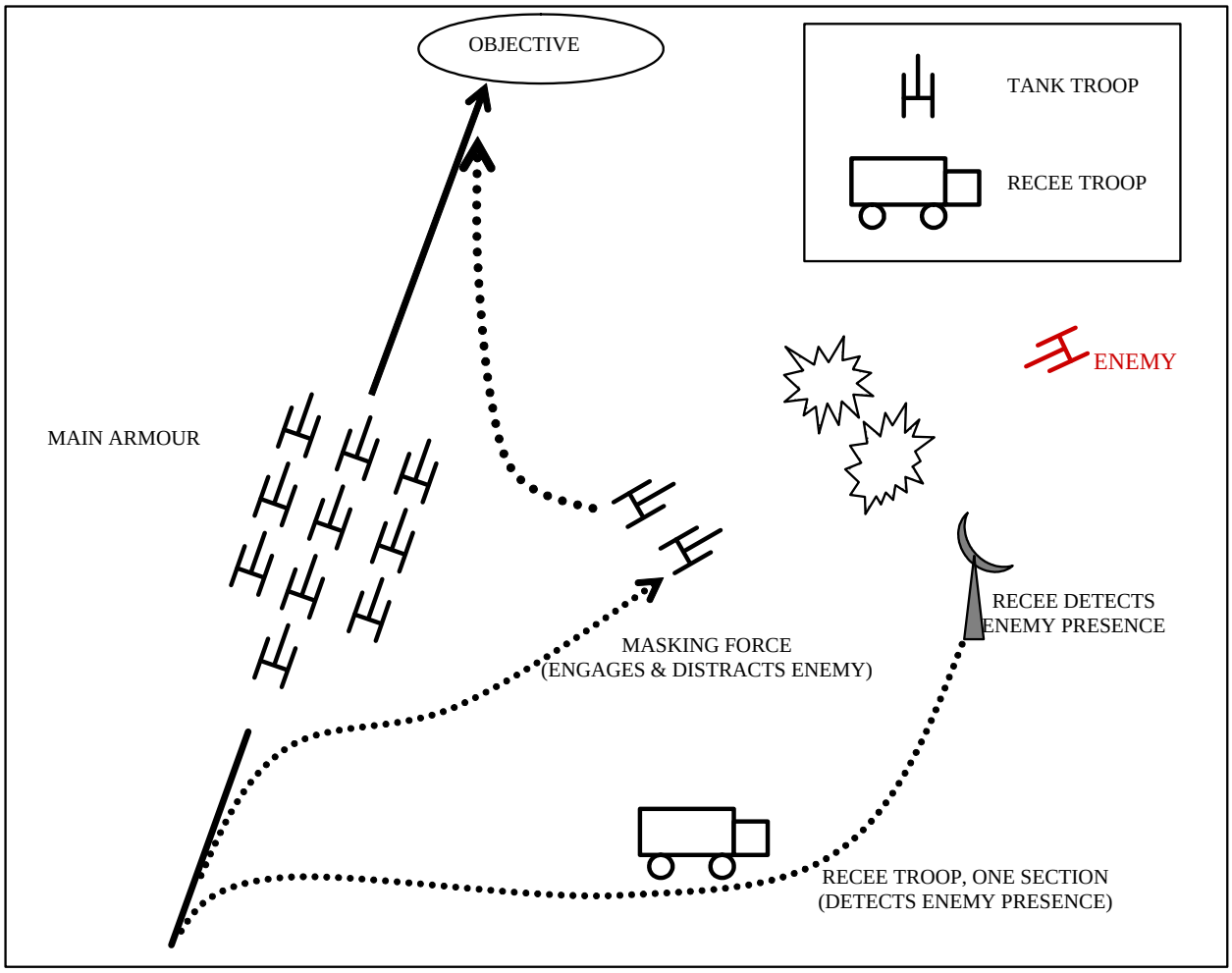

Figure 1. Masking scenario displaying agent-oriented team behaviour of a combat group.

To model and simulate this scenario using team-oriented concepts, first of all the key abstractions have to be identified. This will enable to clearly structure the team and define roles and responsibilities of the team members. From the textual narrative stated above, one can directly identify the team controller as the combat group commander, whose top-level goal is to move towards the assigned objective without any enemy interference. It is obvious that three sub-teams will be involved, namely: recce team, masking team, and main armour team, each performing its respective role by executing the appropriate plans. For example, recce team will handle recee events by having plans for detecting enemy location and informing team controller. Similarly, masking team will have plans for engaging the enemy so as to distract him. The masking team will join the main armour, when the enemy detection range between enemy and main armour is beyond reach or when the enemy has suffered more causality than desired threshold limit. The roles, responding events, and the corresponding plans for this scenario is given in Table 1.

Since team behaviour is modelled as an extension of agent concepts, two types of agents in this armour-masking scenario have been identified: the tank agents and the team agents that have all the capabilities of agents and also encapsulate team behaviour. The tank agents have been discussed in detail[2]. There, the authors had extended the agent class of JACK™ Intelligent Agents framework for representing the tactical and reactive behaviour of tanks which are low-level battlefield entities. Only

a single tank's tactical behaviour like movement along a route, obstacle avoidance, patrol, firing, etc, had been modelled. The authors have extended that work in this paper by modelling a team controller (representing the combat group) and three sub-teams. Block schematic is given in Fig. 2. In the listings given as *Appendix* 1, the authors have focussed on the events and plans relevant only to the team controller and recee team. Only at some places, the statements relevant to mask and armour sub-teams have been included, for the sake of completeness. Listing 1 (*Appendix* 1) gives the code of how the team controller extends the team class. It is written in the controller's team file (controller.team):

In this code segment, team members are declared using the construct '#requires role' which specifies the role requirements rather than explicit sub-team type requirements. The team controller requires a recee_role, mask_role and an armour_role. Once defined, the team members are referred to by their roles, which are associated with each sub-team during formation of a team. Each of the team member acts in coordination to achieve team goal, although they are individually responsible for determining how to satisfy those goals. This way, the functional requirements for the team members are expressed rather than restricting to particular team types.

Since, teams are defined in a way similar to agents in JACK, a team class is an extension of agent class taking on the additional responsibility for managing the coordination of each team member's activity. The team controller is also called the role

Table 1. Teams and their roles, events and plans

Main team	Sub-team (Role performer)	Roles	Responding events	Plans (event handlers)
Team controller	recee_team	RECEE_ ROLE	recee_event	recee_plan
	mask_team	MASK_ ROLE	mask_event	mask_plan
	armour_team	ARMOUR_ROLE	armour_event	armour_plan

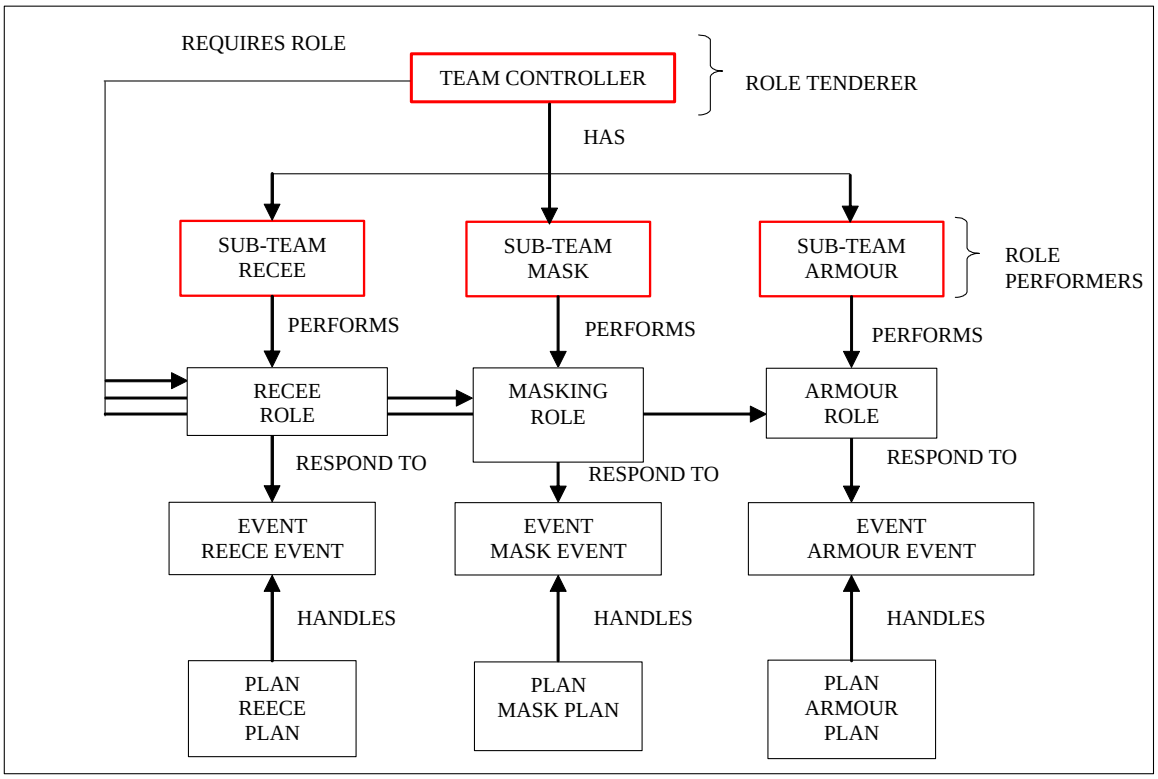

Figure 2. Block schematic of the team modelling of the armour masking scenario.

tenderer. It is composed of sub-teams that perform roles on its behalf. Role is a behaviour that the role tenderer may request the role performer to achieve. It represents behaviour of a team member (sub-team) participating in a particular tactical operation. As an example, consider a troop of tanks. Depending on the battlefield situation, that troop may perform the role of recee, masking, engagement or marching. When a troop performs a given role, it presents a particular view to its battlefield environment at that instance. The other entities that are interacting with it expect certain behaviour at that time, depending on the role that it plays at that time. For example, an instance of the troop in the role of masking would have a different set of capabilities than if that troop was playing the role of main armour heading for assigned objective. This concept of role performer is already part of UML[1], with reference to classes. In this example similar UML concepts have been applied to team modelling and represented diagrammatically in Fig. 3.

Continuing with our programming example, the key element of a team definition is the declaration of which roles the sub-teams can perform. In JACK, '#performs role' statement is used for this and is defined in the corresponding team file, as given in the code segment (listing 2, *Appendix* 1) of the recee team file (recee.team). Similarly, MASK_ROLE and ARMOUR_ROLE are defined in their respective team files as shown in listing 3 (*Appendix* 1).

The actual implementation of the role is specified in terms of events handled and posted by the entity that fulfils such a role within the team. The role construct is used for this purpose and the code segment in the corresponding role file (recee.role) is given in listing 4, Appendix 1. As also mentioned by G. Booch[1], *et al*., role is an interface definition, which declares what an entity that claims to implement a role must be capable of doing. This role definition has two parts. Firstly a 'downward' interface that declares the events an entity must handle to take on a role, and secondly an 'upward' interface that declares the events the team needs to handle when having a team member of the role.

The joint intention of the team is specified using the TeamPlan construct, which is implemented in the controllers plan file (controller.plan), as given in listing 5 (*Appendix* 1). The @team_achieve statement is used to activate a sub-team by sending an event to the sub-team. The team that sends the @team_achieve then waits until the sub-team has processed the event. It is obvious that the movement of main armour and the masking operation have to be executed in parallel. This is depicted in the activity diagram in Fig. 4. The @parallel construct of JACK teams enables this, as can be seen in the above code segment, which gets executed when the main armour team is out of range from the enemy's firing range and the masking team can now join the main armour. The @parallel allows

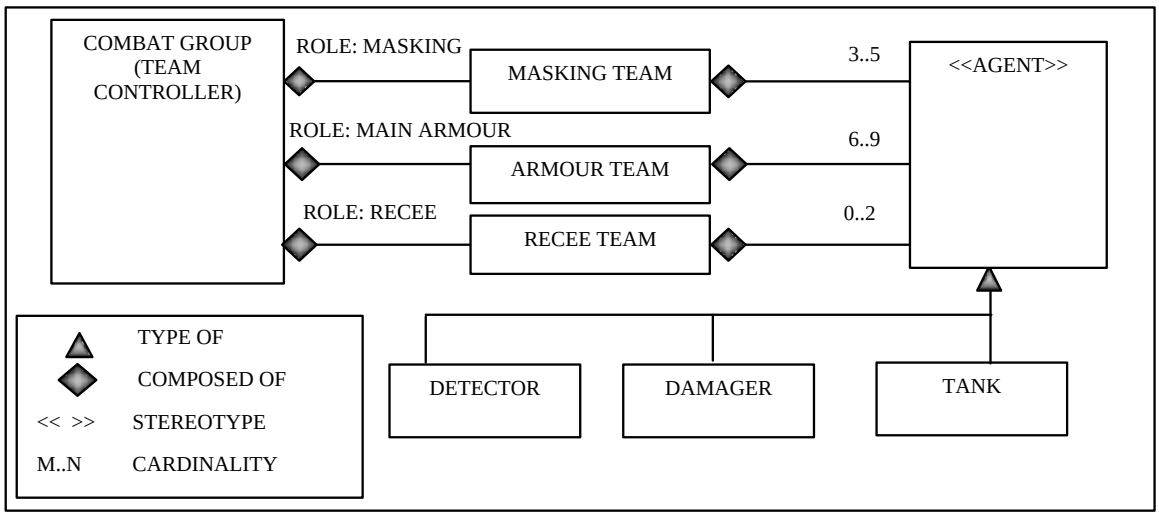

Figure 3. Key abstractions in the masking scenario.

concurrent sub-tasking of a set of statements within reasoning methods. The @parallel statement suspends execution of the calling plan while all enclosed statements are executed in parallel. The first parameter of the @parallel statement is the mode, which is ParallelFSM.ALL in this case. This means the @parallel statement will succeed after all the branches have succeeded, but fail immediately if any branch fails.

The teamplan of the recee team is given in the corresponding plan file (recee.plan), as shown in listing 6 (*Appendix* 1). Similarly, teamplan for mask team and armour team are defined in their respective plan files. The main method that initiates the team formation is given in listing 7 (*Appendix* 1). In this psudo-code, the formation of the controller team is achieved by attaching sub-teams capable of performing roles required by the team. This program also involves tank, detector, and damager agents[8].

4. CONCLUSIONS

In this paper, an attempt has been made to model team behaviour of tactical scenarios using JACK teams. Having an agent-oriented mind-set while modelling tactical scenarios enables one to map the key abstractions and entities involved into teams of agents, which is separate from simulation-specific code, hence encouraging separation of concerns and re-use. Further, the work breakdown in the military hierarchy is mapped directly into roles of team members. Hence, this allows for hierarchical decomposition of tasks. This approach allows the team-tactics of military operations to be captured and simulated with minimal effort, in contrast to the previous laborious construction of complex, scenario specific scripts involving multiple interdependencies between the entities. It encourages clear and concise description of coordinated activities and allows the abstraction of what needs to be done from how it is done, i.e., the responsibilities of the team can be written down without consideration of how the roles would be fulfilled and implemented by the team members.

This example illustrates how easily the textual narrative of a tactical scenario can be mapped and modelled using UML. It is also noted that roles and responsibilities are already well-established concepts in object-oriented analysis and design, and these have been linked up with team-oriented concepts. It shows how rapidly even relatively simple team programs become complex. For example, to implement this scenario, the authors had 4 agent files, 3 capability files, 6 team files, 4 role files, 37 plan files and 25 event files. However the resultant code was highly modular and maintainable, which would not have been possible otherwise.

This simple example has demonstrated that any complicated tactical scenario involving team behaviour can be modelled using the team concepts as discussed above.

5. LIMITATIONS AND FUTURE SCOPE OF WORK

5.1 Limitations

- Team modelling is a powerful scheme for specifying and implementing coordinated distributed behaviour. In this example, application, a limited but well-defined modelling capability has been obtained that can be implemented relatively easily. However, this model is applicable only to certain types of collaborations, which have rigid and well-defined team structures enabling a team to have predefined roles. This reduces the flexibility of the team. This work can be extended by dealing with dynamic team formation.
- Another shortcoming of this approach is that the action taken by the entities and the teams are based only on their current situations. The entities must be able to reorganise themselves into different teams or change their roles based on the changing tactical situations. This would also lead to dynamic changes in the intentions of the entities and teams. One can enhance UML constructs to cater for such dynamic situational changes and also explore the possibility of allowing different types of teams to collaborate.

5.2 Future Scope of Work

- Another desirable agent-team characteristic, missing in this approach is trust. Trust plays a fundamental role in such systems in which

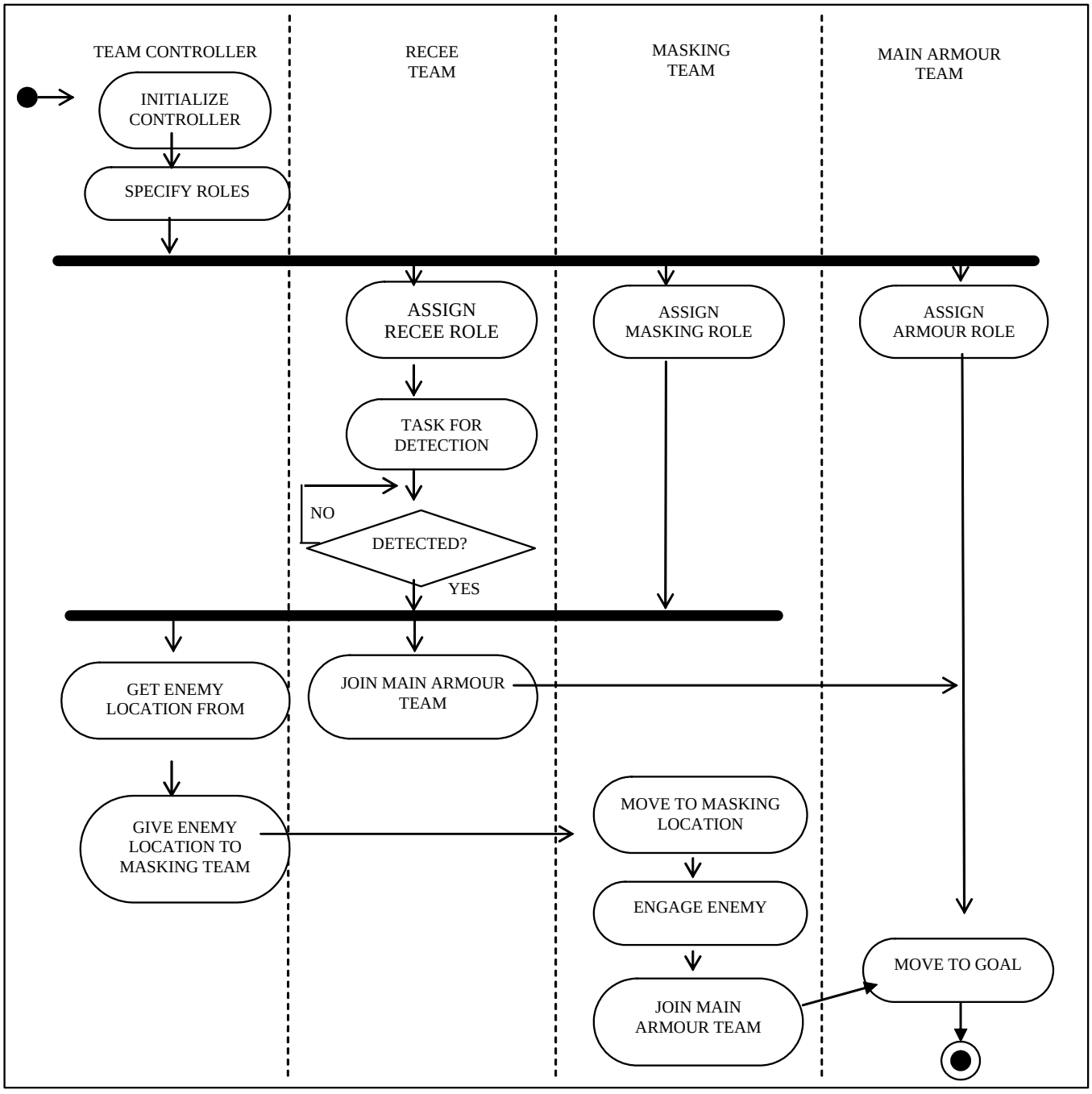

Figure 4. Activity diagram.

tasks are delegated. The concept of trust may be generalised and considered as a level of confidence in one's predictions of another agent's future behaviour. There exists vast potential for incorporating this important aspect of team behaviour into future tactical models.

ACKNOWLEDGEMENT

The authors thank Shri S.S. Prasad, Ex-Director, Institute for Systems Studies and Analyses, Delhi, for his constant encouragement and guidance.

REFERENCES

1. Booch, G.; Rumbaugh, J. & Jacobson, I. The unified modelling language user guide. Addison-Wesley, 1999.
2. Papasimeon, M. & Heinze, C. Extending the UML for designing JACK agents. *In* Proceedings of the Australian Software Engineering Conference (ASWEC01), Canberra, Australia, 2001. pp. 89-97.
3. d'Inverno, M. & Luck, M. Understanding agent systems. *In* Springer-Verlag Series on Agent Technology, 2001.
4. Ishida, T.; Jennings, N. & Sycara, K. *In* Springer-Verlag Series on Agent Technology, 2001.
5. Jennings, N.R. & Wooldridge, M.J. Agent technology, foundations, applications and markets. Springer-Verlag, 1998.
6. Luck, M. & d'Inverno, M. A Conceptual framework for agent definition and development. *The Computer Journal*, 2001, **44**(1).
7. Odell, J. Object and agents: Is there room for both? *Distributed Computing*, November 1999 44-45.
8. Malhotra, A.; Bisht, S. & Taneja, S.B. Using intelligent agents to simulate battle tank tactics. *In* Proceedings of the International Conference for Cognitive Science, New Delhi, 2004.
9. Rao, A.; Lucas, A. & Morley, D. Agent-oriented architecture for air combat simulation. Australian Artificial Intelligence Institute. Technical Note No. 42, 1993.
10. Hodgson, A.; Ronnquist, R. & Busetta, P. Specification of coordinated agent behaviour (The SimpleTeam approach). Agent oriented software Pty. Ltd. Technical Report 99-05, 1999.
11. Jarvis, J. JACK - Intelligent agents TM JACK, Teams Manual. Release 4.1, 2003.
12. Busetta, P.; Ronnquist, R.; Hodgson, A. & Lucas, A. JACK intelligent agents - components for intelligent agents in Java. Technical Report, Agent Oriented Software Pvt. Ltd., Melborne, Australia, December, 1999. (Also at www.agentlink.org as *AgentLink News letter*, January 1999, 2, 2-5)
13. Busetta, P.; Ronnquist, R.; Hodgson, A. & Lucas, A. Light-weight intelligent software agents in simulation. Technical Report 99-03. Agent Oriented Software Pvt. Ltd., Melborne, Australia, 1999.
14. Howden, N.; Ronnquist, R.; Hodgson, A. & Lucas, A. JACK intelligent agents™ - Summary of an agent infrastructure. Agent Oriented Software Pvt. Ltd., Melborne, Australia, 2003.
15. JACK™ Intelligent agents evaluation version. http://www.agent-software.com
16. Tidhar, G.; Heinze, C.; Goss, S.; Murray, G.; Appla, D. & Llyod, I. Using intelligent agents in military simulation or using agents intelligently. DSTO, Fishermen's Bend, Victoria, Australia, 2000.
17. Rao, A.S. & Georgeff, M.P. BDI agents: From theory to practice. Technical Note 56, Australian Artificial Intelligence Institute. [Also *In* Proceedings of the Ist International Conference on Multi-Agent Systems, ICMAS-95, San Francisco, USA, 1995. pp. 312-19].
18. Winikoff, M.; Padgham, L. & Harland, J. Simplifying the development of intelligent agents. Technical Report TR-01-3, School of Computer Science and Information Technology, RMIT University. [Also *In* Proceedings of the 14th Australian Joint Conference on Artificial Intelligence, AI'01, Adelaide, 2001].
19. Cheryl, B.; Cleotilde, G.; Mike S. & Haydee, C. Modelling shared situational awareness. *In* Conference on Behaviour Representation in Modelling and Simulation, 2005 (BRIMS05), Simulation Interoperability Standards Organisation (SISO), www.sisostds.org.

Appendix 1

Inset 1: JACK Teams concepts

The team concept encapsulates coordinated activity and extends the agent concept by associating tasks with roles. Each team is a distinct entity, which is characterised by the roles it performs and roles it requires others to play and attaches sub-teams capable of doing different roles required by containing team. The new concepts added to the basic JACK paradigm provide a consistent scheme for specifying coordinated team behaviour between the various components of the team.

Team: A *team* is a distinct reasoning entity, which is characterised by the roles it performs, and/or the roles it requires others to perform.

Role: A *role* is a distinct entity, which defines a relationship between teams and sub-teams. Roles contain a description of facilities that the participants in team / sub team must provide.

Teamdata: This allows propagation of beliefs from team to sub-team and vice-versa.

Teamplan: This specifies how a team will achieve a task in terms of one or more roles. It represents the activity of a group of sub-teams or agents in order to achieve a team goal. It also dictates the steps directing each sub-team to achieve specific goals.

Inset 2: Tactical Terms

Armour Regiment: It is a unit consisting of 45 tanks, and approximately 600 personnel. It is organised into three fighting squadrons and one administrative squadron. Each fighting squadron consists of a squadron headquarter and four fighting troop.

Combat Group: It is a grouping of all arms based on an armoured regiment or a mechanised infantry battalion. The aim of the grouping is to create a composite force capable of carrying out the mission assigned to it. Its composition will be affected by the type of terrain & obstacles, the mission, anticipated enemy opposition, availability of own troops and time of execution. The combat group normally forms a number of combat teams.

Combat Team: A combined team of an armour squadron and a mechanised company. A combat team is normally a sub-team of a combat group.

Forward Assembly Area: In case distance between Assemble Area and objective is excessive, a forward assembly area may be interposed for purposes of regaining control/halting during daylight hours.

Masking: Masking is the act of covering the enemy force, so that it cannot effectively interfere against our own forces' offensive missions. Masking therefore may be a prelude to bypassing or capturing of an enemy position. In this case the neighbouring position to the one being captured may need masking, so that the operations being conducted are not interfered or assisted by the position being masked.

Outflanking: The movement of a force onto an enemy flank (side) or rear without penetrating his position.

Reconnaissance: It is the process of obtaining information by direct examination of an area of ground.

Reconnaissance Troop (recee troop): The primary task of the reconnaissance troop is battle reconnaissance. Reconnaissance detachments from the reconnaissance section of the Combat Team will be deployed along the likely approach of enemy reaction.

Listing 1

```
public team CONTROLLER extends Team {

   #requires role RECEE_ROLE role_recee;
   #requires role MASK_ROLE role_mask;
   #requires role ARMOUR_ROLE role_armour;

 #handles event Start;     #uses plan ControllerPlan;

   // Used for peer-coordination of Recee, Mask & Main armour team
   #handles event recee_event;     #uses plan CoordinateR;

   // Used when reccee team sends detection information to controller
   #handles event info_detection_to_cont;
   #uses plan info_detection_to_cont_plan;

  // Used when reccee team sends no detection information to controller
   #handles event info_rec_goal_to_cont;
   #uses plan info_rec_goal_to_cont_plan;
      .
   // similar "event-plan" pairs are made for communicating with mask and
   // armour sub-teams
      .
   #posts event Start s; // posts start event to self

   // Attributes of the Controller Team relevant to the recee sub-team
   public String name_controller;
   int receegoalx,receegoaly, armour_goalx, armour_goaly;
   int enemy_x,enemy_y, detection_counter, recee_goal_acheived;

   // constructor
   public CONTROLLER(String name_cont)
   {
    super(name_cont);
    name_controller = name_cont;
    armour_goalx= armour_goaly= enemy_x= enemy_y = 0;
    detection_counter= recee_goal_achieved=0;
   }

   public void initialize_controller()
    {
     postEvent(s.start(receegoalx,receegoaly));
   }

  // Set Recee goal
  public set_recee_goal(int x, int y)
  {
   receegoalx = x;
   receegoaly = y;
  }

  // Used for getting enemy_location from Recee Team
  public void get_enemy(int detect,int x,int y)
  {
   detection_counter=detect;
   enemy_x=x;
   enemy_y=y;
   return;
  }
  // Used for getting objective location
  public void set_goal(Point obj_location)
   {
   armour_goalx= obj_location.x;
   armour_goaly= obj_location.y;
   return;
   }
  // Set enemy location
  public void set_enemy_cur_loc(Point enemy_location)
   {
    enemy_x= enemy_location.x;
    enemy_y= enemy_location.y;
    return;
   }
}
```

Listing 2

```
public team R_TEAM extends Team
  {
    #performs role RECEE_ROLE;

    // Initiates recee team plans
    #handles event recee_event;    #uses plan recee_plan;

    // Events needed to communicate with the team controller
    #sends event info_detection_to_cont;
    #sends event info_rec_goal_to_cont;

    // Attributes of the Recee Team
     String recee_team_name;
     public String TeamController;
     public int detection_counter;
     int recee_goal_achieved, enemy_x, enemy_y;

    // constructor of recee team
    public R_TEAM(String rteam_name)
    {
     super(rteam_name);
      recee_team_name= rteam_name;
      detection_counter= enemy_x = enemy_y= recee_goal_achieved= 0;
    }

   // method for getting the name of the team controller
   public void set_controllername( String name_cont)
   {
    TeamController = name_cont;     return;
   }
}
```

Listing 3

```
public team R_TEAM extends Team
{
#performs role MASK_ROLE;
;
}
public team A_TEAM extends Team
{
#performs role ARMOUR_ROLE;
;
}
```

Listing 4

```
role RECEE_ROLE extends Role
  {
    #handles event recee_event handle_recee_event;
    #posts event recee_event post_recee_event;
    // Team location variables
  }
```

Listing 5

```
teamplan ControllerPlan extends TeamPlan  {
    #handles event Start ev;
    #uses role RECEE_ROLE role_recee;
    #uses role MASK_ROLE role_mask;
    #uses role ARMOUR_ROLE role_armour;
    #uses agent implementing CONTROLLER CONT;
      body()
    {
     // initially, only the reccee team goes for detection.
      //The other two teams are stationary
    while(CONT.detection_counter==0 )
    {
      if(CONT.recee_goal_acheived==1)
          break; // stop receeing
            // tasking recee team to go for recee

      @team_achieve(role_recee,role_recee.handle_recee_event.start(ev.goal_x,ev.goa
      l_y,1));
    }
     // this code executes when the reccee team detects some enemy.
     // It simultaneously assigns relevant tasks to the mask and armour sub-teams
      if(CONT.detection_counter > 0 )
       {
        @parallel(ParallelFSM.ALL,false,null)
         {
           // assigning armour goal to recee team
           @team_achieve(role_recee,role_recee.handle_recee_event.start(CONT.armo
ur_goalx,CONT.armour_goaly,0));
           // assigning armour goal to armour team
           @team_achieve(role_armour,role_armour.handle_armour_event.start(CONT.
armour_goalx,CONT.armour_goaly,0));
           // tasking the mask team to engage detected enemy
           @team_achieve(role_mask,role_mask.handle_mask_event.start(CONT.enem
y_x,CONT.enemy_y,1,0));
         };//@parallel
        }//if


// this code executes when the armour is out of range from the enemy's firing range.
The masking team can now join the main armour

if(CONT.recee_goal_acheived==1)
 {
    @parallel(ParallelFSM.ALL,false,null)
      {
          // assigning armour goal to recee team
          @team_achieve(role_recee,role_recee.handle_recee_event.start(CONT.armou
r_goalx,CONT.armour_goaly,0));
          // assigning armour goal to armour team
          @team_achieve(role_armour,role_armour.handle_armour_event.start(CONT.
armour_goalx,CONT.armour_goaly,0));
          // tasking masking team to join main armour team
          @team_achieve(role_mask,role_mask.handle_mask_event.start(CONT.armou
r_goalx,CONT.armour_goaly,0,0));
      }
  };
 } // body
}
```

Listing 6

```
teamplan recee_plan extends TeamPlan {
   #handles event recee_event ev;
   #applicable_for role RECEE_ROLE self;
   #sends event info_detection_to_cont detect;
   #sends event info_rec_goal_to_cont goalinfo;
   #uses agent implementing R_TEAM Team1;
   #sends event start_sim_from_team start_sim;

   body()
   {
     . . .
    //code to implement the objective that this team wants to
achieve
   @team_achieve(. . . );
   }
 }
```

Listing 7

```
//The main method (executed through GUI), that instantiates the teams
  initialise_teams(Point armour_goal, Point recee_goal)
  {
    // Creation of sub-teams
    // initiate recee_team with name RECEE_TEAM
    R_TEAM recee_team = new R_TEAM("RECEE_TEAM" );

    // initiate mask_team with name MASK_TEAM
    M_TEAM mask_team = new M_TEAM("MASK_TEAM " );

    // initiate armour_team with name ARMOUR_TEAM
    A_TEAM armour_team = new A_TEAM("ARMOUR_TEAM " );

    // Object of the main team controller team instantiated
    CONTROLLER cc_r = new CONTROLLER("TEAM_CONTROLLER_RED" );

    // Attach the three sub-teams with the main controller team
    // role “role_recee” is attached to the team RECEE_TEAM
    cc_r.attachRole( "role_recee", "RECEE_TEAM" );

    // role “role_mask” is attached to the team MASK_TEAM
    cc_r.attachRole("role_mask", "MASK_TEAM" );

    // role “role_armour” is attached to the team ARMOUR_TEAM
    cc_r.attachRole("role_armour", "ARMOUR_TEAM" );

   // Function called by sub-teams for getting the name of their team controller
    recee_team.set_controllername(cc_r.name_controller);
    mask_team.set_controllername(cc_r.name_controller);
    armour_team.set_controllername(cc_r.name_controller);

    // Supply goals to the sub-teams
    cc_r.set_goal(armour_goal);
    cc_r.set_enemy_cur_loc(recee_goal);
    set_recee_goal(recee_goal)

    // Initiates execution of the first controller plan
    cc_r.initiate_controller();
  }
```

Contributors

Mr Sanjay Bisht obtained his MSc (Computer Science) from the DAVV University, Indore. Presently, he is working as Scientist C at the Institute for Systems Studies and Analyses (ISSA), Delhi. His areas of interest are: Heuristic optimisation, genetic algorithm, simulated annealing, and agent technology.

Ms Aparna Malhotra obtained her MSc (Computer Science) from Devi Ahilya Vishwavidyalaya, Indore, in 1992. She joined as a Scientist in ISSA, in1992. Her areas of research are naval wargaming, soft computing techniques, and intelligent agent technology.

Mr S.B. Taneja obtained his MSc (Electronics) from Delhi University; Electronics Fellowship Course from Institute of Armament Technology (now DIAT), Pune; and MTech (Computer Science) from IIT Roorkee. Presently, he is working as Scientist F at ISSA, Delhi, and is involved in design and development of wargame simulation systems. His areas of interest are: Distributed simulation, wargame simulation system design, GIS, and agent-based models for warfare simulation.