Modeling Command & Control Centers

Cagatay Undeger, Askin Ercetin & Ziya Ipekkan
Scientific Decision Support Center
Turkish General Staff, HQ
06650 Ankara,
Turkey

E-mail: cundeger@ceng.metu.edu.tr / aercetin@tsk.mil.tr / zipekkan@tsk.mil.tr

ABSTRACT

To date, legacy simulations of all operation levels have not dealt with the C4ISR aspects of the battle space. Nearly all of these simulations assumed either perfect C4ISR capability on both sides or employed some unjustified approaches to take C4ISR capabilities of the opposing forces into account. These approaches to modeling C4ISR did not make it possible to evaluate C4ISR systems and their contribution to the mission effectiveness.

Currently, especially developed countries are well aware of the potential contributions of robust and integrated C4ISR systems to overall military effectiveness and working on concepts like Information Warfare, Network Centric Warfare and etc. Most of these countries are spending large portions of their budgets on procuring / developing C4ISR systems. C4ISR systems are inherently very complex and as a result it is very hard, if not possible, to develop architecture, concept and tactics with pure analytical approaches. At this point, simulation seems to be the most suitable candidate for this kind of C4ISR analysis.

This paper will present a detailed description of the Command, Control, Communication and Computer Analysis Tool (C4AT) that is currently being developed by the Turkish General Staff Scientific Decision Support Center. When the first version is completed, the tool will be capable of simulating peace time activities of the strategic and operational level command and control centers. The second version will also have the ability to simulate and analyze crisis and conflict time activities of the similar command and control centers.

1.0 INTRODUCTION

The study of complex systems that have many actors and their interactions often becomes too complex for a mathematical model. Agent-based modeling is a tool to study these kind of systems. The tricky part of this modeling tool is to specify the environment, agent-knowledge model and the interactions between the agents.

A software agent can be defined as any type of software entity that fulfills the basic concepts of agency. Ferber defines the properties of a software agent as follows [Ferber 1999] :

- An agent is capable of acting and modifying its environment
- An agent can communicate with other agents in the environment
- An agent has intentions
- An agent controls some local resources

Paper presented at the RTO MSG Symposium on "C3I and Modelling and Simulation Interoperability", held in Antalya, Turkey, 9-10 October 2003, and published in PUB-NBR.

- An agent is capable of perceiving its environment to a limited extent
- An agent has only a partial representation of its environment
- An agent possesses skills and can offer services

The need for simulating a group of agents in an environment leads to the development of Multi-Agent Systems (MAS). Weiss gives the following characteristics of multi-agent environments [Weiss 1999]: They provide a basis for specifying interaction and communication protocols; they are mostly open and have no centralized designer; they contain autonomous and distributed agents that may be cooperative or self-interested. Instead of defining MAS characteristics, Ferber reports elements that comprise a MAS. These elements are environment, objects, agents, relations, operations, and operators. Environment is a space in which every object of the MAS resides. Everything in the environment is an object. An agent is also an object in the environment that satisfies agency requirements. Relations link objects to each other in the environment. Operations are the actions that agents can perform in order to modify the environment and to achieve their goals. Operators can be described as the laws of the environment. Operators are basically the reactions of the environment to the actions taken by agents. Constructing a MAS requires detailed models of these elements.

MAS simulation is a new solution to the problem of imitating complex adaptive systems. Axelrod describes MAS simulation as “a way of doing thought experiments,” the goal of which is to enrich our understanding of fundamental systems [Axelrod 1997]. He contents that the goal of MAS simulation is not to find exact solutions to real world problems, but rather to provide insight into complex systems that conventional approaches cannot model. Therefore, modeling every aspect of the system is unnecessary. Axelrod proposes the famous army slogan, “Keep it simple, stupid” to the MAS simulation designers. Otherwise, the change in the outcome of the simulation cannot be linked to any particular variant in the simulation and hence makes simulation useless. However, one should also be very careful in deciding which aspect of the real world should not be included in the simulation. Omitting a key component of a system from the simulation may result in meaningless, undesired outcomes.

command and control centers are complex organizations in nature. They contain many co-operating actors (agents), each having different personalities, roles, and goals. Within these organizations, many time critical processes take place in parallel which makes it very hard to keep track of the interactions between the agents. These interactions are also highly depended on the work load, which is determined by the outer organizations. With these properties, any command and control center can be thought as a multi agent system.

Since it is too hard, if not impossible, to model such a complex system with conventional modeling and simulation techniques, agent-based approach has been chosen to study command and control centers, and a generic agent-based simulation engine is implemented.

Section 2 describes the generic simulation engine and the methodology for creating a new project. Section 3 deals with the communication devices and their working principles. Section 4 introduces the environment design. Section 5 presents detailed information on the agent architecture. Section 6 is intended to demonstrate a scenario in execution. Finally, Section 7 gives a summary of the study and its results.

2.0 THE SIMULATION ENGINE

In order to implement our agent-based tool for simulating command and control centers (ComConCent), first a generic simulation engine independent of the tool was developed. The generic simulation engine is actually a single software class (*TBtnEngine*), which is used as a base class and inherited in order to create

our analysis tool, C4AT. The functions of the *TBtnEngine* is as follows:

- Enabling creation and modification of the project environment and the scenario,
- Managing environment and scenario files (save and load operations)
- Defining scenario agents incuding their behavioral characteristics and tasks based on the developed High Level Task Management Script (HLTMS) and Behavioral Transition Networks (BTN) architecture.
- Accepting insertion of any project specific resources (properties, methods, classes, data, etc.)
- Allowing agents to access all the resources of the project through a C-like run-time interpreter developed within the scope of the study.
- Running scenario with a selected time management mode (event based, real-time, and constant time interval).

As mentioned above, TBtnEngine can be customized using object inheritance, thus new projects can be created by just over-writing the bold lined functions depicted in Figure 1. *TBtnEngine* controls the whole simulation activities through these functions. For instance, when a new agent needs to be created, the engine calls the *CreateAgentProc* function (pointer) to let creation of a project specific agent instance.

```
class TBtnEngine

            TBtnEngine        ( TObjectBaseClass * includes, TCreateAgent createAgentProc );
    virtual ~ TBtnEngine      ( );

    TCreateAgent CreateAgentProc;

    virtual void SaveToStream   ( TStream * stream );
    virtual void LoadFromStream ( TStream * stream );

    virtual void SimStarted     ( );
    virtual void SimStopped     ( );
    virtual void SimAdvanced    ( double simTime, double deltaTime );
    virtual bool InLOS          ( TAgentBaseClass * agent, TAgentBaseClass * target );

    void SaveToFile             ( AnsiString filename );
    void LoadFromFile           ( AnsiString filename );

    void Start                  ( );
    void Pause                  ( );
    void Stop                   ( );
    void Advance                ( );

    TAgentBaseClass * AddAgent  ( );
    TAgentBaseClass * FindAgent ( AnsiString name, int * index = NULL );
    ...
    ...
```

Figure 1. TBtnEngine Class Interface: new projects can be created by customizing the bold lined functions

The critical point in customizing the engine is to use *TObjectBaseClass* as a base class to all the new classes. This base class enables registering any variables and functions in real time to let the run-time interpreter access project resources. To extend the flexibilty of customation, *TBtnEngine* and all of its sub-classes are also inherited from this class. With the customized engine, C4AT, the geographical location of the scenario can be set, the ComConCent buildings including their interior can be designed, the communication devices can be introduced and the agents can be defined. The C4AT architecture and its class definition are shown in Figures 2 and 3 respectively.

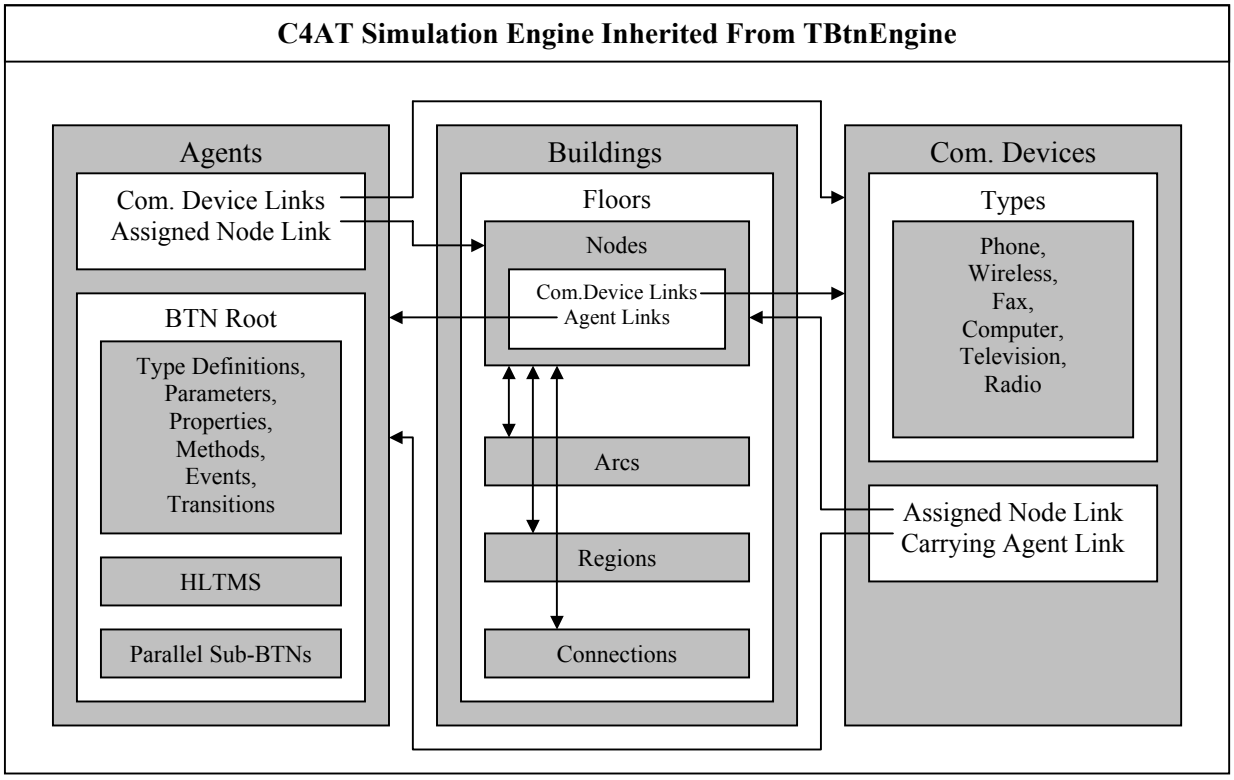

Figure 2. The general architecture of the C4AT

```
class TC4ATEngine : public TBtnEngine

      TComDevices  * ComDevices;
      TBuildingSet * BuildingSet;

               TC4ATEngine       ( TObjectBaseClass * includes, TCreateAgent createAgentProc );
      virtual ~ TC4ATEngine      ( );

      virtual void SaveToStream   ( TStream * stream );
      virtual void LoadFromStream ( TStream * stream );
```

Figure 3. C4AT Simulation Engine Class : The bold lined objects are extension of the project, and the rest are functions modified in order to customize the engine

3.0 DEFINING THE COMMUNICATION DEVICES

The communication devices that are employed in command and control centers can be categorized as: phone, radio, fax, computer, and multimedia (television, radio, newspaper, etc.). Since, speed has generally higher priority than information security in peace time operations, the phone is the most preferred communication device in such operations.

Consequently, we started the development of communication devices with the phone. For the time being, the first version of the telephone communication framework is completed. In this framework, the phone is designed to be in one of the following states: available, waiting for dial tone, ready to dial, calling, connected, busy, disconnected or ringing.

The agents are designed in such a way that they can detect if a phone is in-use, if not, they can pick up the

phone, wait for the dial tone, dial the number and take different actions depending on the communication state (calling, connected, busy or disconnected) and finally hang up the phone.

An agent can only use three kinds of phones categorized by their ownership: the ones he carries on (e.g. cellular phones), the ones he is responsible for (e.g. at home, at the office), and the ones that are owned by his group members. When a phone rings, the agents around the phone can hear the ring. If it is one of the phones he carries on or he is responsible for, he immediately tries to pick up the phone. If it is owned by one of his group members, then he waits for a short period of time and if no one picks it up, he tries to do it, and else no reaction is performed.

4.0 DESIGNING THE ENVIRONMENT

The simulation environment consists of a set of buildings and their interior. The location of each building is described by its geographical coordinate in lattitude and longitude. Following the definition of the location, the building images are automatically displayed on the screen if corresponding data exists in the environment database. A sample building exterior is illustrated in Figure 4.

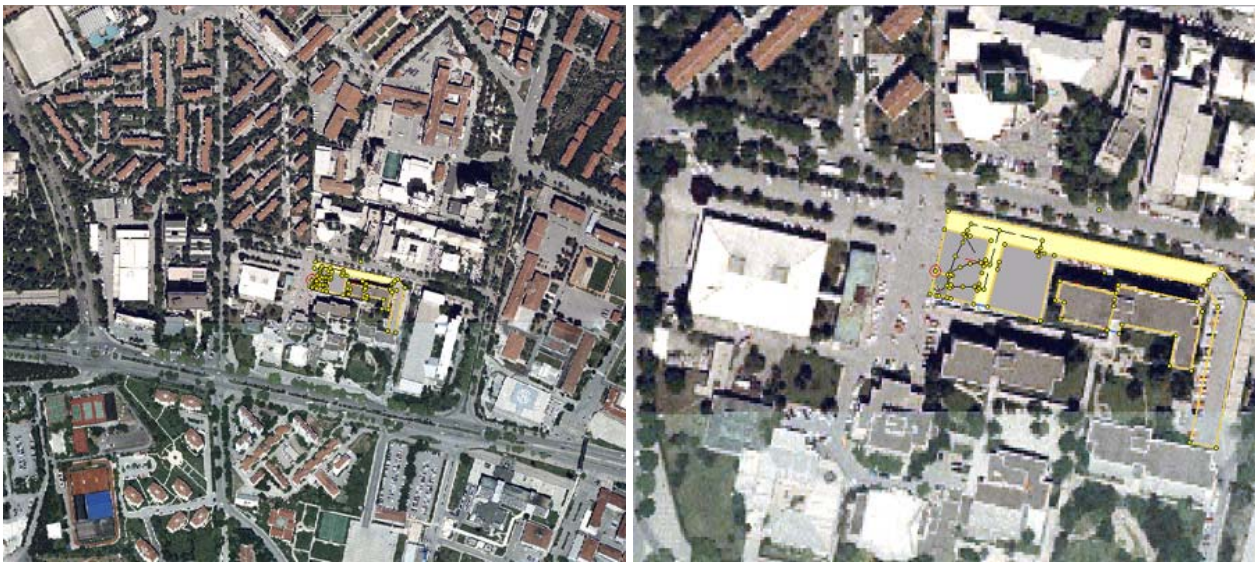

Figure 4. A building exterior visualized with two different zoom levels

In order to create building models, a building editor is developed. This editor is capable of designing buildings by creating each floor and their connections with other floors. Floors contain nodes (connectors, tables, chairs, etc.), arcs (walls, windows, doors, etc.), regions (room floors, roofs, etc.) and connections (walking routes, relational links, etc.). A sample building interior design is demonstrated in Figure 5. Every object in the environment is positioned on one of the nodes. Connections are used for defining possible routes that could be used for navigation and specifying relations between objects. For example, when an agent percepts that a phone is ringing, he first tries to go to a neighbor node of the table on which the phone is located by searching the connections to the table.

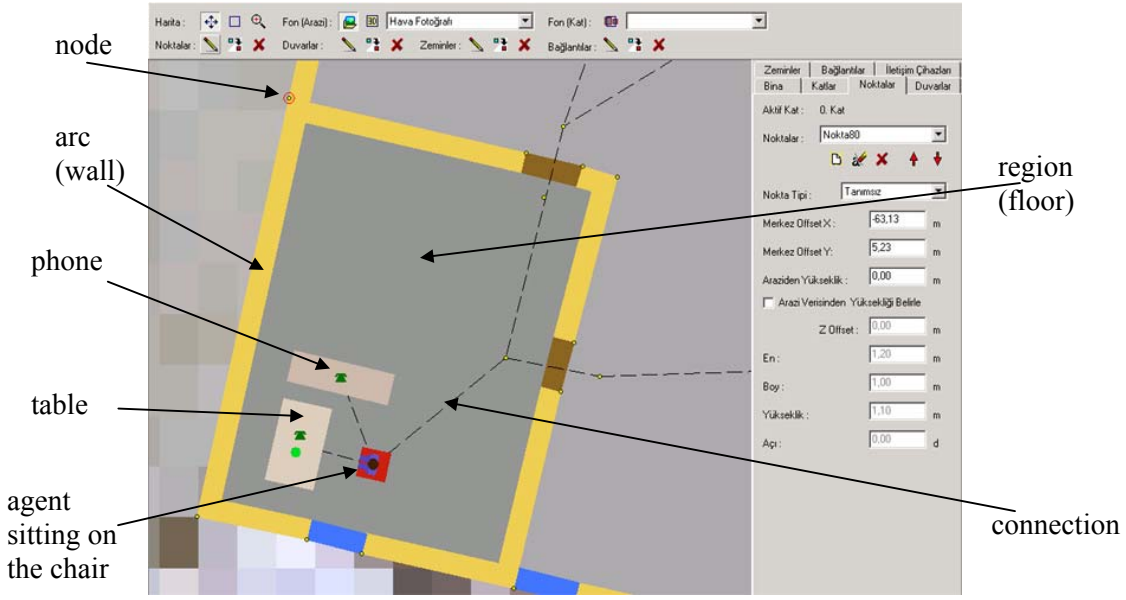

Figure 5. A building interior : The buildings are designed as set of floors.

5.0 DEFINING THE AGENTS

The behaviors of the agents are modeled using Behavioral Transition Networks (BTN) and a sub-feature defined within BTN structure called High Level Task Management Script (HLTMS). BTNs [Houlette 2000] are firstly introduced by game developers to increase practical aspects of defining behaviors. Later on, they became more common and used in many other fields. In fact, BTNs are just a specialized approach based on State Transition Diagrams and Harel Diagrams (statecharts) [Harel 1988, Ghezzi 1991, Rosenblum 1994, Budgen 1994], and defined using nodes and state transitions between nodes. Nodes may possess sub-nodes, resulting a type of hierarchy.

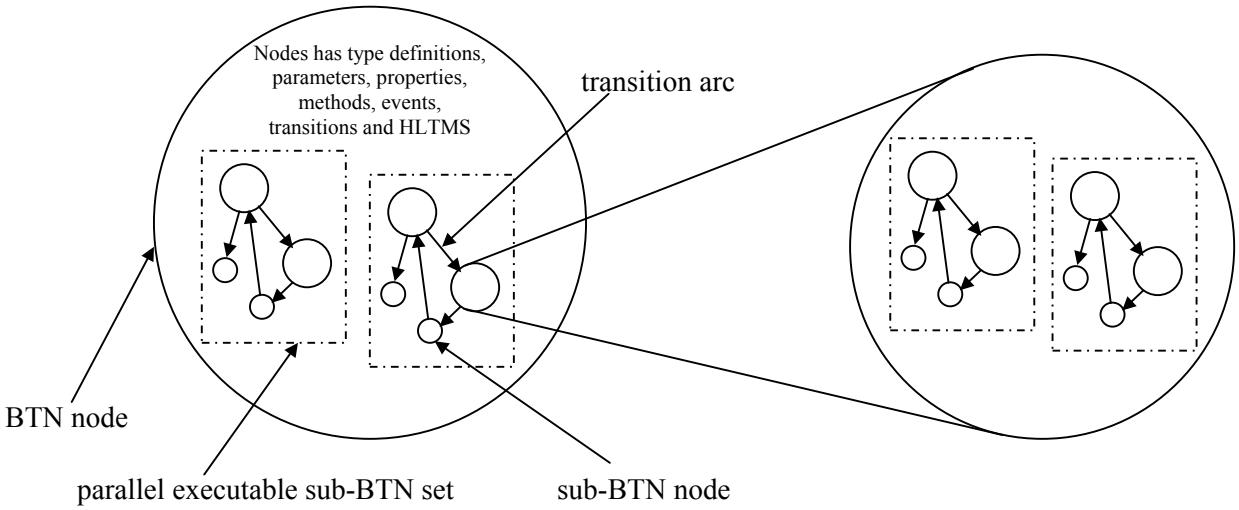

Figure 6. General Architecture of a BTN

A modified version of BTN framework is developed for *TBtnEngine*. This framework has some additional features such as parallel executable sub-BTNs and a HLTMS. General architecture of our BTN framework

is illustrated in Figure 6.

Each BTN node has its type definitions, in-going transition parameters, out-going transitions (in a hierarchical case based structure), properties, methods, events, and a HLTMS. BTN nodes are activated, executed and deactivated by transitions, HLTMS and events: *on enter*, *on leave*, *on message* and *on process*. Each event is a script which is executed by the run-time interpreter. All the events but *on process,* are executed from start to end at once. Execution of *on process* events can be interrupted by using *wait* or *breakcode* statements defined in the script.

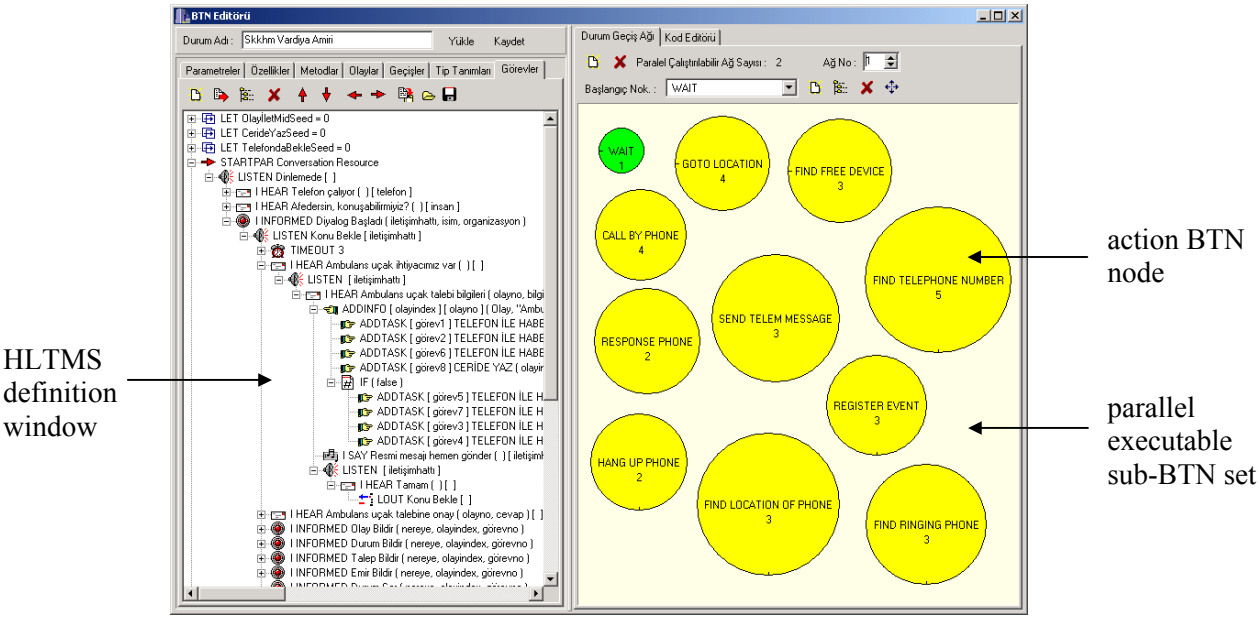

Figure 7. A root BTN sample for agent behavior modeling

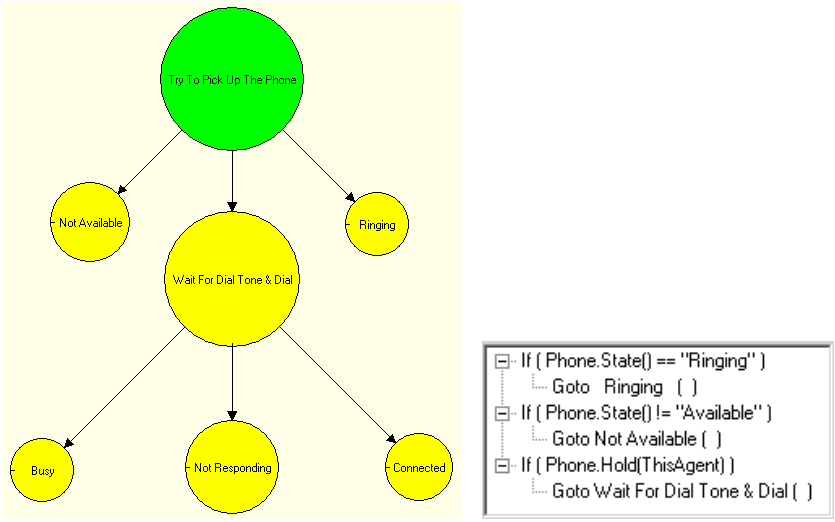

Figure 8. "CALL BY PHONE" action BTN (left), transitions of "Try To Pick Up The Phone" (right)

Each agent has a set of behaviors assigned to him, which are defined in a single BTN node (root BTN). Therefore, customizing that BTN node (defining HLTMS, adding sub-BTNs, etc.) also changes the

behavioral characteristics of the agent. In our agent design, first level BTNs, the ones directly owned by root BTN, are generally used for action modeling such as calling by phone, going to a location, etc. A root BTN and an action BTN are demonstrated in Figures 7 and 8 respectively. As seen in Figure 7, there are no transition arcs between action nodes, because actions are fired by HLTMS.

HLTMS is actually a hierarchically defined script, the statements of which can be executed sequentially or in parallel. Some of HLTMS statements and their descriptions can be viewed in Table 1.

Table 1. Some of the HLTMS statements

Statement	Description
STARTPAR	Starts a parallel execution (starts executing sub-script under STARTPAR in parallel)
CANCELPAR	Terminates a parallel execution
WAIT	Waits for a specified time period
LET	Assigns a value to a variable. If variable is not defined, it is created
ADDINFO	Inserts an information/data item into BTN node
ADDINFORM	Inserts an information message into a parallel execution
ADDTASK	Inserts a task into a parallel execution
REMOVETASK	Removes the active task of a parallel execution
DELAYTASK	Delays execution of the active task of a parallel execution
I SAY	Sends a voice message to an agent or object (phone)
I DO	Triggers an action
DORESULT	Captures the result of an action
LISTEN	Starts listening for perception messages, information messages and tasks
I HEAR	Used inside a LISTEN statement. Enables receiving voice perception messages
I INFORMED	Used inside a LISTEN statement. Enables receiving information messages
I SELECT	Used inside a LISTEN statement. Enables selecting from tasks. The task with highest priority is always selected.
LOUT	Backtracks to the start of a specified LISTEN statement

An example HLTMS for responding a phone call is illustrated in Figure 9. In the sample, the execution is started from the very first line. When a *STARTPAR* statement is reached, a parallel execution for the sub-statements called “Conversation Manager” is started. After that, the execution continues and reaches to another *STARTPAR*, which starts another parallel execution called “Task Manager”. And finally the execution reaches to a *LISTEN* block that contains a single *WAIT* statement, which causes an infinite loop. When “Conversation Manager” is executed, it starts listening for voice messages. If a phone rings, the agent gets a voice message informing him that the phone is ringing. This causes the addition of a “Respond Phone” task to the parallel execution called “Task Manager”. When “Task Manager” is executed, it starts listening for tasks. When it encounters a “Respond Phone” task, it starts executing the sub-statements of the task. The agent first identifies the phone to see if he has right to pick it up. If so, he finds the location of the phone, goes to that location and picks up the phone. If the connection is established, the agent says hello to the remote side and waits for a reply. If he gets the reply, he insert an information message to “Conversation Manager” to inform the starting of the conversation and waits until the phone conversation is terminated. Then, the “Conversation Manager” captures the information message and guides the conversation. For simplicity, the conversation part is skipped. After the termination of conversation, the conversation manager inserts an information message to “Task Manager” informing that the conversation is over. In this case the “Task Manager” hangs up the phone and starts waiting for another task.

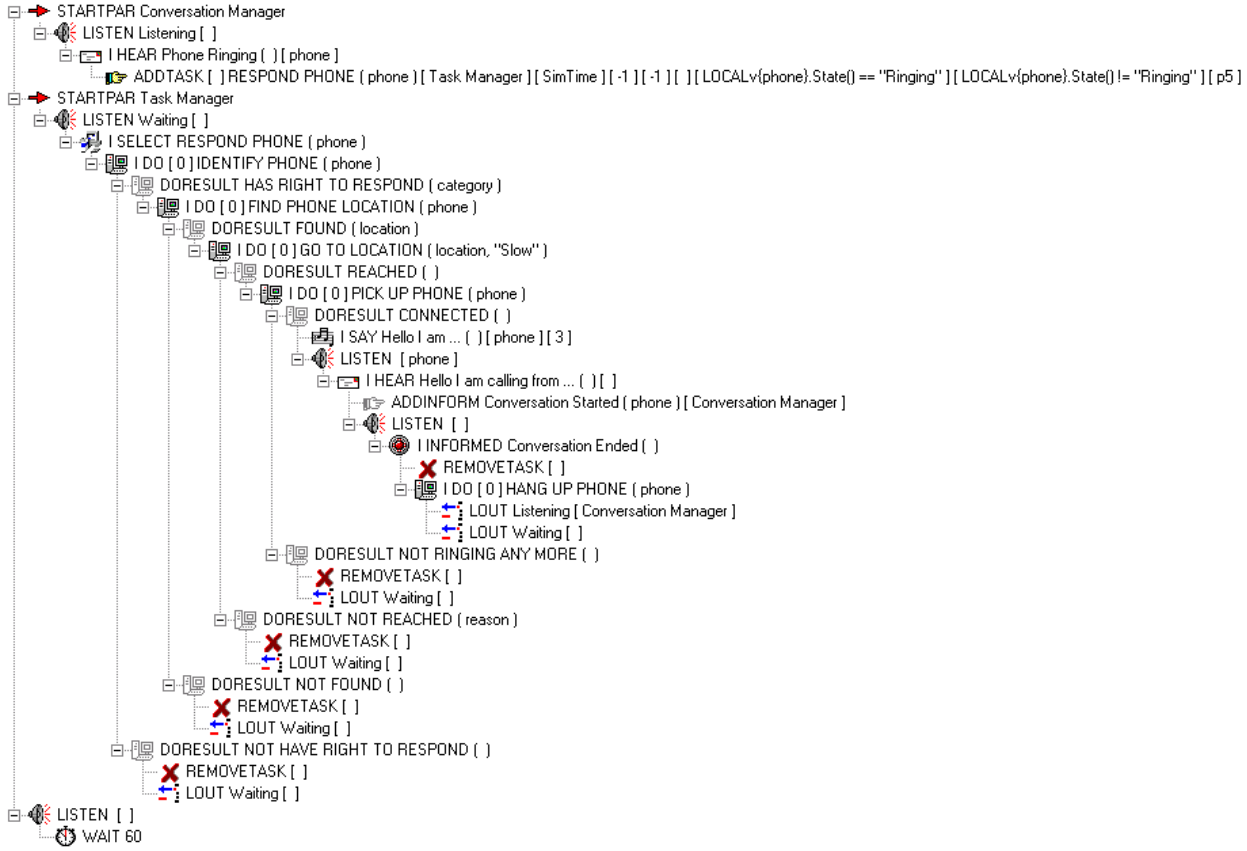

Figure 9. A sample HLTMS for responding a phone call

In order to avoid implementing complex perception algorithms, which are beyond the scope of this study, we assumed that the agents can perceive (hear and see) any object that is at the same floor and within 10 meters range.

Following the development of voice and visual perception mechanism, we were able to form a methodology for task distribution among agents. Although there are many complex ways to introduce collaboration among agents [Axelrod 1997, Feber 1999, Ercetin 2001], we used a simple but effective model, which reflects nature of command and control hierarchy. An agent informed of a task (event, request, order) generates a set of sub-tasks to meet the requirements of the main-task. Following task decomposition, additional sub-tasks for task distribution management are inserted. For instance, if the task is an event, the agent generates sub-tasks of the event and an additional task for reporting to the superior. If the agent could not manage to inform his superior, he starts doing tasks that are not strictly depended on the completion of informing the superior. When he manages to give the report to the superior, the sub-tasks not completed yet are fully canceled, because because they will be distributed by the superior. Then the superior generates a list of sub-tasks for himself and for his sub-ordinates, and distributes the sub-tasks to his sub-ordinates considering the work load.

The path planning of agents for navigation is another challenging problem to be solved. Path planning is defined as searching for a set of state transitions to reach to a goal location from an initial location. It is cetegorized in to two: off-line [Deloura 2000] and real-time [Undeger 2001a, Undeger 2001b] path planning. Off-line path planning has the advantage of generating high quality routes, but takes much CPU time and not suitable for highly dynamic environments. On the contrary, real-time path planning algorithms give poor solution quality, but is highly interactive and very adaptive to changing conditions.

The technique employed in our study is off-line path search through the connection graph, which is discribed in Section 4.

6.0 RUNNING THE SCENARIO

Once the behaviors and tasks of agents are modeled, the next step is to run the scenario. This function is directly supported by *TBtnEngine*. The *TBtnEngine* allows selection of time management metodology and simulation time multiplier. There are three main time management modes: event based, real-time and constant time interval. In the event-based time management mode, the events are tracked in the order that they will occur and the simulation is advanced to the nearest one. Therefore, the scenario is run as fast as possible in this mode. This mode is currently under development. The second mode is real-time, in which the time is advanced in parallel with the real-time (or a multiplier of real-time). In this mode, there is a possibility that the advance of a single step of the simulation will take more time than desired. For this reason, this mode is divided into three sub modes: unlimited time steps, constant time steps, upper bounded time steps. If unlimited time steps mode is used, the simulation time steps continue until all the related code is executed. If constant time steps mode is preferred, the execution is interrupted and passed to the next step when the specified constant time is exceeded, else a delay is inserted to reach the specified constant time. In upper bounded time steps mode, the execution is interrupted and passed to the next step if specified constant time is exceeded. The last time management mode, constant time interval, is generally used in debug mode, which ignores real-time and advances simulation time with constant time step, no matter how much time the step actually takes.

After starting the simulation, the simulation state can be observed on visualization window and messages are printed on the message window. The environment, the location and body posture of agents, the state of devices and the messages exchanged are all shown on visualization window. The messages (BTN execution messages, HLTMS execution messages, and run-time interpreter messages) are printed to the message window. The visualization and message windows are shown in Figure 10.

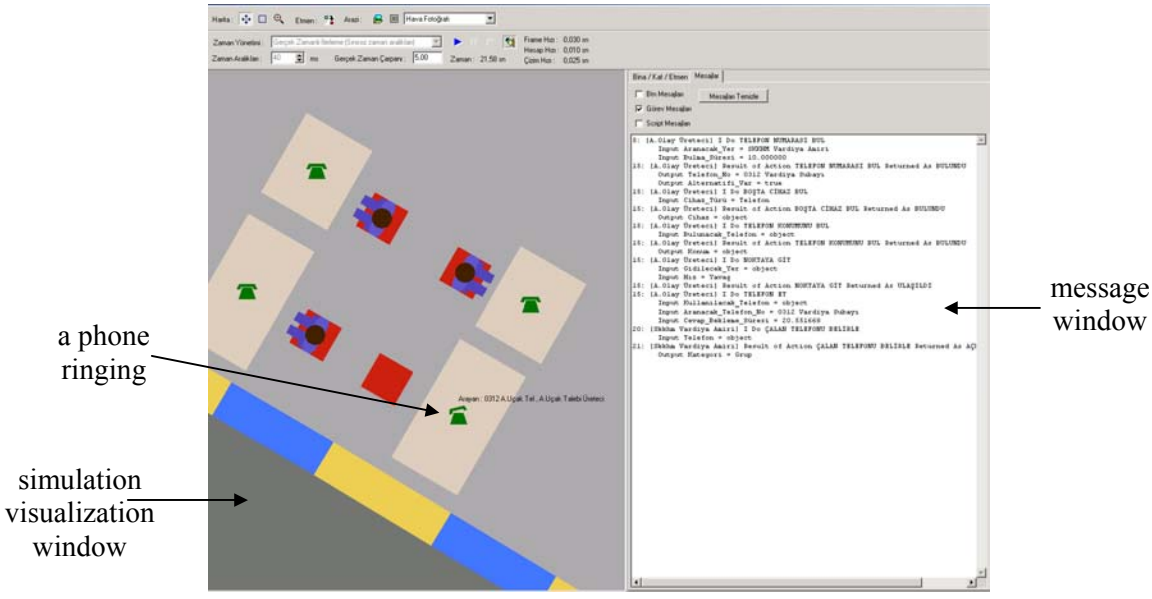

Figure 10. A snapshot of the simulation in execution

7.0 CONCLUSION

In this paper, we have proposed a simulation framework and a simulation tool for modeling command and control centers. First, we have started with the definition and properties of software agents, and clearly stated the reasons for developing our framework on top of agent based systems. Later, the generic simulation engine designed to realize the framework, and the customized engine for C4AT have been presented in detail. We have mostly focused on our agent-based system, which employs Behavioral Transition Networks and a newly proposed approach called High Level Task Management Script. For C4AT analysis toolkit, a communication framework and an environment design is developed and a sample scenario is generated. The first version of our implementation has given promising results for modeling a larger scenario that covers all the functions of a ComConCent. Thus, we are currently studying on the tool for defining new agent behaviors and tasks to improve the system.

8.0 REFERENCES

[Axelrod 1997] R. Axelrod, “The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration”, Princeton University Press, 1997.

[Budgen 1994] D. Budgen, “Software Design”, Addison-Wesley, 1994.

[Deloura 2000] Mark Deloura, “Game Programming Gems ”, Charles River Media, Inc., 2000.

[Ercetin 2001] A. Ercetin, “Operational-Level Naval Planning Using Agent-Based Simulation”, MS. Thesis, Department of Modeling, Virtual Environments and Simulation, Naval Postgraduate School Monterey, California, 2001.

[Ferber 1999] J. Ferber, “Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence”, Addison-Wesley, 1999.

[Ghezzi 1991] C. Ghezzi, M. Jazayeri, D. Mandrioli, “Fundamentals of Software Engineering”, Prentice-Hall, 1991.

[Harel 1988] D. Harel, “On Visual Formalisms”, CACM, Vol. 31, No. 5, 1988.

[Houlette 2000] R. Houlette, Daniel Fu, and D. Ross, “Towards An IA Behvior Toolkit For Games”, American Association for Artificial Intelligence, 2000.

[Rosenblum 1994] D. Rosenblum, A. L. Wolf, “Formal Software Engineering”, Tutorial SIGSOFT’94 FSE, New Orleans, Dec., 1994.

[Undeger 2001a] C. Undeger, “Real-Time Mission Planning For Virtual Human Agents”, M.S. thesis, Computer Engineering Department of Middle East Technical University, Ankara, Turkey, 2001.

[Undeger 2001b] C. Undeger, F. Polat, & Z. Ipekkan, “Real-Time Edge Follow: A New Paradigm To Real-Time Path Search”, The Proceedings of GAME-ON 2001, London, England, 2001.