Research Journal of Applied Sciences 2 (7): 780-787, 2007

Agent-Based Simulation for Real-Time Schedulers

Moutaz Saleh and Zulaiha Ali Othman
Faculty of Science and Information Technology, University of Kebangsaan Malaysia,
43600 UKM Bangi, Selangor

Abstract: Since several years, communication networks have known a surprising growth. The increased number of users, the consequent increase of traffic and the request for new services involve the development of new technologies and the deployment of high throughput networks. Networking technology has correspondingly grown to meet the diverse needs of applications and network administration. In response to the complexity of communication networks, simulation was and remains the only way to evaluate network performance. Unfortunately, traditional simulation methods are not adapted to all networks such as networks with quality of service and networks with dynamic aspect. To overcome this limitation, a new method to simulate dynamic networks based on multi-agents simulation and behavioral approach had been proposed. In this study, we present an adaptive approach to schedule real-time network traffic using the agent based simulation concept. The study introduces an Adaptive Real-Time Agent Scheduler (ARTAS) architecture and agent model as basis for scheduling real-time packet networks.

Key words: Simulation, agent bas, real-time, ARTAS, dynamic aspect, communication network

INTRODUCTION

There has been considerable recent interest in agent-based systems (Abeck *et al*., 1998; Dipippo and Wolfe, 2001; Drogoul *et al*., 2002; Gelenbe *et al*., 2001; Uhrmacher and Guglor, 2000) systems based on autonomous software and/or hardware components (agents) which cooperate within an environment to perform some task. An agent can be viewed as a self-contained, concurrently executing thread of control that encapsulates some state and communicates with its environment and possibly other agents via some sort of message passing (Wooldridge and Jennings, 1995). Agent-based systems offer advantages when independently developed components must interoperate in a heterogeneous environment, e.g., the Internet and agent-based systems are increasingly being applied in a wide range of areas including telecommunications, business process modeling, computer games, control of mobile robots and military simulations (Bradshaw, 1997). Agents are emerging as one of the key technologies which will facilitate the transition to and the implementation of the Information Society and, as such, they can have enormous long-term benefits to society.

While agents offer great promise, adoption of this new technology has been hampered by the limitations of current development tools and methodologies. Multi-agent systems are often extremely complex and it can bedifficult to formally verify their properties (Jennings and Wooldridges, 1998). As a result, design and implementation remains largely experimental and experimental approaches are likely to remain important for the foreseeable future. In this context, simulation has a key role to play in the development of agent-based systems, allowing the agent designer to learn more about the behavior of a system or to investigate the implications of alternative architectures and the agent researcher to probe the relationships between agent architectures, environments and behavior (Logan and Theodoropoulos, 2001). The use of simulation allows a degree of control over experimental conditions and facilitates the replication of results in a way that is difficult or impossible with a prototype or fielded system and it allows the agent designer or researcher to focus on a particular aspect of the system, deferring problems that are not central to the research or that are beyond the capabilities of current AI technology. Hence, simulation has traditionally played an important role in agent research and a wide range of test beds have been developed to support the design and analysis of agent architectures and systems.

In fact, multi-agent simulation derives from three strong ideas: To use models centered on entities and their interactions rather than on relations between measured values, to consider that the general dynamicity of the

Corresponding Author: Moutaz Saleh, Faculty of Science and InformationTechnology, University of Kebangsaan Malaysia, 43600 UKM Bangi, Selangor

system results from interactions between these entities and to make the assumption that agents are able to describe the behavior of the entities.

REAL-TIME SYSTEMS

A real-time system is typically composed of several or sequential tasks with timing constraints. In most real time systems, tasks are invocated repeatedly: Each invocation of a task is referred as a job; and the corresponding time of invocation is referred as the job's release time or job's deadline (Srinivasan, 2003). Thus, the relative deadline parameter is used to specify the timing constraints of the jobs.

A real-time system has two notions of correctness: Logical and temporal (Srinivasan, 2003). In particular, in addition to producing correct outputs (logical correctness), such a system needs to ensure that these outputs are produced at the correct time (temporal correctness). However, selecting appropriate methods for scheduling activities is one of the important considerations in the design of a real-time system (Zahang and Ferrari, 1994); such methods are essential to ensure that all activities are able to meet their timing constraints. These timing constraints are usually specified using a deadline, which corresponds to the time by which a specific operation must complete.

Real-time systems can be broadly classified as hard or soft depending on the criticality of deadlines (Lu *et al*., 2002). In hard real-time systems, all deadlines must be met; equivalently, a deadline miss results in an incorrect system. On the other hand, in a soft real-time system, timing constraints are less stringent; occasional deadline misses do not affect the correctness of the system.

RELATED WORK

To manage and promote disciplined development of agent-based systems, a comprehensive and detailed architecture is required. For such purpose, the past decades witnessed several proposed architectures with particular attention on the following architectures:

AI-based approaches: A variety of agent architectures has been proposed and employed with the emphasis on decoupling agents' actions (Brooks, 1986; Ferguson, 1992; Kaelbling, 1990; Muller and Pischel, 1994). Such designs can be characterized as deliberative, reactive and hybrid combinations of the two. The deliberative architecture promotes local and collective control, decision making and execution based on declarative knowledge. The class of reactive architectures is based on building a hierarchy of tasks ranging from primitive to complex, with the former having precedence over the latter. The remaining class of architectures is known as hybrid with the obvious interpretation. For example, the architecture proposed in (Brooks, 1986) is composed of three layers: Reactive, planning and modeling. The reactive layer is used to respond to simple needs of agents in the spirit of the reactive architectures. The other two layers are supposed to support other needs of an agent representing cognitive state of agents, reasoning about actions and communicating their decisions to the reactive layer.

Distributed software object-based approaches: The basis for such architecture is an object model accounting for abstraction, encapsulation, modularity and hierarchy (Booch, 1994). The object model provided the foundation to support the fundamental extensibility, scalability, management and persistence traits for software architecture. The Unified Modeling Language (UML) provides a set of modeling concepts and constructs. With UML diagrams, one can capture various static and dynamic aspects of a system.

Layered architecture: This architecture consists of seven layers, network (the lowest layer), middleware, simulation, modeling, search, decision and collaboration, where the lower layers provide services for the upper layers. Higher layers can use and possibly subsume the roles of the lower layers. Each layer is considered in isolation and in relation to others (Sarjoughian *et al*., 2001).

MULTI-AGENT SIMULATION PROCESS

In a discrete event simulation the time and nature of future events is computed in a predetermined fashion from the list of past events which have occurred. Thus the designer of a simulation will typically pre-specify all possible transitions and will not leave the definition of state transitions to entities which are not fully pre-determined. Thus it would be very useful to introduce agents in a simulation whose behavior is determined by specific circumstances and through adaptive behavior. The alternative we propose is that, in addition to the usual attributes of a discrete event simulation (such as event lists and possibly random number generator driven events), a simulation should contain a certain number of agents. These agents store information during a simulation and use it to modify their behavior during that same simulation or during distinct simulation runs.

From the point of view of agent-based simulation, we break up our simulation into 3 stages as shown in Fig. 1.

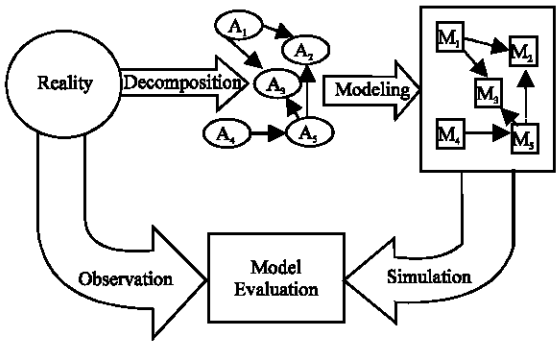

Fig. 1: Agent-based simulation stages

First stage: Includes the decomposition of the real phenomenon into a set of autonomous elements that interacts between each other and whose interactions reproduce the real phenomenon.

Second stage: Includes modeling each element by an agent and definition of its knowledge, its functional capacities, its behaviors and its interaction modes with other agents.

Third stage: Includes the description of possible actions between agents, by defining the environment in which these agents evolve and the rules which control them.

By taking again the three stages described in Fig. 1, we place ourselves in an agent context. Thus, we follow the steps of a multi-agents simulation process:

System decomposition: A queuing model is an infrastructure that gathers 6 types of elements: server, scheduler, queue, source, packet and clock. In addition to the model itself, we have to create a new element (Main) that will be the principle actor of the system behavior.

Modeling each element by an agent: With regard to the elements of the environment, we consider two types of elements: static (passive) agents and dynamic (active) agents. To bring great dynamicity to the system and preserve computer resources we decide to model server, scheduler, queue and main as active agents while source, packet and clock as passive agents.

Description of possible actions between the agents: There are many possible actions between the agents, i.e., messages exchange between the agents to take certain action, change their policy, or update their information.

ARTAS ARCHITECTURE

Our Adaptive Real-Time Agent Scheduling (ARTAS) architecture is a three-layered architecture as depicted in Fig. 2. At the lowest layer, we assume having a Real-Time Operating System (RTOS). Above that are the real-time middleware services which consist of the real-time scheduling agent and the real-time queue agent. At this layer, all tasks are scheduled based on the implemented algorithm, i.e. in our case the Hierarchical Diff-EDF scheduler. Finally, the real-time agent services layer extends the scheduling services resulting in task completion. Moreover, the top layer provides a service for real-time agent communications and interactions.

RT packet agent: Communication among agents in this architecture is performed through a packet request for service from one agent to the others. Each packet request has a formal description of <A, I, D >, where A represents the source to generate this packet request, I is the flow priority to which this packet belongs and D represents the deadline by which the request packet must be completed. If the servicing agent cannot complete this request before its deadline expires then it will be discarded.

RT main agent: As we mentioned earlier, the Main agent is the principle actor of the system behavior. Hence, all packet requests should be sent through this agent. It is the agent to control the entire environment through communications with the other agents in the system. The Main agent enforces other agents' policies, disciplines and actions.

RT service agent: Is an active agent which is responsible for packet request completion. It is the Main agent which tells this agent when to change their service discipline. The service agent keeps track of the missed packet and reported them to the Main agent.

RT scheduler agent: Our real-time agent scheduling algorithm performs schedulability based on the EDF algorithm. The packet request is scheduled depending on their original flow, i.e., RT or Non-RT flow. The scheduler agent treats the incoming packets to achieve the best QoS flow's requirements.

RT queue agent: This is an active agent which treats the incoming packets and place them in the appropriate defined queue based on their flow characteristics

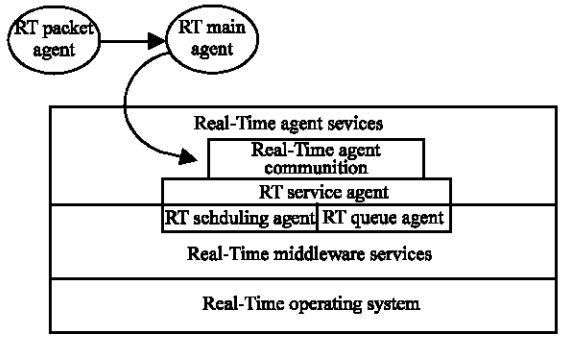

Fig. 2: ARTAS architecture

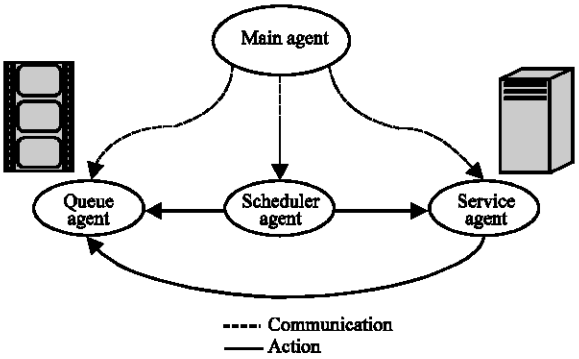

Fig. 3: Agent model

requested by the Main agent. The queues have limited size and provided with threshold. The queue agent monitors the queue threshold and interacts with the Main agent to inform the queues' status. It also, has the authority to control the queues' filling rate in real-time and the number of discarded packets according to their class.

AGENT MODEL

Our RT Agent model allows for the expression and enforcement of timing constrains on real-time agent interactions. The model is based on the assumption that agents may be able to perform their tasks in variety ways. It is made up of Real-Time Agents (RTAgent) and a set of communications among the real-time agents. Figure 3 displays the active elements of the model.

SYSTEM AGENTS IMPLEMENTATION

The system is modeled as agents and their interactions rather than relations based on calculated measures. The agents in the system are these entities

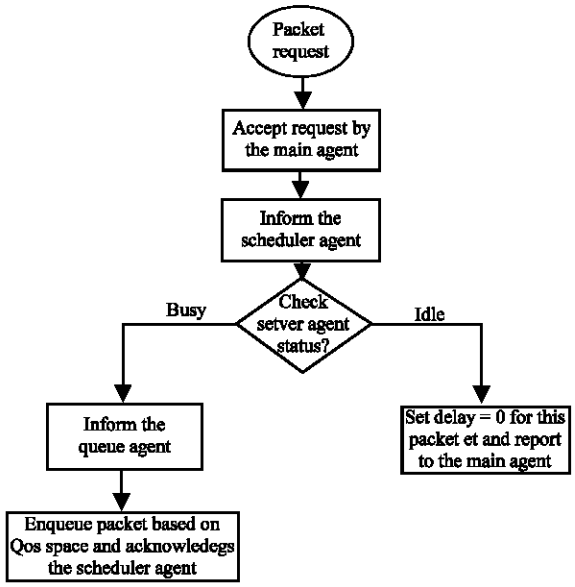

Fig. 4: Packet request process

capable of describing it own behavior with other entities within the system and with the environment i.e., constraints specifying dynamicity. The phenomenon (real system) is modeled by composing the agents shown in the agent model. For each agent, we identify the definition of its knowledge, its functional capabilities, its behavior and its interaction modes with other agents.

Main agent: The main agent is the principle actor of the system behavior. It monitors the other agents and provides them with the directions for use. It decides changes of behavior, treats the messages coming from the other agents. The main agent takes decisions, has learning and high communication level possibilities and can provide other agents with information at any time during the simulation. The main agent has several functional capabilities such as setting the system environment, managing packets' request and generating reports. Figure 4 shows the packet request process handled by the main agent. The process starts when the Main agent accepts the request. At that time the Main agent will interact with the scheduler agent to inform a new request arrival. As a part of the operation, the scheduler has to communicate with server agent to check his status. If the server agent replies with idle then the request will immediately served and reported to the Main agent. Otherwise, if replies with busy, the scheduler agent has to communicate with the queue agent requesting to enqueue the packet. The queue agent will execute the request, based on the packet QoS specifications and the action will be acknowledged to the scheduler agent.

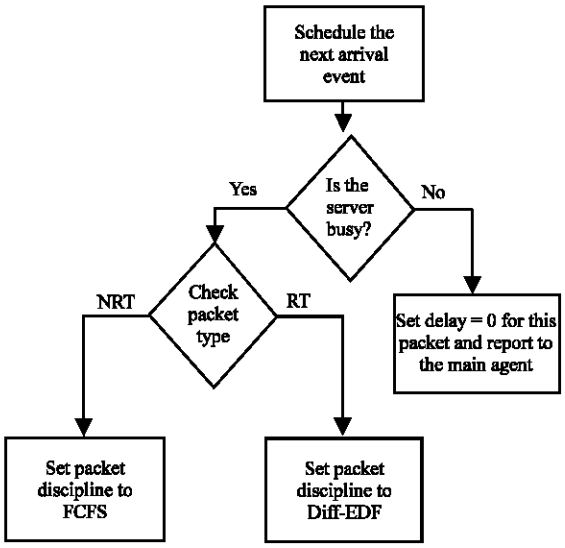

Fig. 5: Packet classification process

Scheduler agent: The scheduler agent is the key behind enforcing expressed timing and QoS constrains. It plays the main role in determining the disciplines of how the packets are going to be served. For our scheduler agent, two disciplines, in hierarchical arrangement, have been defined: The Real-Time (RT) such as EFD used to schedule multimedia traffic and the None-Real-Time (NRT) such as FCFS used to schedule the best effort traffic. The main tasks this agent has to perform as a part of our model implementation include packet classification, packet deadline adjustment and monitoring system events such as arrival and departure. For example, Fig. 5 shows the packet classification process being handled by the scheduler agent.

Queue agent: This agent is the system queues' manager. It is responsible for treating the incoming packets, when the server agent busy, by placing them into the appropriate defined system queue. To complete such a task, the queue agent performs an enqueuing operation; where the EDF queue is used for the real-time packets while the FCFS queue is used for the Non real-time packets. On the other hand, the queue agent performs a dequeuing operation whenever the server agent requests an enqueued packet to get ready for service.

Enqueuing operation: To implement such operation, we use the Linked List data structure. For FCFS queue, we use ordinary linked list as packet arrangement is based only on the inter-arrival time of each packet. Consequently, packets are enqueued with smallest inter-arrival time at the head of the queue. On the other hand,

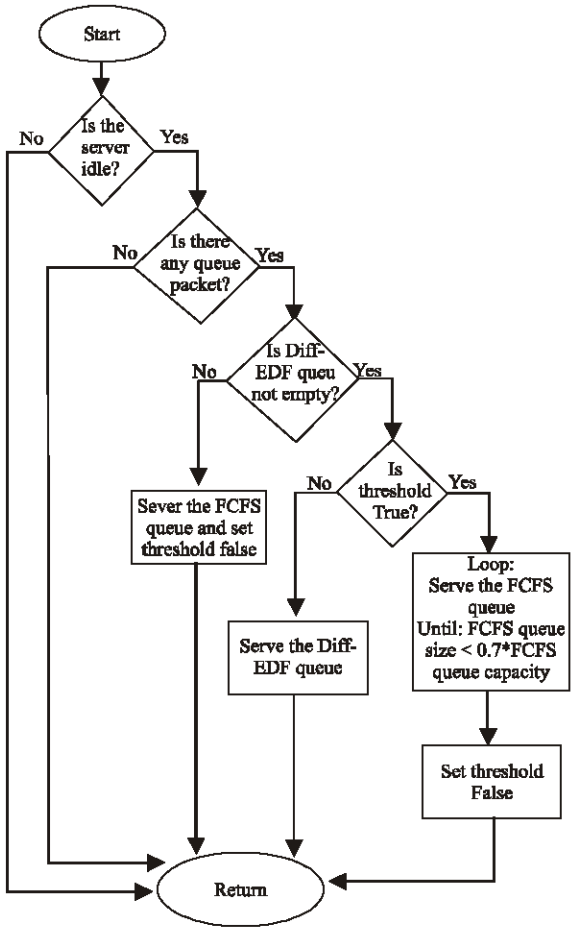

Fig. 6: Service process control

the EDF queue is implemented using Sorted Linked List, where sorting is based on the min value of packet deadline and inter-arrival time. The reason to choose this type of Linked List is to reduce scheduler complexity, so that rather than the scheduler will spend the time in picking up the shortest lead time packet to get serve, the queue is ready to be served starting from the queue head.

Dequeuing operation: This operation allows the queue agent to keep track of all packets leaving the queues to be served. The implementation of this operation is simply carried out through always picking up the packet on the head of the requested queue, identified by the server agent, to be served. Such implementation shows higher system utilization.

For the purpose of managing hierarchical scheduling, our system features a feedback control mechanism that detects overload conditions and modifies packet priority assignment accordingly. This feature is handled by the queue agent through keep observing any overload

conditions at the FCFS queue. The overload condition is detected whenever the FCFS queue size exceeds its defined threshold value. If the overload condition is detected, the queue agent immediately interacts with the Main agent to inform the condition. At this point, the Main agent changes the server agent policy, to serve the FCFS queue, until system stability. The key behind enforcing the threshold is to prevent packet starving which consequently improves the system performance.

Server agent: As its name indicates, it is the agent to handle the service process resulting in packet life cycle completion.

The two main tasks assigned for this agent are:

Controlling service process: As we mentioned earlier, if queue agent detects overload conditions then the Main agent has to change the service policy of the server agent. Hence, the server agent will control the service process to achieve system stability. To perform this task, the server agent follows the steps identified in Fig. 6. The algorithm is based on the idea that the server always serves the packets in the EDF queue (high priority) and serves the FCFS queue (low priority) if and only if the EDF queue is empty or the FCFS queue reaches its threshold value.

Performing packet service: In this task, the server agent has to initially compute the packet queuing time as in the following formula:

Packet Queueing Time = Simclock Current Time-Packet Born Time

After getting this value, a comparison operation is performed to determine whether the packet can proceed in its service process or it is considered to miss its QoS constrains. The comparison operation is defined as in the following:

If (PacketQueueingTime >= PacketQoS) → Missed Packet Otherwise → Proceed Service process

If this operation returns a missed packet, then the server agent will check the flow to which this packet belongs and encounter a miss in that flow. The result will be transferred to the Main agent for the purpose of generating the report at the end of simulation. On the other hand, if the operation results in proceeding the service then the server agent will set its status to busy, request the needed service time from the packet agent and

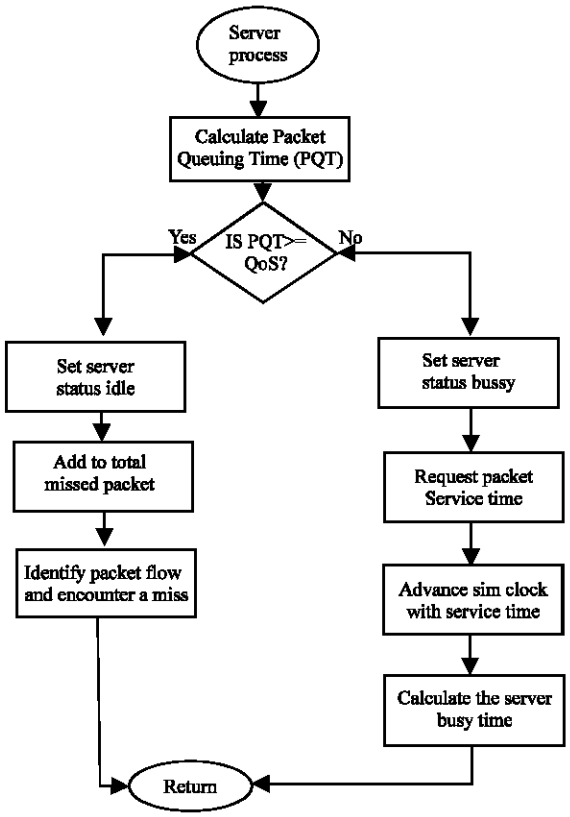

Fig. 7: Packet service process

start the service process. During each packet service, the server agent will advance the clock agent with the packet service time value, increase the number of served packets and compute the server busy time to be used in calculating the server utilization, by the main agent, at the end of the simulation. The whole service process carried out by the server agent can be shown in Fig. 7.

Source agent: This agent is the system actor of generating the requested traffic determined by the Main agent. It uses several arrival processes' models and the distributed random variants to generate the required traffic. Packet generation steps carried out by the source agent includes determining the exact source which this packet belongs, determining the Packet type, generating packet's inter-arrival time and service time, generating packet deadline and obtaining initial seeds.

For each packet to be generated there should be an exact time at which the packet born and an exact time at which the packet completes. Consequently, the source agent uses the exponential distribution with a mean of $1/\lambda_j$ to generate the packet inter-arrival time and $1/\mu_j$ to generate the packet service time. For QoS constrains, each

packet is associated with a deadline that precisely identify its living period. If the packet exceeds its predefined deadline before being served then it is considered to be useless. Hence, the source agent has to generate this value based on the uniform distribution as in the following formula:

PacketQoS = QoSGenerator.uniform (QoSMax, QoSMin)

Another main task to be handled by the source agent is to ensure that all generated packets obtain an initial seeds based on their flow characteristics. This task is carried out by implementing seed initializer technique that initializes different seeds for the variety flows as shown in Fig. 8.

```
private static void assignSeeds(){
   for(int streamID=1;streamID<=numberOfStreams;streamID++){
     seed[0] = 1973272912;
     double z=1973272912.0;
     for(int i=1;i<=numberOfStreams;i++){
       do{
       z=Math.IEEEremainder(715.0*z,2147483647.0);
       z=Math.IEEEremainder(1058.0*z,2147483647.0);
       z=Math.IEEEremainder(1385.0*z,2147483647.0);
       }while(z<0.0);
       seed[i] = (long) z;
     }
   }
  }
```

Fig. 8: Seed initializer technique

Packet agent: This is a passive agent since it does not have any learning capabilities. It only holds the packet information, received by the source agent and responds to other agents' requests with the needed information. The packet agent identifies the packet QoS constrains and its type.

Clock agent: This is the system actor of recording the events occurrence. It uses the Next-Event Time Advanced approach, shown in Fig. 9, to advance the simulation clock to the time of occurrence of the system events, where at this point the state of the system is updated to account for the fact that an event has occurred. This process of advancing the simulation clock from one event time to another is continued until eventually some pre-specified stopping condition is satisfied.

The advantage of using this approach is that periods of inactivity can be skipped over by jumping the clock from event time to the next event time. This is perfectly safe; since by definition all state changes only occur at event times. Therefore, causality is guaranteed. Furthermore, the event-driven approach is usually exploited in queuing and optimization problems.

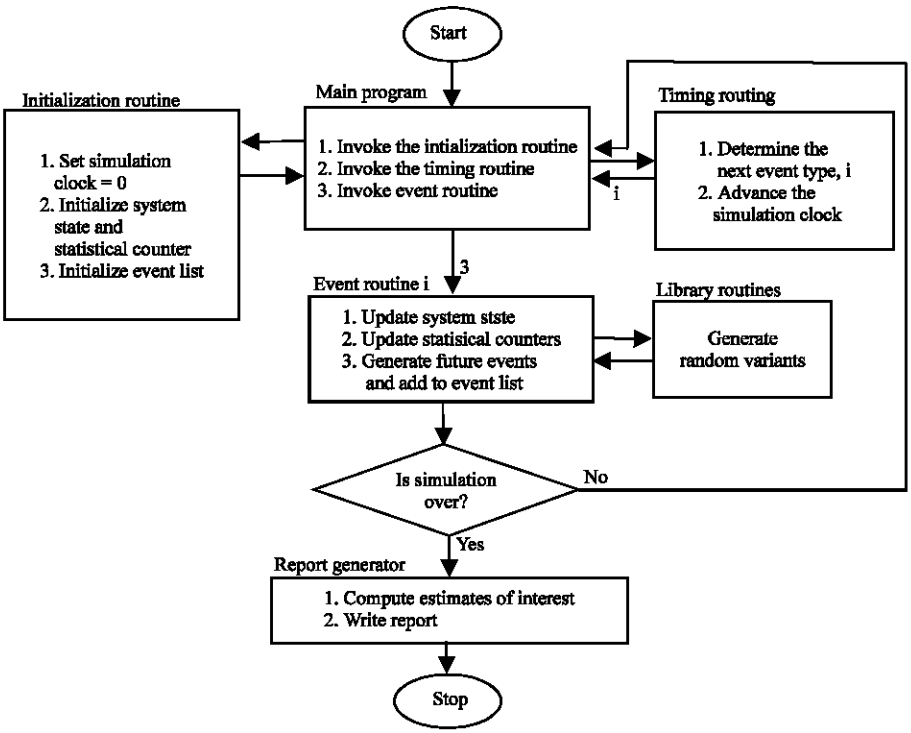

Fig. 9: Flow control for the next-event time-advance approach

Agents management: For our model management, agent actions are reported to the main agent. The main agent receives these reports as agent events. Possible events are arrival, departure, migration and termination. Dependent on the agent, any other events are possible.

We discriminate synchronous events where the event emitting agent waits for the main agent acknowledge and asynchronous events where the event emitting agent continues its work without acknowledge. Synchronous events will be used for agent control and monitoring. Asynchronous events represent simple monitoring information. Events like migration may be implemented as synchronous events. Control of the agent's moves is necessary if domain administrators want to restrict access to certain agent classes. Events like Termination may be asynchronous since it is sufficient to simply know that this particular agent ceases. Keep in mind that event classification is still a matter of research and highly application dependent.

The main agent handles the list of agents. This list comprises all agents that are currently in its domain. Every agent is listed with its identity, class, current location and task identifier. The main agent stores the incoming events and controls, in case of synchronous events, continuation of execution for every agent in the list. An event consists of the event type, the source (i.e., the emitting agent) and some event-specific information such as deadline.

CONCLUSION

During the past decade, there has been considerable recent interest in agent-based systems and multi-agent simulation. In this study, we have presented an adaptive approach to schedule real-time network traffic using the agent based simulation concept. During the multi-agent simulation process, we identified our simulation stages by decomposing the system into a number of elements, modeling each element with agent and last describing the possible actions between these agents. Also, we introduced our three-layered ARTAS architecture, agent model, system agents' implementation and management.

As a future research, we suggest to deploy the ARTAS architecture and agent model, proposed in this research, in variety real-time schedulers such as EDF and GPS to experience its efficiency in refining system's design and management.

REFERENCES

Abeck, S., A. Koppel and J. Seitz, 1998. A Management Architecture for Multi-Agent Systems. Proc. 3rd IEEE. Int. Workshop Sys. Manage., pp: 133-138.

Booch, G., 1994. Object-Oriented Design with Applications. Redwood City, CA: Benjamin/ Cummings.

Bradshaw, J., 1997. Software Agents. Menlo Park, CA: AAAI Press.

Brooks, R.A., 1986. A Robust Layered Control System for a Mobile Agent. The IEEE J. Robotics Automation Manage., 2: 14-23.

DiPippo, L.C. and V.F. Wolfe, 2001. A Real-Time Multi-Agent System Architecture for E-Commerce Applications. Proc. 5th IEEE Int. Symposium on Autonomous Decentralized Sys., pp: 357-364.

Drogoul, A., D. Vanbergue and T. Meuisse, 2002. Multi-Agent Based Simulation: Where are the Agents? The 3rd International Workshop on Multi-Agent Based Simulation, Lecture Notes in Computer Science, Springer-Verlag, 258: 1-15.

Ferguson, I.A., 1992. Toward An Architecture for Adaptive, Rational, Mobile Agents. Proc. Third European Workshop on Modeling Autonomous Agents and Multi-agent Worlds, pp: 249-262.

Gelenbe, E., E. Seref and Z. Xu, 2001. Simulation with Learning Agents. Proc. IEEE., 89: 148-157.

Jennings, N.R. and M. Wooldridge, 1998. Applications of Intelligent Agents. New York, Springer-Verlag, pp: 3-28.

Kaelbling, L.P., 1990. An Architecture for Intelligent Reactive Systems. San Mateo, CA: Morgan Kaufmann, pp: 713-728.

Logan, B. and G. Theodoropoulos, 2001. The Distributed Simulation of MultiagentSys. Proc. IEEE. 89: 174-185.

Lu, C., J.A. Stankovic, G. Tao and S.H. Son, 2002. Feedback Control Real-time Scheduling: Framework, Modeling and Algorithms. Special issue of Real-Time Sys. J. Control-Theoretic Approaches to Real-Time Comput., 23: 85-126.

Muller, J.P. and M. Pischel, 1994. An Architecture for Dynamically Interacting Agents. Int. J. Int. Coop. Inform. Sys., 3: 25-45.

Sarjoughian, H.S., B.P. Zeigler and S.B. Hall, 2001. A Layered Modeling and Simulation Architecture for Agent-Based System Development. Proc. IEEE. 89: 201-213.

Srinivasan, A., 2003. Efficient and Flexible Fair Scheduling of Real-time Tasks on Multiprocessors. Chapel Hill.

Uhrmacher, A.M. and K. Gugler, 2000. Distributed, Parallel Simulation of Multiple, Deliberative Agents. Proc. 14th IEEE Workshop on Parallel and Distributed Simulation, pp: 101-108.

Wooldridge, M. and N.R. Jennings, 1995. Intelligent Agents: Theory and Practice. Knowledge Eng. Rev., 10: 115-152.

Zhang, H. and D. Ferrari, 1994. Rate-Controlled Service Disciplines. J. High Speed Networks, 3: 389-412.