8-16-2011

Software Development Approach for Discrete Simulators

Grzegorz Chmaj
University of Nevada, Las Vegas, chmajg@unlv.nevada.edu

Dawid Maksymilian Zydek
University of Nevada, Las Vegas, zydekd@unlv.nevada.edu

Software Development Approach For Discrete Simulators

Grzegorz Chmaj
Department of Systems and Computer Networks
Wroclaw University of Technology, Poland
grzegorz@chmaj.net

Dawid Zydek
Department of Electrical and Computer Engineering
University of Nevada, Las Vegas, USA
dawid.zydek@unlv.edu

Abstract

Simulation is the most common approach to perform the problem research. Among several types of simulation, the most common way is the discrete simulation, which assumes the division of the time scale into fixed length time slots. Depending on investigated problem, simulation packages may be used or it could be necessary to design and create own simulation system. In this paper, we propose the complete pre-study scheme and the most commonly appearing implementation problems with suggested solutions. We also describe how to implement the exemplary simulator in C++.

Key Words: simulation, distributed computing, programming.

1. Introduction

Simulation and static modeling are most commonly used ways to research new ideas in computer systems. They are often used in computational grids, mesh structures, computer networks, etc. However, they operate in different areas of scientific research [11], [12]. Static modeling is based on building the optimization model, having the criterion function, decision variables and constraints. The aim of optimization may be either to minimize or to maximize the value of criterion function. To achieve the solution, each decision variable has to be set to the value in valid range (often these are binary variables), then the criterion function takes some certain value. Static modeling is used to solve Mixed Integer Problems (MIP) and enables to get the optimal solution, however this is possible only for very small network sizes. MIP problems are usually very complicated and for bigger problems (i.e. more close to reality), other solution approach has to be used. Heuristic algorithms may also solve static model, however then we are still constrained by the simplicity of static modeling.

The other way to research the scientific problem is to use the discrete simulator. In this case, the simulation model has to be build (static model may be used as the base model) and results are produced as the output of simulation, which is the process intended to act as close to real working system as possible [13], [14]. On the contrary to static modeling, where the result is ready after solution lookup is done (which can take lots of time in case of big problems), simulation allows observing partial results as they appear in the simulation process [13]. As the example, let us take the network containing many nodes, which exchange files between each other. This process takes time in real networks, so time progress has also to be considered in static modeling and simulation. In case of static modeling, time variable has to be used to state, if file *f* is available at the node *v* at time slot *t*. Static solver looks for the solution, and when it is ready, a researcher may analyze results and verify, when file *f* appears at the node *v*. Whole result for all researched time span is available (i.e. decision variables). In case of simulation, researcher may observe the exchange of files across the time, and does not have to wait for the final solution (end of investigated time span). In addition, simulation is able to investigate much bigger problem size than it is in case of static modeling. This example shows us the advantages of simulation research approach.

Literature shows us also the third way to research this kind of problems – the observation of real systems. This approach is rarely used, as it requires to monitor all units taking part in the process (in our example, we would like to monitor all network traffic for all nodes), what is usually not possible. In this paper, we will focus on building the simulator, including the software development aspect.

The rest of this paper is organized as follows: in section 2 we present related work, focusing on software implementation of discrete simulation, section 3 contains the simulation characteristics and guide to choose the right simulation approach. Chapter 4 presents the exemplary distributed computing system, section 5 describes the

software approach to simulation and chapter 6 concludes and describes the future work.

2. Related work

Discrete simulators are widely used to research complex problems, which cannot be investigated using the real hardware infrastructure (or this approach is too complicated or expensive). [1] describes the general properties of discrete simulation, the simulator components and the time scale issue. The authors of [3] described the method of consistency checking in discrete event simulators. They implement the mechanism to trace the event during simulation run and check if it is correct due to model constrains and requirements. This approach allows detecting errors in discrete simulation models. Software approach with the use of C++ was presented in [8]. C++/CSIM17 toolkit was described as the base architecture for creating further discrete simulation systems. Authors presented the framework containing the set of classes and processes. However the memory pointers are used, which we do not recommend here in our work. Web based simulation JBDS (Java Bean Discrete Simulation) architecture was shown in [4]. It is built using Java beans and was designed to implement the logic simulation with decision taking beans. It contains nodes (basic, controller and decision making nodes), events and entities. The goal of JBDS is to simulate the 'what if' decision problems. The aspect of presenting the simulation result data is shown in [10]. Authors focus on techniques of creating 3D representations based on simulation results, which are visualized as the graphics structure showing all possible sequences of simulation events. There is another approach to simulation – called *event-driven simulation* [5], which is more rarely used than discrete simulation. Event-driven simulation is controlled by events – time in simulation advances only when events occur (contrary to discrete simulation, where time advances no matter if any events occur or not). There are also hybrid solutions – in [6] authors proposed dual approach for time scale, which introduces event-driven mechanisms into discrete simulation.

3. Simulator characteristics

Discrete simulation offers many virtues, what is the reason it is widely used in scientific research. There are many simulation packages like OPNET [9] and NS-2 [7], however often researched problem does not fit to be simulated using them – for example NS-2 is designed to simulate networks focusing on protocols, what makes it not easy to simulate other category of phenomenon there. Main merits of discrete simulation are:

a) the ability to compress or expand time scale – some phenomenon in real system last certain number of time slots Δt. Because of that, using the third way of systems research (i.e. monitoring of real system) may be inefficient in case when operating time t is long enough to cause, that researcher will have to wait lots of time for the result. Using the simulation, Δt may be compressed to new value $\Delta t'$ without loosing the quality of simulation. The same profit we get in case when t is very small and investigated phenomenon is hard to observe in real system. Then the Δt may be expanded to $\Delta t'$ to make researched issue easy to observe in real time;
b) no measure errors – they do not exist in static modeling, but when the experiment uses real system monitoring, then the measure errors may appear and influence the result;
c) ability to tamper the simulation – the researcher may pause the experiment and set the unit property to certain value (e.g. set some flags, put more tasks to the unit, etc) or issue the event. This is the analogy to software debugging process. It is possible to set the breakpoints or to stop simulation manually;
d) ability to save and restore the simulation state – this enables to research the phenomenon which occurs somewhere in the simulation process and is not easy to be simulated directly (i.e. by inputting specified set of input data). In simulation, we can specify the breakpoint (which will be characteristic for researched phenomenon), stop the simulation after it occurs, save the simulation state for future restorations;
e) possibility to simulate in deterministic and stochastic way – simulation offers the deterministic mode – where for certain input data set, the result will always be the same. Stochastic mode allows using randomness – usually this mode moves simulation closer to real system;
f) possibility to operate on different levels of detail – some problems are hard to be modeled (also to be modeled in simulation), in this case it may be hard to find the appropriate level of abstraction. In simulation, researcher may add constraints one by one and observe, if the simulation results are satisfying.

To create the simulator, user has to go through several steps – to find the simulation type most matching the researched problem. This process is depicted on the diagram in Fig.1. At first, the type of model must be selected (decision 1). Stochastic model type allows using randomness and for specified input data, simulation may result with different output. In case of deterministic model type, no randomness is allowed, so for specified input data simulation will always return the same output. Chaotic model is the deterministic model having non-predictable elements). After model type is selected, time aspect is to be considered (decision 2): in static model no time scale is used, dynamic model changes across the time. In case of dynamic model, the type of time scale has to be selected

(decision 3). Discreet time scale divides time into slots (i.e. smallest time unit possible), and each time moment is characterized by integer number. Continuous time scale does not implement division into slots and the moment in time is characterized by continuous variable.

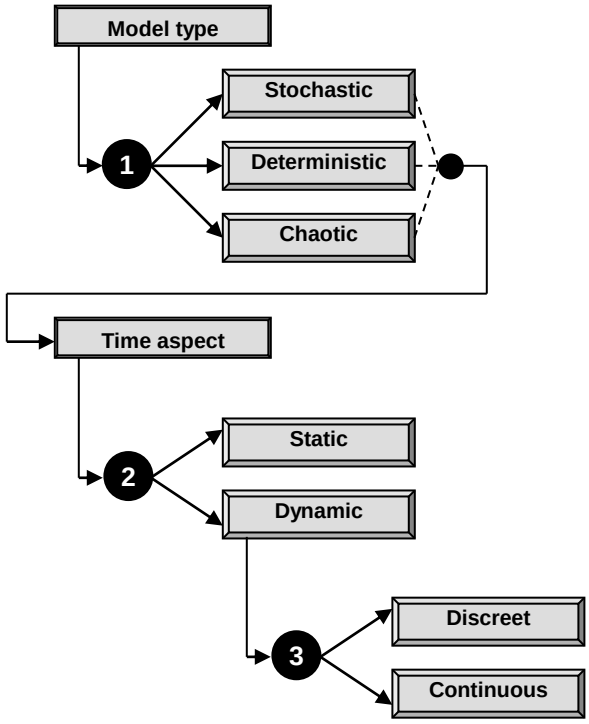

Fig. 1. Simulation type selection diagram

It is worth to consider other aspects of modeling: choose the most important model characteristics (more constraints may be added later), use the gradual model creation (adding more and more constraints, each time testing if model is valid/satisfactory). Output format is also important: it should be clear and contain appropriate information. Simulation output mechanism should be flexible and offer presenting certain variable only in specified range, or after specific event. This prevents the output from having thousands of records with variable having for example value of 0 from the beginning and for most of simulation time. In this case, its value should be recorded once at the beginning and then recording should start when the value starts changing. This mechanism we call *result triggering* (RT). After researcher went through above path, he is ready to develop the simulation system.

4. Distributed Computation Peer-to-Peer System (DCPS)

In this paper, we will use designed by us [2], exemplary distributed computing system based on peer-to-peer architecture. Distributed computing systems are well known in the IT world, the most famous one, Seti@Home is used to look for extra terrestrial intelligence and has few million of participants, offering their computation power. These kinds of systems are used for computation of tasks, which are too big to be processed on one machine, even on a super-computer. The task is divided at the managing node into small fragments called blocks, which are then sent to participants for computation. Receiving node performs computation and sends the result block to the managing node, where all partial results are combined into the final result. In case of problems, where the result is "binary" (i.e. thesis is confirmed or not, solution is found or not, etc.) there is no problem with sending the result to interesting nodes. However, for problems, where the result is big in the mean of byte size, the process of its sending to all interesting participants may overload server and choke the network. As the example, let us take a look at the ClimatePrediction project – its result is the weather prediction and has size of several MegaBytes (MB). This is why simple unicast network flow (server-client architecture) is not efficient in this case. That is why peer-to-peer flow is used – the result is disseminated among interesting nodes without significant use of managing node. Optimally, the managing node (which, at the beginning is the only node that owns the result) sends his copy only once. Summarizing, our distributing computation system computes the problem in distributing manner, and disseminates the complete result to all participants, taking part in computation process. We can describe our model the following way:

- system consist of two elements: *node* (v) – the machine (usually PC, Mac or other personal device) participating in computations and willing to receive the final result; *tracker* – managing node, hosting the knowledge base of data locations,
- *slot* (t) is the smallest time period considered in the system, during the slot node may perform some actions which do not require the response from other nodes nor tracker,
- *message* – nodes participating in DCPS exchange control messages between each other
- *request* – nodes send requests, when they want to receive some information or data from other system participants,
- *channels* – we propose to divide node resources into channels. Each node has specific computation power, upload and download speed. Thus we divide these resources the following way: download link is divided into C channels, each channel can transmit data with speed $D = d_v / C$, where d_v is the download bandwidth. Upload link is organized the same way. Computation power is also expressed as channels: available power is divided into P channels, each channel can compute one block at the time. The

channel model makes easier to manage the resources and in other form is used in many systems (e.g. DirectConnect protocol).

- *queues* – each node *v* has two queues: *queue of incoming messages (QIM_v)* and *queue of incoming requests (QIR_v)*. QIM_v receives the messages sent by other nodes to node *v*, *QIRv* receives the requests, respectively.
- *messages and requests* – we define the following types:
    - source block request
    - tracker update
    - result block location request
    - result block request
    - upload acknowledgement
    - block's locations list
- decision at nodes in DCPS are taken based on:
    - list of missing blocks
    - costs of transfer
    - knowledge about frequency of blocks among nodes.

DCPS takes the following assumptions, strongly related to simulation process:

- the process of sending the source block request takes some time, measured in number of slots *t*;
- the number of time slots *t* required to transmit the block between pair of nodes depends on the upload speed of sending node and download speed of receiving node;
- source block computation time at certain node depends on its computation power, one block cannot be computed using more than one slot, even if other slots are not busy;
- all activities performed by the node are fully independent;
- node may simultaneously perform the following activities:
    - send the request
    - send the acknowledgement for block downloading
    - compute the source block
    - download and upload result blocks.

DCPS uses the overlay network – this approach is very often used in network modeling and in daily used systems. The overlay network assumes, that top network layer provides direct network connection between each pair of machines connected to overlay network. They do not have to wonder how this connection is established, as this process happens at some lower layer. The examples of overlay networks are: Internet – where machine A sends data to machine B using its IP address, routing of packets is not visible for A and B; traditional phone calls: person A dials the number of person B without wondering how phone centrals commute the connection. Each node has its upload and download speed, defined as the speed to the overlay network – this resembles the network providers parameters, which limits user's speeds to network gateway, which for user acts as the entrance to the overlay network. The scheme of our DCPS is presented in Fig. 2.

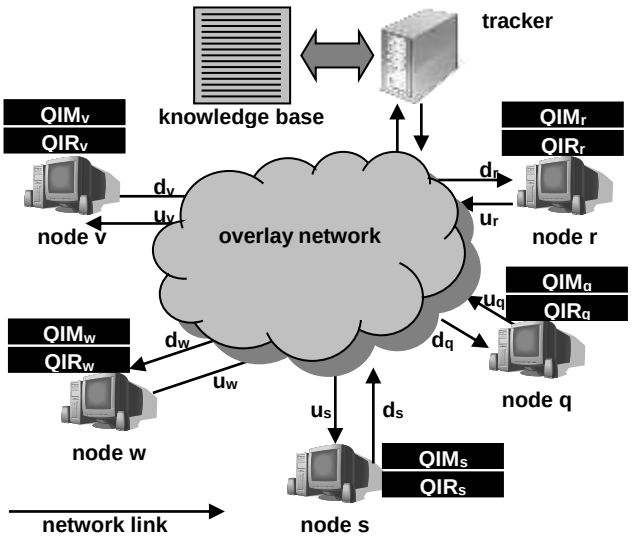

Fig. 2. DCPS network architecture

5. Software development approach

Starting the software development process for the discrete simulator, the author has to choose the programming environment. At first it is worth to discuss if the standard simulation environment, such as OPNET [9] may be used. If not, (the case we discuss in this paper) we come to general purpose programming environment. Among many of them, authors mostly focus on Java, C++, C#, also script languages such as python become popular recently. Sometimes the good idea is to relate the simulation part with static optimization part, as common parts of code may sometimes be used in both subsystems. We will describe the architecture of our simulation system using C++ with STL library. We formulate the following programming suggestions:

- if using C++ it is suggested to use STL or other library enabling not to use pointers and avoid memory fault problems;
- build multi-platform program: this is 'automatic' in case of Java technology, in case of C++, following the standards enables building the programs that can be compiled and run both under Linux and Microsoft Windows (two most popular operating systems);

- use C++ preprocessor to cut out unnecessary parts of code, what may be very important in complicated simulations.

We propose to build the simulation system as fully object oriented architecture, so we define the following classes, also depicted in Fig. 3:

- *grid:* defines the base structure of system, and is the skeleton for nodes and tracker;
- *node:* describes computational node properties and functions. Implements channel architecture for downloading, uploading and computation. Models computation and transfer costs, QIM and QIR queues and mechanisms determining which node's resources are busy at certain time;
- *tracker:* models the tracker element, implements knowledge base, functions for receiving requests and sending answers to nodes, and functions managing source blocks;
- *task:* describes computational task, contains task data as `string`;
- *source block:* describes source block object, contains computational data as `string`, and property determining to which node particular source block is assigned;
- *result block:* models resource block, contains result of computation as `string`;
- *gpsim:* class responsible for simulation system as the general organization. Is the top form for all other objects (they are created within gpsim object). Defines system structures such as:
  - Link (the network link between two nodes)
  - NodeRequest (request sent by node)
  - NodeMessage (message sent by node)

  Also defines helping functions, such as: sort, statistical and data load functions. Defines all STL vectors, creates all objects required for simulation based on other classes, and defines functions implementing simulation process, algorithms, etc.
- *base:* helper class, providing result logging and some helper parameters;
- *simulator:* is the main class for simulation part (important, if same system implements also other types of research, such as MIP solving, heuristics, etc.). Manages simulation cases (for example various network flows) and runs appropriate simulations;
- *sys:* manages all non-classifiable functions. Retrieves and parses command-line parameters, provides help for the user and manages time measurement (used to measure how long the simulation takes).

Additionally, two files are used:

- *main:* system part, retrieves input parameters and initializes appropriate classes
- *const.h:* contains global constants for the system: architecture global values, such as queue capacities, request costs, labels for binary variables. Handles preprocessor parameters.

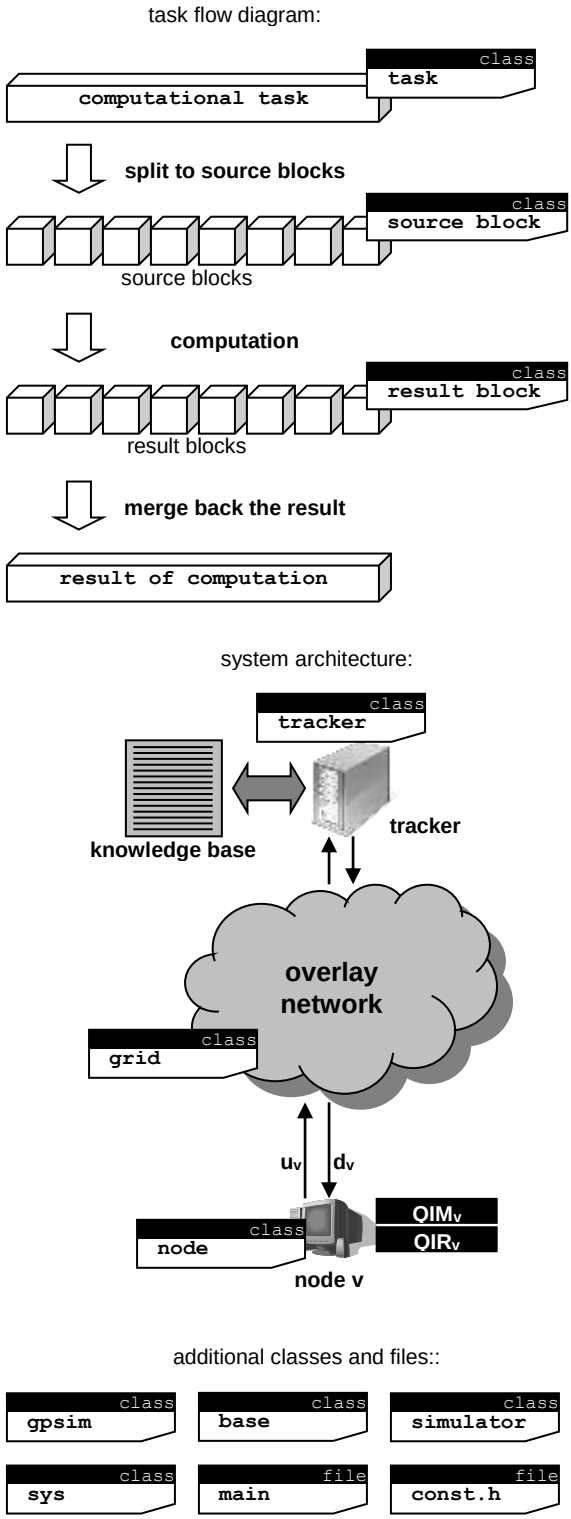

Fig. 3. Classes structure for DPCS

The development of discrete simulation system requires to handle several problems, which main of them are:

start and stop triggers – each simulation has to be started by start trigger which should lead to perform the full simulation. The approach in this case will be different for deterministic and stochastic simulations. For deterministic system, start trigger has to be strictly defined (e.g. sending the message to certain node). For stochastic approach, trigger may be selected randomly (e.g. random node receiving the message). Stop trigger determines when and how the simulation ends, so the simulation does not run infinitely. Stop trigger may be the same for deterministic and stochastic case.

the handling of node order processing – deterministic approach should handle the node processing by itself, and the processing order has to be strictly predictable. While using stochastic, which is far more close to real systems (because always we deal with more or less random things, e.g. two signals coming to one node at the same time causes situation, that it is not possible to state which of them will be processed first) – this mechanism must be implemented. Regarding our computation system, all nodes operate simultaneously, so we have to implement this phenomenon by random order of nodes processing. Each node can perform some actions, which may take some resources – so the order will influence on the result.

the handling of queues – as in real systems, nodes have queues with limited capacity. The way of taking elements from queue is determined by node algorithms, but looking from architecture point of view – the best way is to create STL vectors with position indicators. Queues could be implemented as one broker system – then one broker object handles all of them and internally tracks which queue belongs to which node. We propose to implement queues as the part of node class. If certain queue has different length among all nodes, then we also have to manage this parameter.

6. Conclusions

Simulation systems are very good way to investigate scientific ideas. When standard simulation packages are suitable to be used, then the researcher may focus only at implementing the right simulation parameters. Otherwise, simulation system written from scratch requires the programmer to deal with several difficulties. In this paper we presented the whole process of creating the simulation, including pre-study and analysis, implementation and handling the result output. We present our computational system and show how to create the simulation for such case. We also describe the mostly appearing problems and suggested solutions, so this paper can act as the guide for researcher planning to build the simulation system. As the future work we plan to work in areas of output management, to create best practices of creating output file according to researcher's needs.

7. References

[1] M. Bumble, L. Coraor, "Architecture for a non-deterministic simulation machine", *Proceedings of the 1998 Winter Simulation Conference*, 1998, pp. 1599–1606.

[2] G. Chmaj, K. Walkowiak, "A P2P computing system for overlay networks", *Future Generation Computer Systems*, 2011

[3] J.W.G. Fleurkens, C.A.J. van Eijk, J.A.G. Jess, "Run-time Consistency Checking in Discrete Simulation Models", *EDTC '95 Proceedings of the 1995 European conference on Design and Test*, 1995, pp. 223-227

[4] M. Fukunari, "JavaBean-Based Simulation With a Decision Making Bean", *Proceedings of the 1998 Winter Simulation Conference*, 1998, pp. 1699-1702

[5] I. Jawhar, "A Flexible Object-Oriented Design of an Event-Driven Wireless Network Simulator", *Wireless Telecommunications Symposium*, 2009, pp. 80–86.

[6] A. Lalomia, G. Lo Re, M. Ortolani, "A Hybrid Framework for Soft Real-Time WSN Simulation", *13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, 2009, pp. 201–207

[7] M. Rahimi, S. Parveen, M. Morshed, M. Khan, P. Sarker, "Development of the Smart QoS Monitors to Enhance the Performance of the NS2 Network Simulator", Proceedings of 13th International Conference on Computer and Information Technology ICCIT, 2010, pp. 137-141.

[8] H. Schwetman, "Object-oriented Simulation Modeling with C++/CSIM17", *Proceedings of the 1995 Winter Simulation Conference*, 1995, pp. 529-533

[9] Webpage of OPNET project: http://www.opnet.com

[10] Y. Zhong, B. Shirinzadeh, "Analysis, Conversion and Visualization of Discrete Simulation Results", Proceedings of the Eight International Conference on Information Visualization, 2004, pp. 118-123

[11] D. Zydek, H. Selvaraj, "Processor Allocation Problem for NoC-based Chip Multiprocessors," *Proceedings of 6th International Conference on Information Technology: New Generations (ITNG 2009), IEEE Computer Society Press*, 2009, pp. 96-101.

[12] D. Zydek, H. Selvaraj, "Hardware Implementation of Processor Allocation Schemes for Mesh-Based Chip Multiprocessors," *Journal of Microprocessors and Microsystems*, vol. 34, no. 1, 2010, pp. 39-48.

[13] D. Zydek, H. Selvaraj, "Fast and Efficient Processor Allocation Algorithm for Torus-Based Chip Multiprocessors," *Journal of Computers & Electrical Engineering*, vol. 37, no. 1, 2011, pp. 91-105.

[14] D. Zydek, H. Selvaraj, L. Koszalka, I. Pozniak-Koszalka, "Evaluation Scheme for NoC-based CMP with Integrated Processor Management System," *International Journal of Electronics and Telecommunications*, vol. 56, no. 2, 2010, pp. 157-168.