

# Dasor, a grid model based simulation library

Alain Bui, Olivier Flauzac, and Cyril Rabat

SysCom, CReSTIC

Université de Reims Champagne-Ardenne

BP1039, F-51687 Reims Cedex 2, France

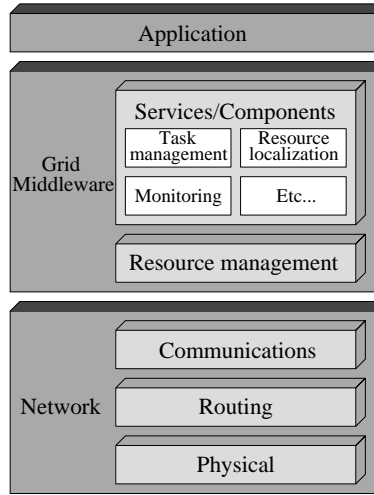
{alain.bui,olivier.flauzac,cyril.rabat}@univ-reims.fr

**Abstract.** Simulation is a key point for designing applications for large scale networks like grids or peer-to-peer networks. To be relevant, simulations must be computed with adequate models: communication models, fault models or network models (bandwidth, latency). In previous works, we proposed a five layer model to map grid applications and the subja-cent network. Particularly, it shows how subjacent protocols can impact higher layers.

In this article, we describe a simulation library called *Dasor* based on this model. It is a C++ discrete event simulation library that allows to build simulators independently of the network and the simulation models. So, a simulator can be executed on different environments like dynamical networks or wireless networks without any modification of the simulator. The simulation granularity can also be modified easily.

## 1 Introduction

Designing distributed solutions for large scale systems like grids or peer-to-peer networks implies several challenges. First, it needs a theoretical model to analyze the system (network limitations, impacts of data localization or resource volatility). Several models have already been proposed and we can gather them into two categories : the models that focus on application architecture and the models that focus on material and resources. The authors of [1] choose to model the network grid topology. This kind of model can highlight the bandwidth consumption and the problem of network overload but it is independent of the grid application. In [2], the authors propose a model based on the fabric notion. The component and the grid architecture are organized on several layers. But this model focus on the grid and does not take into account subjacent protocols or mechanisms. So, in [3], we propose a multi-layer model for grid applications that highlights both the application and the underlying protocols. Particularly, we are able to observe protocols impacts in higher layers. Figure 1 shows a simplified view of our model.



**Fig. 1.** The grid application model proposed in [3].

The next step for an application development is the distributed algorithm design depending on the chosen model. Before the development of a final application, we can simulate the behavior of the distributed algorithms to observe their scalability or their fault tolerance. But it could be difficult to choose a suitable simulation tool and simulation models to apply.

A number of simulation tools have already been proposed. But the simulation granularity is often hard to adapt. For instance, *OMNeT++* [4] and *Ns-2* [5] have been designing for the oriented network applications. These solutions focus on the low-layer protocols and do not propose to simulate component of grid or peer-to-peer applications. Other tools like *Narses* [6] that is a peer-to-peer application simulator, are based on flow simulation instead of a packet management. The simulation computation time is reduced but the simulation granularity is high. For instance, bottleneck problems cannot be simulated. Simulator users often need to modify the granularity depending on the tested component and the previous tools cannot be used easily.

In this article, we present *Dasor* that is a C++ library for discrete event simulation based on our model. It allows the user to interact at each level of our model. So, a simulator user can choose the suited simulation granularity by selecting the simulation models. Because simulators written with *Dasor* are designed independently of the network and the

simulation models, this choice can be modified without changing the simulator.

The rest of the paper is organized as follows. Section 2 provides related works on simulation tools and a presentation of our theoretical model. Section 3 gives a short description of the simulator. Finally, we conclude in last section.

## 2 Preliminaries

**Related works** *OMNeT++* [4] and *Ns-2* [5] aim to simulate the network protocols or a stack of protocols like TCP/IP and they propose a collection of communication models including TCP/IP, ATM, Ethernet and wireless protocols. But these tools focus on network low-layers with the exchanges of packets and it could be difficult to simulate large-scale systems. On the other hand, they do not propose tools to simulate specific components of grid or peer-to-peer applications like the scheduling or the resource management.

Some simulators have been proposed to study specific components of a grid. *GridSim* [7] is a toolkit to build simulators for application scheduling. Particularly, it *provides mechanisms to model performance characteristics either as constants or from traces*. *OptorSim* [8] is a Java simulation package and focus on Data Grids. Especially, it allows us to study and to design replica optimization algorithms. *MicroGrid* [9] creates a *virtual grid environment*. It enables to model networks, resources and information services. So, real application software and middleware can be executed above this virtual grid without any change on them. Naturally, experiments on *MicroGrid* imply having an implemented application. Other tools like *GridG* [10] focus on topology generation for grid based on power laws of Internet topology. These grid simulation tools cannot be easily used to simulate a whole grid application and particularly to show interactions between application components.

To focus on peer-to-peer networks, some simulators like *3LS* [11], *Narses* [6] have been proposed. They allows us to simulate protocols on a large-scale networks. *NeuroGrid* [12] focus on the simulation of file exchange protocols as *Gnutella* or *FreeNet* ones. These tools are protocol oriented and no module is proposed for scheduling or resource management simulation.

**A model for grid.** In [3], we have proposed a five-layers model for grid applications shown on Figure 1. It allows to distinguish each application

component and links between these components. It underlines also sub-jacent mechanisms that interact independently of the application. The three lower layers focus on the physical resources and the next two layers concern the middleware (the grid components). 6<sup>th</sup> Layer represents the application that interacts with the grid components.

*Network layer* (Layer 1). At this level, we model a network as a graph  $G_1 = (V, E)$ . Each node of  $V_1$  is a network element. It can be an active one (for example a desktop computer or a parallel machine) or a passive one (for example a router, a switch or a modem). A link represents a physical cable between two components or a wireless connection. So, it is not a directed link except for wireless connections if nodes ranges are different (and if the communication protocol supports it). On each Node  $i$ , we consider its neighborhood noted  $N_i^1$  that is the set of all connected nodes to Node  $i$  (*i.e.* all nodes that Nodes  $i$  can send a message to).

*Routing layer* (Layer 2). Above the physical network, a routing protocol builds routes between nodes of a network. Two nodes can contact each other even if they are not physically connected. But firewalls and network protocols can block some connections. It induces directed links. So, a network is modeled as a graph  $G_2 = (V, E')$  with  $E'$  is the set of allowed communication links. On each Node  $i$ , neighborhood  $N_i^2$  is the set of all nodes that Node  $i$  can establish a connection.

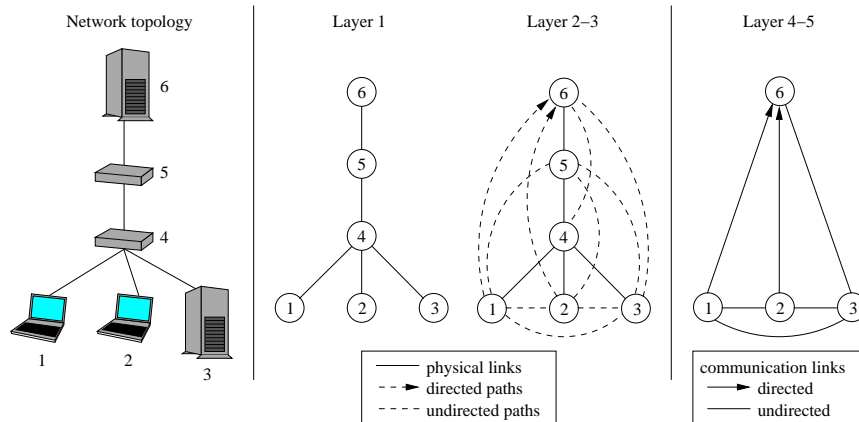
*Communication layer* (Layer 3). Above the routing layer, a set of protocols allows nodes to transfer data to another node by using built routes. According to the deployed protocols, some options can be applied: acknowledgment messages to ensure a message reception and a hash mechanism to check validity of message content. With IP protocols, if a message is not valid, it is automatically sent again by the emitter. In this layer, we have  $G_3 = G_2(V, E')$  and  $N_i^3 = N_i^2$ .

*Topology layer* (Layer 4). A grid or peer-to-peer application needs to manage its topology. It ensures connections of new nodes or disconnections and it maintains the reachability of all the resources (localization). We distinguish two kind of nodes. The *active* ones are connected to the grid and use or share resources. The *inactive* ones are the network components (routers or switches) or nodes that are not in the grid. So, the grid is modeled by a graph  $G_4 = (V', E'')$ , where  $V'$  is the set of active nodes

and  $E''$  the communication links between active nodes.

*Components layer* (Layer 5). The last layer concerns the grid application and its components. These components depend on the application. For example, in file exchange peer-to-peer systems, we need components to localize and transfer files and to manage the users. For a computational grid, we need a task management for task submission, assignment and resources allocation. In this layer, the grid is modeled by a graph  $G_5 = G_4(V', E'')$  and  $N_i^5 = N_i^4$ .

*Application layer* (Layer 6). Above the entire model, applications can use the grid components to access the resources transparently for the application user.



**Fig. 2.** Example of a grid through each layer of the model.

*Example 1.* Figure 2 shows an example of a simple grid topology through each layer of the model. We suppose we have 3 computers (1, 2 and 3) connected to a router (4). A distant server (6) is also connected to a router (5). At Layer 1, we represent each link between nodes (here, we have only undirected links) and at Layer 2/3, we represent paths between nodes. We suppose that Node 1 and 2 are protected behind a firewall and it induces two directed paths (Node 4 cannot contact directly Node 1 and 2). To finish, at Layers 4/5, the set of paths is a subset of Layer 2/3. Only some connexions are maintained between nodes.

This model highlights the impacts of protocols that interact under a grid. We can focus on a specific layer or a component. For example, the resource management (layer 4) has already been focused in [3, 13] and task management in [14].

### 3 Dasor

*Dasor* [15] is a C++ library for discrete event simulation of distributed algorithms. The aim of *Dasor* is to build simulators that are written independently of the network and simulation models. Indeed, the network and the models are only specified at the execution thanks to a *network description file*. The file syntax describes easily complex topologies and predefined models. The execution model of a simulator created with *Dasor* is based on the theoretical model described previously.

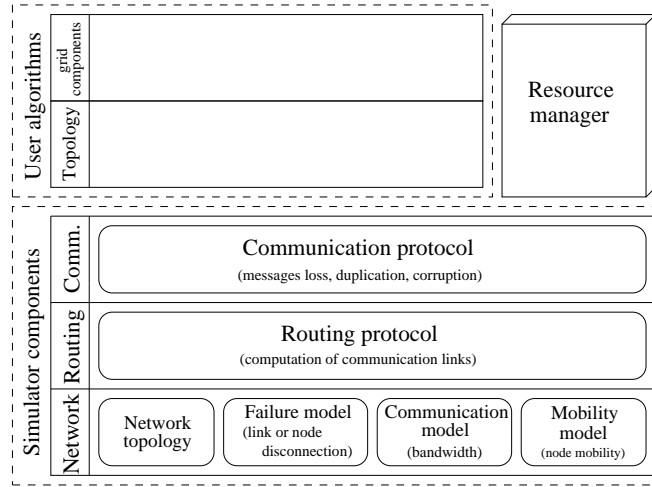
#### 3.1 Description of the library content

*Dasor* is used to build independent simulators that need some files at the execution: a configuration file (*.ini*) that contains general options like the number of simulations or the simulation maximal time, a network description file (*.net*) that describes the network and the used simulation models (see next section) and eventually some data files (for resource description like tasks, files or libraries). A simulator can execute series of simulations and provides a description file (*.des*) that contains all information about the simulations and a statistics file (*.sta*) that contains registered values.

The execution model of a simulator written with *Dasor* is based on the theoretical model as shown in Figure 3. Each simulation model is considered as a component of a theoretical model layer<sup>1</sup>. The selected simulation models are managed from the lower layer to the upper layers. A simulator is written independently of them and they are specified through the description file. So, it is possible to change between models or to modify models parameters without any modification of the simulator. By default, several models are applied on the network and it is not necessary for the simulator user to specify all models. So, it is possible to modify the simulation granularity by selecting or ignoring models. For instance, the description of a link between two nodes in the network description file can be a physical link if a routing model is chosen else it can be a communication link without routing model (by default, no routing

---

<sup>1</sup> If some models are not specified, lower simulation models can overlap several layers



**Fig. 3.** Execution model of simulators written with *Dasor* based on our theoretical model.

model is applied). The simulation granularity can also be adjusted according to the applied models. Some routing models simulate the packet exchanges whereas some models compute globally (in a centralized way) the paths between nodes. The computational power needed to execute simulation can be reduced but with a higher simulation granularity.

GNU Scientific Library [16] (GSL) has been integrated to *Dasor* for the random numbers generation. It can also be used in the description file to configure simulation models. For instance, the simple communication model called *fixed* interacts on the global communication time (the average time for a message to be transmitted from a node to another one). It can take as parameter a random number generator (for instance an uniform generator or a gaussian generator).

*The Dasor Toolbox* is a set of tools provided in the library. For instance, *makeSimu* assists to the creation of new simulators and *makeDiag* builds execution diagrams from output simulator files to show the task computation, the node failures or the message exchanges. It provides also some statistics about the produced events during the simulator execution. *convertNet* tool supports the conversion of a network file to a postscript file and can also be used to create a new network description file by re-

moving random events of specified simulation models. Then, such files can be used to reproduce some results.

### 3.2 Network description files

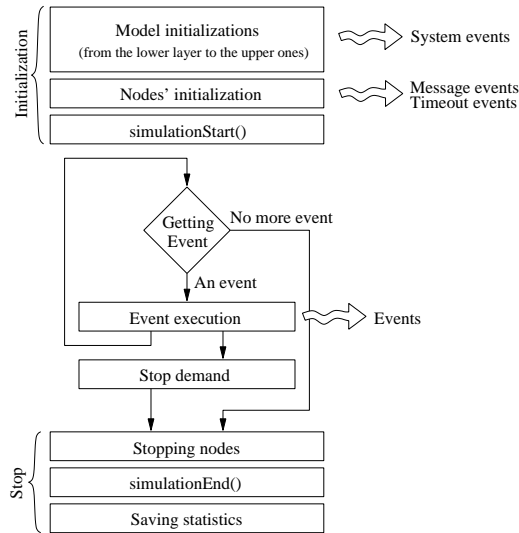
A *network description file* contains the network description and the simulation models applied on it. First, the number of nodes and global options must be specified: the network can be wired or wireless, the links can be directed or not and weighted or not (used to simulate distributed algorithms). For a wired network, the syntax of the network files allows to describe simple topologies (like directed or undirected rings, stars, grids, chains or trees) or more complicated topologies (like random networks, directed or undirected degree-constant networks, hierarchical caveman [17] and small-world). Links can be described one by one or with the full adjacency matrix. For wireless networks, links can be ignored and are computed from positions of nodes. Predefined position models can be used (like circle, random, grid or fixed density) and links will be computed during simulation depending on the node positions and the chosen communication model.

Network files allow also to describe the models used during the simulation. Particularly, we can choose between several fault models (fixed, random, or periodical), routing protocols, communication models or mobility models. By default, no fault model or mobility model are applied on the network and a default communication model is selected. Each model can be configured by specifying options. The periodical fault model involves that each node fails periodically. This model accepts two parameters: the interval for the fault length and the interval for the alive length. Instead of a model or to complete a model, the simulator user can describe simple events like a node failure or a node move.

### 3.3 Writing a simulator

A simulator is based on the description of the actions associated to an event. In *Dasor*, the following events are considered: a message reception, an ended timeout, a node breakdown and a node awakening, the beginning and the end of a task... In addition, other actions can be executed at the node initialization or at the end of a simulation. Figure 4 shows the execution diagram of a simulation.

By default, only one node type is specified. So, all nodes have the same behavior. The simulator writer can create several kind of nodes to differentiate the behavior of the nodes depending on an event trigger. In



**Fig. 4.** Execution diagram of simulators.

the same way, the messages types are defined by the simulator writer.

In the library, several structures are proposed to simplify the creation of new simulators: simple structures as tree or matrix and also complete tools as the resource manager. This manager automates the resource management like tasks, files or libraries and physical resources like the computational power of nodes, the physical memory or the storage space. So, it is possible to simulate easily the transfer of a resource from a node to another one. The physical capacities of nodes and logical resources (tasks, files and libraries) are defined in the network description file. It is possible to select a model for each one.

## 4 Conclusion

In this article, we introduce *Dasor* that is a C++ library for building discrete event simulators of grid applications. It is based on a theoretical model for grid and peer-to-peer applications. Simulators are written independently of the network and the simulation models that are given as parameters at the execution. So, simulator users can select the appropriate simulation granularity without modifying the simulator.

Our future works will focus on the development of new models for *Dasor* especially fault models that take into account the lifespan of nodes.

We have already developed a tool to compute simulations on parallel machine. We plan to integrate the simulator to a grid middleware to compute simulations in parallel.

**Acknowledgments:** *This work was partly supported by “Romeo”<sup>2</sup>, the high performance computing center of the University of Reims Champagne-Ardenne, France.*

## References

1. S. Lacour, C. Perez, and T. Priol, “A Network Topology Description Model for Grid Application Deployment,” in *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 61–68.
2. M. Baker, R. Buyya, and D. Laforenza, “Grids and grid technologies for wide-area distributed computing,” *Softw. Pract. Exper.*, vol. 32, no. 15, pp. 1437–1466, Dec. 2002.
3. C. Rabat, A. Bui, and O. Flauzac, “A random walk topology management solution for Grid,” in *II2CS*, ser. LNCS, vol. 3908. Springer-Verlag, 2006, pp. 91–104.
4. A. Varga, “The OMNeT++ Discrete Event Simulation System,” in *ESM'2001*, June 2001.
5. “The Network Simulator - Ns-2,” <http://www.isi.edu/nsnam/ns/>.
6. T. J. Giuli and M. Baker, “Narses: A Scalable Flow-Based Network Simulator,” *CoRR*, vol. cs.PF/0211024, 2002.
7. H. Casanova, “Simgrid: A Toolkit for the Simulation of Application Scheduling,” in *CCGRID '01*. IEEE Computer Society, 2001, p. 430.
8. W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini, “Simulation of Dynamic Grid Replication Strategies in OptorSim,” in *GRID '02*. Springer-Verlag, 2002, pp. 46–57.
9. X. Liu, H. Xia, and A. A. Chien, “Validating and Scaling the MicroGrid: A Scientific Instrument for Grid Dynamics,” *Journal of Grid Computing*, vol. 2, no. 2, pp. 141–161, 2004.
10. D. Lu and P. A. Dinda, “GridG: generating realistic computational grids,” *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 4, pp. 33–40, 2003.
11. N. S. Ting and R. Deters, “3LS - A Peer-to-Peer Network Simulator,” *p2p*, vol. 00, p. 212, 2003.
12. S. Joseph, “An Extendible Open Source P2P Simulator,” *P2P Journal*, 2003.
13. T. Bernard, A. Bui, O. Flauzac, and C. Rabat, “Decentralized Resources Management for Grid,” in *RDDS'06*, ser. LNCS, vol. 4278. Springer-Verlag, 2006, pp. 1530–1539.
14. A. Bui, O. Flauzac, and C. Rabat, “Fully Distributed Active and Passive Scheduling for Grid Computing,” in *ISPDC07*. IEEE Computer Society, 2006, to appear.
15. “Homepage of DASOR,” <http://cosy.univ-reims.fr/~crabat/DASOR/>.
16. M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi, Eds., *GNU Scientific Library Reference Manual*. Network Theory Ltd., 2006.
17. D. J. Watts, *Small Worlds*. Princeton University Press, 1999.

---

<sup>2</sup> <http://www.univ-reims.fr/Calculateur>