Dasor, a Discrete Events Simulation Library for Grid and Peer-to-Peer Simulators

Cyril Rabat

ABSTRACT. Simulation is a key point for designing applications for large scale systems like grids or peer-to-peer networks. To be relevant, a simulation must be computed with adequate models: communication models, fault models or network models (bandwidth, latency). In previous works, we proposed a five layers model to map grid applications and the subjacent network. Particularly, it shows how subjacent protocols can impact higher layers. Here, we describe a simulation library called Dasor based on this model. It is a C++ discrete events simulation library that allows to build simulators independently of the network and of the simulation models. So, a simulator can be executed on different environments like dynamical networks or wireless networks without any modification of the simulator code. The simulation granularity can also be modified easily.

RÉSUMÉ. La simulation est un point clef pour la création d’applications destinées aux systèmes déployés à grande échelle tels que les grilles ou les réseaux pair-à-pair. Pour être pertinentes, les simulations doivent être réalisées avec des modèles appropriés : des modèles de communication, de fautes ou de réseau (bande passante, latence). Dans des travaux précédents, nous avons proposé un modèle en 5 couches pour mapper des applications de grille ainsi que le réseau sous-jacent. En particulier, il montre comment les protocoles sous-jacents peuvent impacter les couches supérieures. Nous décrivons ici une bibliothèque de simulation appelée Dasor basée sur ce modèle. C’est une bibliothèque C++ pour la simulation à événements discrets qui permet de construire des simulateurs indépendamment du réseau et des modèles de simulation. Ainsi, un simulateur peut être exécuté sur différents environnements tels que des réseaux dynamiques ou des réseaux sans-fil sans aucune modification du code du simulateur. Le grain de la simulation peut ainsi être modifié simplement.

KEYWORDS: Discrete events simulation, Grid, Peer-to-peer

MOTS-CLÉS : Simulation à événements discrets, Grille, Pair-à-pair
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 define the system (network limitations, impacts of data localization or resource volatility). Several models have been already proposed and we can gather them in two categories: the models that focus on application architecture and the models that focus on material and resources. The authors of [LPP04] choose to model the network grid topology. This kind of model can highlight the bandwidth consumption and the network overload but it is independent of the grid application. In [BBL02], the authors propose a model based on the fabric notion. The components and the grid architecture are organized on several layers. But this model focuses on the grid and does not take into account subjacent protocols or mechanisms. So, in [RBF06], we propose a five layers model for grid applications that focuses both the application and the underlying protocols. Particularly, we are able to measure protocols impacts in higher layers. Figure 1 shows a simplified view of our model.

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++ [Var01] and Ns-2 [Ns-] have been designing for the oriented network applications. These solutions focus on the low-layer protocols and do not propose to simulate components of grid or peer-to-peer applications. Other tools like Narses [GB02] 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. The bottleneck problems cannot be simulated. Simulator
users\textsuperscript{1} often need to modify the granularity depending on the component tested and the previous tools cannot be easily used.

In this article, Dasor is presented: this is a C++ library for discrete events simulation based on our model. It allows the user to interact at each layer of our model. So, a simulator user can choose the suited simulation granularity by selecting the simulation models. Because simulators developed with Dasor are designed independently of the network and the simulation models, this choice can be modified without changing the simulator code.

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 description of the simulation library. First, a global description of Dasor is proposed. Then, the development of a simulator is described. Finally, the simulators execution model and the simulation models description are presented. In last section, we conclude and we present our future works.

\textsuperscript{1} In the following, a person who develops a simulator thanks to a simulation library is called a \textit{simulator developer} and a person who uses a simulator is called a \textit{simulator user}. 
2. Preliminaries

In this section, we present a state of the art of simulation tools and we describe the theoretical model we propose in [RBF06] to model a grid or a peer-to-peer application.

2.1. State of the art

2.1.1. Network simulation tools

In most cases, the grid resources are deployed in several interconnected local networks. Local networks interactions may have an impact on the grid performances, decreasing its efficiency for example. When the communications are critical (for example if a lot of data is exchanged between nodes), it is necessary to focus on the physical network view to improve the grid application efficiency.

*OMNeT++* [Var01] is a discrete events simulation library written in C++. It aims to simulate communication networks. An *OMNeT++* simulator is based on modules that communicate by message passing. To represent a network, each node can be composed of a modules stack. So, when a message is sent between two nodes, it is transparently transferred through each subjacent module. This behavior allows to simulate a protocol stack like the *TCP/IP* one. The network structure is described thanks to a specific file written in *NED* language. From this file, a compilation tool generates C++ classes of the simulator that the developer has to modify.

*Ns-2* [Ns-] also proposes to simulate network protocols. It allows to create and to test new protocols or can be used to compare similar protocols. It is based on *OTcl* scripts to describe the events or to build a network by specifying the bandwidth of each link. It is possible to select a routing protocol, a transport protocol (*TCP, UDP*) or application protocol (*FTP, Telnet*).

These tools are used to study the network protocols. They can also be used to simulate distributed applications. But the specific needs of grids are not taken into account. So, dedicated tools have been proposed.
2.1.2. Grid simulation tools

A grid middleware can be viewed as a set of components like a resources manager or a tasks manager. During the middleware designing, we have to check the compatibility of such components and the communications impacts between these components. Several tools have been developed for the specific needs of grids.

SimGrid [LMC03] is a C library to build discrete events simulators. These simulators gathers 5 elements: 1) the agents are in charge of the scheduling, 2) the locations are the places in the topology where agents are executed, 3) the tasks are either computations or data transfers, 4) the paths correspond to the paths between locations and gather communication resources, 5) the channels symbolize a communication between agents. SimGrid allows specific functionalities for the tasks management with the management of tasks dependencies.

The Java package OptorSim [BCC+02] is dedicated to the data grid simulation. During a simulation, each node of the grid gives computation and storage capabilities that can be used for task computation. A replica manager is in charge of the transfer and the duplication of data between nodes. This manager can be modified by the developer, so it is possible to compare several strategies of data replication.

MicroGrid tool [LXC04] aims to emulate virtual grids. Contrary to the previous tools, it provides to an existing application a virtual grid. The application is transparently executed over this grid. It is possible to study the behaviors of the application in function of the grid parameters (number of nodes, network properties). Other libraries like GridG [LD03] allow to create realistic grid topologies based on Internet. These topologies can be used in other simulation tools.

2.2. A model for grid and peer-to-peer applications

In [RBF06], we have proposed a model based on 5 layers for grid or peer-to-peer applications. This model is represented on Figure 1. It is used to model the subjacent network of a grid that interacts independently of the grid middleware. It also models the components of the grid middleware and interactions between them.
2.2.1. Physical network layer

The first layer concerns the physical network. The network is represented by a graph $G_1 = (V_1, E_1)$. $V_1$ is the network nodes set. A node can be an active component (like a desktop computer, a server or a parallel machine), or a passive component (like a router, a switch or a modem). $E_1$ is the set of links that interconnect the network nodes. We distinguish two kinds of links: a physical wire and a wireless connection. The physical wires are naturally undirected. But with the wireless connections, we have to take care of the different emission ranges of the network nodes. If a node has a higher range than another one, this induces a directed link in $E_1$.

Figure 2: Example of a network representation in Layer 1.

**Example 2.1** Figure 2 shows an example of a network (left figure) represented in the first layer of the model (right figure). The nodes (1 to 7) are connected with wired connections: the links in the corresponding graph are undirected. For the two wireless nodes, we can remark that Node 9 has a higher range than Node 8 (ranges are represented by a circle on the figure). We obtain a directed link in $E_1$.

2.2.2. Routing layer

Over the physical network, a routing protocol builds and maintains paths between the nodes of the network. Two entities can communicate even if they are not directly physically connected. The paths construction takes into account the several security policies deployed over subnetworks (firewalls). In the same way, some protocols (like NAT)
can limit the access to nodes. For these reasons, directed communication links are considered. In Layer 2, the network is represented by the graph $G_2 = (V_2, E_2)$, where $V_2 = V_1$ and $E_2$ is the set of paths between nodes of $V_2$.

**Example 2.2** Figure 3 is based on the example of Figure 2 and shows the representation of the network in Layer 2. To simplify, the paths that start or end from passive components (Nodes 4, 5 and 7) are not displayed. We remark new links that represent the paths computed by the routing protocol. For the wireless entities (Nodes 8 and 9), we remark that the directed link $(9, 8)$ has been deleted: Node 8 can contact Node 9 through Node 7.

![Figure 3: Example of a network representation in Layer 2.](image)

2.2.3. **Communication layer**

Over the paths built from the subjacent layer, it is possible to send data between two distant nodes that are not physically connected. The network is represented as a graph $G_3 = G_2$. In this layer, we can have the message send and receive capabilities thanks to a given protocol or a protocols stack. Several mechanisms can be proposed to manage communication problems (loss or duplication of messages, data corruption). An acknowledgment mechanism can ensure that a sent message has been received. If a message is lost, it is sent again. Another mechanism can ensure the message integrity.
2.2.4. **Topology layer**

The two higher layers focus on the grid middleware. Layer 4 is the resources management layer that can be viewed as an interface between the components and the services of the grid and the subjacent layers. In this layer, we distinguish two kinds of nodes. The first ones, called the **active nodes**, are within the grid. These nodes share their own resources or use the grid resources. The others ones, the **passive nodes** are outside the grid such as routers or switches. In this layer, the grid is represented as a graph $G_4 = (V_4, E_4)$ where $V_4$ is the set of active nodes and $E_4$ is the set of communication links between active nodes.

In this layer, the passive nodes are not represented in the graph (cf Figure 4). But they can influence the efficiency of the grid application (due to network overloads). So, it is the reason why we propose a several layers model.

![Figure 4: Example of a grid representation in Layer 4.](image)

2.2.5. **Grid components and services layer**

The last layer concerns the grid components and services including the tasks management and the resources monitoring service. The deployed components depend on the middleware: if it concerns the file sharing, it must have a component to transfer the files and maybe a component to manage access rights and queues for users.

The grid is represented by the graph $G_5 = (V_5, E_5)$ where $V_5 = V_4$ and $E_5$ is the set of communication links proposed by the topology layer. $E_5$ is not equal to $E_4$: it depends on the protocol that manages the grid topology or the peer-to-peer overlay network.
Example 2.3 For the middleware CONFIIT [FKF03], a computational peer-to-peer middleware, nodes are setup into a virtual ring in which tokens circulate (one for the topology management and one for the tasks management). Connections are established between the neighbors in the virtual ring. To be fault tolerant, a $k$-tolerance mechanism has been proposed: connections can be established with the $k$ next neighbors in the ring. In CONFIIT, $E_5$ contains only undirected links which are the links of the virtual ring and the links related to the $k$-tolerance mechanism (cf figure 5 a).

For Gnutella [Gnu], connections are established between neighbors in the grid, the neighborhoods are randomly built among the connected peers. The established connections are used to transfer the protocol messages (control and user requests). For a file transfer, a new connection is established between the file requester and the peer that owns the file. The connection is established by the file requester. But if the peer that owns the file is protected behind a firewall, the establishment of the connection can be impossible. Gnutella protocol proposes another message to ask the peer that owns the file to establish the connection. So, in the case of Gnutella, $E_5$ contains both directed and undirected links (cf figure 5 c).

![Figure 5: Example of grid representations in Layer 5.](image)

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 resources management (Layer 4) has already been focused in [RBF06, BBFR06] and tasks management in [BFR07].
3. Dasor

Dasor [DAS] is a C++ library for discrete events simulation of grid and peer-to-peer applications. The aim of Dasor is to develop simulators independently of the network and the simulation models applied at the execution. Indeed, the network and the models are only specified at the execution thanks to a network description file. The file syntax easily describes complex topologies and predefined models.

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) containing general options like the number of simulations or the simulation maximal time, a description file (.net) describing the network and the simulation models used (see next section) and eventually some data files (for resources description like tasks, files or libraries). A simulator can execute series of simulations and provides an output description file (.des) containing information about the simulation execution and a statistics file (.sta) containing registered values (more details on these files given in Section 3.6).

The execution model of a simulator developed with Dasor is based on the theoretical model as shown in Figure 6. Each simulation model is considered as a component of a theoretical model layer. 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 thanks to the input description file. By default, several models are applied and it is not necessary for the simulator user to specify the simulation models for each layer of the theoretical model. So, it is possible to modify the simulation granularity by selecting or ignoring some models. For instance, the description of a link between two nodes in the input description file can be either a physical link when a routing model is specified or a communication link without routing simulation model (by default, no routing model is applied). The simulation granularity can also be adjusted according to the applied models. Some routing models simulate the packet

2. If some models are not specified, simulation models can overlap several layers.
Figure 6: Execution model of simulators developed with Dasor based on our theoretical model.

exchanges whereas some models compute globally (in a central-
ized way) the paths between nodes. The computational power needed
to execute simulation can be reduced but with a higher simulation grain.

GNU Scientific Library [GDT+06] (GSL) has been integrated to Dasor for the random numbers generation. We propose interfaces to use the random number generators as tools to develop simulators. They can aslo be used in the input description file to configure simulation models. When a simulation model needs an integer or a real as a parameter, it is possible to specify a random number generator.

The Dasor Toolbox is a set of tools provided with the library. For instance, makeSimu assists in the creation of new simulators and makeDiag builds execution diagrams from output simulator files to show the tasks computations, the nodes failures or the messages exchanges. It also provides some statistics about the produced events during the simulator execution. convertNet tool supports the conversion of a input description file to a postscript file and can also be used to create a new input description file by removing random events of specified simula-
tion models. Then, such files can be used to test the reproducibility of results.

3.2. Input description file

An input description file contains the network description and the simulation models applied on the execution model. 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 [Wat99] and small worlds). Links can be described one by one or with a full adjacency matrix. For wireless networks, links can be ignored and computed from nodes positions. 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.

Description files also describe the models used during the simulation. Particularly, we can choose between several fault models, 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 needs as parameter both the interval for the faults lengths and the interval for the alive lengths. 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 nodes initialization or at the end of a simulation. Figure 7 shows the execution diagram of a simulation.

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

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 resources manager. This manager automates the resources 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 input description file.
3.4. Execution model of simulators

The execution model of simulators built with Dasor library is based on the theoretical model we present in Section 2.2. Figure 6 presents a simplified view of the execution model of simulators and shows how simulation models can be applied at each layer of the theoretical model. The simulation models are described in the input description file, this file is only specified at the execution of the simulator. Each applied model corresponds to a line in the description file and each type of model is specified by an unique keyword (see Section 3.5 for explanations about the selection of simulation models).

At the physical layer, several simulation models can be applied. First, Dasor proposes a set of topologies to define physical links between nodes: simple topology (star, grid, ring), small world topologies, random topologies... For wireless networks, it is possible to apply a mobility model. In this layer, it is also possible to chose a model for the physical links to define the characteristics of the links (bandwidth or latency).

Layer 2 concerns the routing and the simulator user can select among several routing protocols. But this layer can be ignored to increase the simulations grain and to reduce the computational power needed to compute the simulations. The topology described at Layer 1 is used to describe the physical links or to define the communication graph if Layer 2 is not specified. For example, starting from a star topology, a routing protocol will build paths between the nodes of each star, in order to create the communication graph. To increase the simulations grain, we can use a small world topology [WS98] in Layer 1 and ignore the routing layer. So, the communication graph is built without creation of paths.

Actually, in Layer 3, only a communication model can be specified. According to the selected model, the messages transfer can be configured. By default, if no communication model is specified, all sent messages are received without error. But, it can be possible to specify models that simulate the message loss, message data corruption and message duplication.
Layer 4 and 5 concern the grid middleware. This layer is in charge of the simulator developer: it represents the algorithms to simulate. To simplify the development of grid applications, a resources manager is proposed in Dasor. It manages the physical resources (computational power of nodes, storage capacity and memory capacity) and the logical resources (tasks, files and applications). To put a logical resource on a node, the node has to be associated to a pool of resources. A mechanism, called image, is proposed to transfer logical resources from a node to another one. Thus, a node can create an image of a file and sends this image through a message to another node. The message transfer depends on the subjacent models (routing protocol and communication model). The size of the message is computed from the size of the file image.

To describe grid resources, it is possible to describe manually the resources of each node. But, it also possible to use models to automatize the specification of resources. In that case, the resources are randomly generated with random dependencies. For example, to simulate clusters, it is possible to specify the same physical resources to nodes sets. For heterogeneous systems like Internet, it is possible to generate random resources. A logical resource model, called application model, proposes to create a set of tasks that needs an input file, an application to compute the tasks and an output file as the result of a task. The tasks, the files and the application are automatically created and can be put into the resources pools. This model can be used to simulate the behaviors of a Boinc [And04] like applications.

3.5. Simulation models

The execution model of Dasor simulators is only specified at the execution thanks to a file called description file. This file is necessary to execute a simulator since it describes the network and the used simulation models. The syntax of this file has been simplified: each line begins with “KEYWORD=” followed by parameters (a global option or the name of the model with its parameters). The description file contains the general options of the network like the number of nodes or the options about the links (wired or wireless network, directed or undi-
rected links...). Then, it is possible to specify all the simulation models that are applied during the simulator execution. The order of the models definitions in the description file is not important: each model is replaced in the right layer of the execution model. The event manager manages the models (and the associated events) according to the layers of the theoretical model.

Several simulation models have been integrated to Dasor library. For instance, some node failure models can be directly applied. To select a node failure model, the keyword is “FAILURE” followed by “=”.

– FAILURE= random p i m M (random failure model): p is the probability for a node to fail, i specifies the time interval and the failures length is randomly chosen in interval \([m, M]\). Every interval of time i, each active node fails with probability p.

– FAILURE= periodical a b c d (periodical failure model): the lifetime of each node is selected between \(a\) and \(b\) and the failure length between \(c\) and \(d\). These two values are different for each node, but they are not modified during the execution.

– FAILURE= n a b (fixed failure model): \(n\) is the number of disconnected nodes and the failure length is selected in \([a, b]\). During a simulation, the number of inactive nodes does not change: when a node wakes up after a failure, another one simultaneously fails.

When a model is selected in the execution model, a method (in the C++ class corresponding to this model) is called at the beginning of each simulation. The model is converted in a set of events called system events (on the contrary, there exist user events). The events manager of the simulator manages first the system events (these events can produce other system events or user events) according to the theoretical model: the events associated to the lower layers are managed before the events associated to higher layers. The system events are managed transparently but the actions corresponding to user events have to be specified by the simulator developer. The actions are specified in the associated methods in node classes. To limit the memory consumption of the simulator, each event has a lifetime: it can be executed only one time or several times, and it can be applied to several nodes.
Only few simple models are proposed by *Dasor* but the integration of new simulation models has been simplified. The first solution is to describe a new model by specifying each event of the model in the description file. These events are added at the beginning of the simulation in the events manager. The keyword is “FAILURE” followed by “:". This is an example to add a failure event:

\[
\text{FAILURE: node=5 time=200\%300 length=100}
\]

In this example, node 5 fails during 100 time units from time 200, then periodically each 300 time units. The file syntax allows to use directly the random number generators of the *GSL*. So, it is possible to replace the length of failures in the previous example by “\text{length=uniform 100 200}”. In this case, the length of failures is chosen uniformly randomly in the interval \([100, 200]\). In the same way, the node that fail can be selected with a random integer generator.

The second solution to describe a new simulation model is to use existing models by combining them or by adding new events. So, it is possible to choose the mobility model called *random walk* (all nodes move like a random walk [BS07]) and to specify the behavior of a node with an event to move this node through a predefined path.

The last solution is to write directly new models in the library (it needs a partial compilation of *Dasor*). Each model is defined with a C++ class that inherits from a virtual class. To write a new model, we have to describe it by a list of events (single or periodical). The list content is added to the events manager at the execution beginning. The creation of new classes is achieved independently of the theoretical model. The events manager is in charge of the events management according to the layers of the model.

### 3.6. Parameters and statistics

As we described previously, *makeSimu* tool helps the developers to create new simulators. It creates the C++ classes needed for a simulator.

---

3. Each node is identified by an integer from 0 to \(n - 1\) where \(n\) is the number of nodes.
4. There is no time unit in the simulator. The time is defined as a real, that gives a total freedom to the developer and the users. The simulation models have to be configured coherently.
Particularly, it generates a main file with the “main” function. In this method, the parameters specified through the command line to the simulator are analyzed. The parameters contain the global options needed to specify the number of simulations (by default, only one simulation is executed), the name of the input description file (it is the only option that must be specified), the name of the output files (description and statistics files) and others options. There are also the user parameters. An user parameter is added by the simulator developer to configure the simulator. The developer specifies the name of the parameter, the minimum and maximum ranges for its values (if the parameter is a value), its default value and an help message for the simulator user (this message is displayed when an error occurs before the simulation starts). The user parameters avoid the multiple compilations of a simulator.

To obtain statistics or to analyze the results of simulation series, a simulator can create a statistics file and an output description file.

In the main function, the developer can register some variables or functions. At the end of each simulation, the registered values are saved in the statistics file. This file can be used to create figures thanks to a tool like GnuPlot.

If the user specifies a specific option, the simulator creates an output description file that contains all the events generated during the execution of the simulator and all the simulation models applied. To extract data from this file, the user can use makeDiag tool. With this tool, it is possible to create several diagrams: tasks computations, nodes failures or messages exchanges.

4. Conclusion

In this article, we introduce Dasor a C++ library for building discrete events simulators of grid and peer-to-peer applications. The execution model of simulators is based on a five layers theoretical model. Simulators are developed independently of the network and the simulation

5. The variables and functions that can be registered must be static.
models, specified as parameters at the execution. So, the simulator users can select the appropriate simulation grain without modification of the simulator code.

Our future works will focus on the development of new simulation models for Dasor especially fault models that take into account the lifespan of nodes. We have already developed a tool to compute simulations on a parallel machine. We plan to integrate the simulator to a grid middleware to compute simulations in parallel.

We also plan to create a graphic interface for the simulators users. From the output description file, it is possible to extract the simulation events and we propose to use them to create animations of the execution.

References


[BFR07] Alain Bui, Olivier Flauzac, and Cyril Rabat. Fully Distributed Active and Passive Scheduling for Grid Comput-


ANNEXE POUR LA FABRICATION
A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE
PAPIER
DE LEUR ARTICLE

1. ARTICLE POUR LA REVUE :
   Studia Informatica Universalis.

2. AUTEURS :
   Cyril Rabat

3. TITRE DE L’ARTICLE :
   Dasor, a Discrete Events Simulation Library for Grid and Peer-to-Peer Simulators

4. TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :
   Simulation Library Dasor

5. DATE DE CETTE VERSION :
   February 24, 2009

6. COORDONNÉES DES AUTEURS :
   – adresse postale :
     SysCom, CReSTIC
     Université de Reims Champagne-Ardenne
     BP1039, F-51687 Reims Cedex 2, France
     cyril.rabat@univ-reims.fr
   – téléphone : +33 26 91 32 04
   – télécopie : +33 26 91 33 97
   – e-mail : cyril.rabat@univ-reims.fr

7. LOGICIEL UTILISÉ POUR LA PRÉPARATION DE CET ARTICLE :
   LATEX, avec le fichier de style studia-Hermann.cls,
   version 1.2 du 03/12/2007.