Declarative Task Delegation in OpenMOLE

Romain	Florent	Mathieu	Thierry	Nicolas	David
Reuillon	Chuffart	Leclaire	Faure	Dumoulin	Hill
ISC-PIF	*LIP*	*ISC-PIF*	*LISC*	*LISC*	*LIMOS*
CNRS	*INRIA, ENS*	*CNRS*	*Cemagref*	*Cemagref*	*CNRS, UBP*

romain.reuillon@iscpif.fr

ABSTRACT

In this paper we present OpenMOLE, a scientific framework providing a virtualized runtime environment for distributed computing. Current distributed execution systems do not hide the hardware and software heterogeneity of computing and data resources whereas OpenMOLE provides generic services to develop distributed scientific algorithms independently from the execution environment architecture. OpenMOLE uses abstraction layers to delegate computing tasks with the same high level interface for the major underlying architectures: local processors, batch systems, computational grids, Internet computing and cloud computing. The file access abstraction layer is another key feature helping a generic usage of the computation power provided by grids and clusters. The OpenMOLE framework has been tested with the exploration of a bacterial biofilm simulation with an individual-based model.

KEYWORDS: Task Delegation, Grid Computing, High Perfomance Computing, Distributed Simulation Exploration, Design of Computer Experiments.

1. INTRODUCTION

The computational grid concept was originally presented as a metaphor of the US power grid [1]: a global infrastructure where end-users can access computing power on demand. But this concept clashes with reality. Indeed, end-users are faced with heterogeneity of the execution environments. Power grid has to deal with potential difference and electricity flow, whereas computational grid has to deal with not only network and computing power availability considerations, but also higher level concepts like operating systems, processor architectures and available applications. Even if grid infrastructures are able to take advantage of storage and computational power through the grid middleware, a fine-grain analysis is still necessary at the user level to achieve a successful execution on such an environment.

The main object of this paper, is to describe distributed computing facilities within OpenMOLE. OpenMOLE is a scientific framework, which aims at providing a virtualized runtime environment. In this approach the user can be confident that an experiment successfully executed on a local computer will be successfully executed on other execution environments without preliminary tests or software installation.

This paper is organized as follows. Section one describes existing tools to execute jobs on large scale grid computing and clusters. The lack of existing tools compared to our main goal are pointed out. Section two presents novel methods implemented in OpenMOLE providing a transparent access to various distributed computational environments.

2. RELATED WORK IN DISTRIBUTED EXECUTION MANAGEMENT TOOLS

Batch systems such as openPBS[1], Condor[2] or Sun Grid Engine[3] (SGE) and grid middleware like globus [2], gLite [3] or DIET [4] provide integrated commands and/or specific Application Programming Interfaces (API) to submit and manage jobs as well as transfer data.

[1] http://www.pbsgridworks.com/
[2] http://www.cs.wisc.edu/condor/
[3] http://gridengine.sunsource.net/

978-1-4244-6830-0/10/$26.00 ©2010 IEEE

Even though these distributed execution systems share many common concepts, the set of commands provided by each of them is specific. In this context, using abstraction layers is a key issue to easily benefit from the distributed computing power independently of the underlying architecture. A well designed generic abstraction layer for computational resources should allow the use of local processors, batch systems, computational grids, Internet computing and cloud computing through the same high level interface.

2.1. Ganga

Ganga [5] is a Python application for job management. Ganga allows local and distributed executions on many grid middleware and batch systems based on: openpbs, lsf (distributed by Platform[4]), dirac [6] and gLite [3].

Even though Ganga provides a generic job management on many cluster and grid environments, it does not provide a full support for accessing grid storage elements and file servers. Ganga enables the user to define input and output files for jobs. This functionality fits the "bag-of-task" [7] paradigm, in which a job is transfered jointly with data it refers to in an object called input sandbox. However in many real distributed simulation cases, jobs need to access large files. In this case, this voluminous file should not be mentioned in the input sandbox. Otherwise the file would be uploaded whenever the job is launched instead of being stored once and for all on a storage element.

In our experience, this missing functionality prevents designing generic jobs. Indeed, the lack of generic file access interface necessitates writing specific (for a given execution environment) file access instructions in the job executable code. Therefore, the file access abstraction layer is another key feature for generic usage of the computation power provided by grids and clusters.

2.2. G-Eclipse

G-Eclipse[5] [8] is an integrated workspace environment for accessing grid computing power. It is based on Eclipse[6]. G-Eclipse provides high-level abstractions for grid resources such as job management services, authentication services and information services. This abstraction layer is adaptable for various kinds of distributed execution environments. Currently, the g-Eclipse project supports implementations for two grid middleware systems: gLite[7] [3] and gria [9] and one cloud computing service: Amazon EC2 (Elastic Compute Cloud)[8]. These three environments are based on similar concepts but on radically dissimilar architectures, illustrating the genericity of the g-Eclipse middleware abstraction layer.

Besides the generic access to several grid systems, g-Eclipse provides an implementation of the Eclipse filesystem (efs) to access grid storages. efs is an abstraction layer in which files are referenced by Uniform Resource Identifier (uri)s [9]. Actual file accesses are achieved using dynamically registered modules containing implementations of the file system access protocols. Various implementations of efs are available, among them: SFTP, HTTP, CVS. Implementations for gLite [3] file access protocols have been developed by the g-Eclipse project, including the following protocols: GridFTP (the grid file transfer protocol) and srm (Storage Resource Management). An implementation for the on-line storage Amazon S3 is also available.

G-Eclipse provides very useful classes to interact with various execution environments in a generic way. However its development has been more focused on building a nice graphical interface interacting with the grid than a nice application programming interface (API). A consequence for developers is that further development based on it, might be tricky.

2.3. JavaGAT and JSAGA

Java Grid Application Toolkit (JavaGAT) [10] is a Java API designed to abstract various distributed execution environments. It is designed as an extensible environment in which modules for accessing computing platforms can be plugged. Modules, also called "adaptors", for execution systems (such as gLite, Globus, SGE, SSH) and for remote file system access (such as HTTP, FTP, SFTP or GridFTP) are currently available. On this work basis, the Simple API for Grid Applications (SAGA) [11] generalizes the JavaGAT architecture by providing a language-independent specification for accessing distributed execution environments. Several implementations of SAGA have been achieved in C++, Python and Java. This approach, endorsed by the Open Grid Forum (OGF), is the most promis-

[4] http://www.platform.com/
[5] http://www.g-eclipse.eu/
[6] http://www.eclipse.org/
[7] http://glite.web.cern.ch/glite/
[8] http://aws.amazon.com/ec2/
[9] URIs are normalized in the RFC3986, they allow identifying resources on a network such as servers, web services or files

ing we have seen for a transparent access to remote computing and storage resources.

3. OPENMOLE

Tools such as G-Eclipse, Ganga or JSAGA simplify the access to distributed execution environments, however they do not hide the hardware and software heterogeneity of computing resources. From our previous work on grid computing [12, 13], this drawback prevents developing distributed scientific algorithms independently from the execution environment architecture. In order to tackle this problem, OpenMOLE takes advantage from generic interface of JSAGA and provides, on top of that, a virtualized runtime environment as shown on Figure 1. Transparent delegation of scientific algorithms to remote environment becomes possible within OpenMOLE. Indeed, virtualizing guarantees a successful achievement of the remote execution.

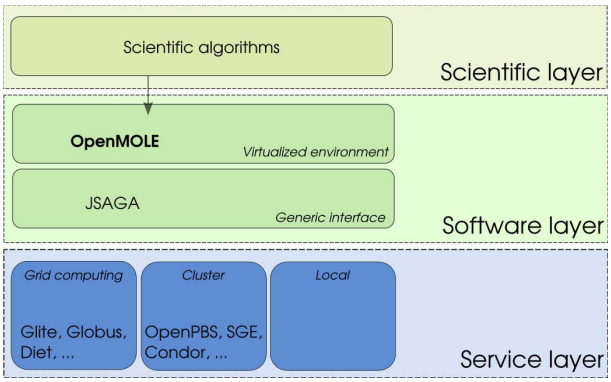

Figure 1. Layer Architecture of the OpenMOLE Framework.

3.1. Execution Environment Constraints

OpenMOLE is a framework providing distributed computing facilities. It has been designed to work out of the box on the user desktop with the idea of completely hiding the fact that computation may be carried out on distributed environments. To achieve this goal one has to respect certain constraints:

1. the end-user must be able to define and integrate his own software components in OpenMOLE, which should be able to execute them remotely,
2. OpenMOLE should not rely on preliminary installed software on the target environment,
3. OpenMOLE should be autonomous in accessing various remote execution environments, without requiring third party servers,
4. the end-user's desktop computer can be connected to the Internet without being necessarily the owner of a public IP address (Internet access via a Network Address Translation (NAT) system is enough).

Constraints 1 and 2 have been established so that the user can work on his own algorithms, without knowing how to deploy or execute them on various execution environments. OpenMOLE should provide means to integrate external programs, which can then be executed directly and with confidence on targeted execution environments despite the hardware and software heterogeneity.

Constraint 3 ensures the scalability and the user-friendliness of the application. No central server is required by OpenMOLE in between the user desktop computer and any targeted execution environment. Furthermore, the user desktop computer may not itself act as a server because of constrain 4: actually user computers are usually behind a NAT.

Those constraints should be taken into account in our framework in order to make the execution environment transparent for the end-users. This gives rise to major problems:

- How can we address the portability issue of user defined software components on heterogeneous execution environments?
- How can we access various execution environments in a generic way without relying on a third party server?

3.2. Portability of User Defined Software Components

In order to tackle this problem, OpenMOLE provides generic services implemented in the "Task" class. A task is executed in a specific Context and uses some given Resources. As exposed on Figure 2, the "Task" class has been defined including concepts of execution context and required resources. This design makes it possible to migrate and execute tasks remotely. The "Task" class is polymorphous and each specialization of "Task" is provided as a new service. Tasks use resources and are executed in an execution context. The resources can be files, libraries or software pieces,

which must be available at runtime. The context contains the variables that are used by the task at runtime. Each context matches particular execution conditions.

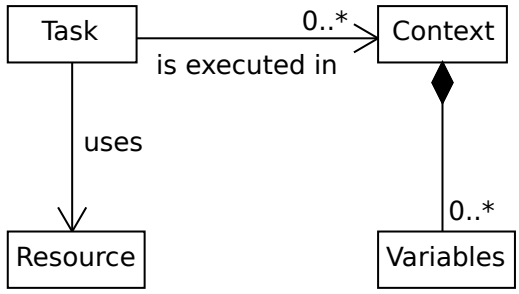

Figure 2. UML Class Diagram for the Task Class.

By default OpenMOLE provides a set of predefiened tasks. These tasks are designed to support external resources such as Java jar libraries or Octave[10] script files. This way third party software pieces can be introduced in the OpenMOLE framework.

Since tasks can be migrated, they should be portable from one environment to another. In OpenMOLE, the heterogeneity of the various execution environments is mostly addressed by using the Java Virtual Machine (JVM) virtualization system. To our mind, virtualization is the right way to support task portability. For such purpose, the JVM is a mature virtualization system. Furthermore it is:

- executable in user space (no need to get special rights or special operating system kernel to execute it),
- compatible with most current operating systems,
- fast,
- available as free (as in speech) software[11].

For tasks calling binary x86 code, several versions of the same executable files can be defined as resources. The appropriate version of the resource is automatically selected at runtime. This solution works but is not sustainable in the long term. Binary software pieces should be packaged for each execution environment. The packaging stage may result in problems that will be only encountered at runtime. In that case, it is clear that this solution does not solve the problem of remote execution in a transversal and reliable manner. In the future we plan to address the problem of execution of binary code on remote environments by using full hardware virtualization on top of the JVM. The Java library JPC[12] provides such solutions.

3.3. Delegation of the Tasks

In the previous part (3.2) we presented a solution for delegating a task from one computer to another. However the problem of accessing the distributed execution environment from the user desktop still remains. In this part a method for task delegation taking into account constraints described in Section 3.1. is presented. This method has been designed to run on any environment containing at least both a file storage and a job submission system.

This method is shown in Figure 3. The first step consists in storing the required files for the remote execution, meaning:

- an archive, called "runtime", containing the Java Virtual Machine and the OpenMOLE framework,
- a "Task" object as well as its execution context serialized in a file using the XStream[13] library,
- the files required to deploy the resources on the target environment.

Once this steps have been completed, a job is transmitted to the submission system. Let us specify that a job contains the references of the previously uploaded files and a location where it will copy the results at the end of the process. At the very beginning of its execution, the job prepares the environment on the computing node: it downloads the runtime, deserializes the task and its execution context, downloads and deploys the resources. Then it runs the task in that context. Doing that, the context is modified and output files are produced. Finally, it stores those outputs back on the storage system at the predefined location. During job execution, the OpenMOLE framework tracks its state. Once it is finished the results are downloaded and made available on the local computer for the user or for another task.

Job submission and data transfers within OpenMOLE benefits from the abstraction layers of JSAGA. On top of that it provides both transparent portability of task

[10] http://www.gnu.org/software/octave/
[11] http://openjdk.java.net/
[12] http://www-jpc.physics.ox.ac.uk/home_home.html
[13] http://xstream.codehaus.org/

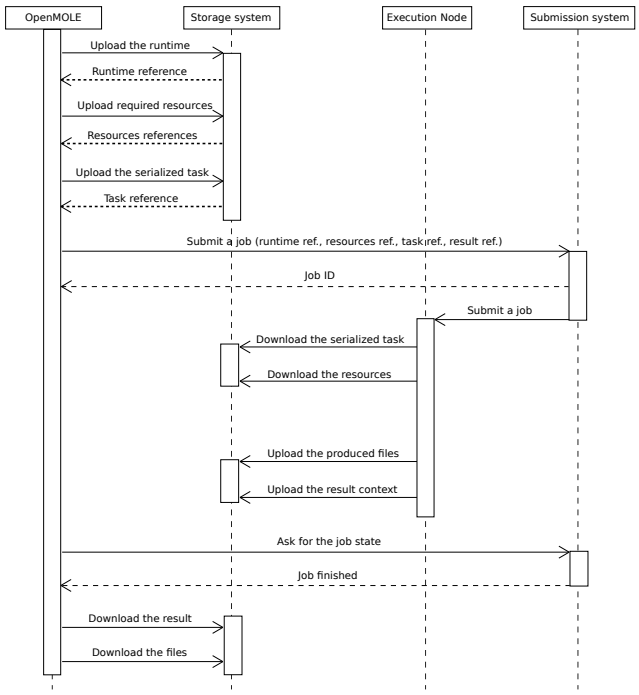

Figure 3. UML Sequence Diagram of the Delegation of a Task in a Distributed Execution Environment.

and transparent access to various execution environments. By doing so, OpenMOLE implements the functionality to switch task execution from one execution environment to another in a declarative way.

The delegation mechanism of task execution described in this section is fully implemented in the OpenMOLE platform. It is furthermore refined with advanced functionalities such as transparent compression/decompression of files during transfer operations, automatic files integrity checking by computing/verifying hash values, automatic task resubmission in case of failure, multi-threaded transfer of the files in background. This method has been extensively tested on the EGEE[14] grid and is being currently implemented for SGE and for remote servers accessible via SSH.

3.4. A Small Example

By design, it is straight-forward in OpenMOLE to switch from a local task execution to a remote one. Between those two options a only few changes are made. For instance, lines of code below expose how to implement in OpenMOLE a task which launches a Java model. The model is contained in a Java jar archive.

[14] http://www.eu-egee.org/

It requires a file parameter and produces an output file named "result.txt".

```
/* - Local machine execution - */
JavaTask model = new JavaTask();

model.addJarResource(
       "/home/user/model.jar")

model.addFileResource("file",
       "/home/user/file.txt")

model.setCode("Model m =
                  new Model(file);\n"+
              "m.run()\n");

model.addOutputFile("result.txt");

LocalEnvironment.submit(model);
```

First a "JavaTask" object called "model" is instantiated. Then two resources are associated to this object: a jar resource and the file parameter. The file parameter is referenced by the variable named "file". The Java code for launching the model is then mentioned. This code uses the variable "file" which points at the file parameter. After that, the task is set to retrieve the output file named "result.txt". At the end of the code, the task is executed on the local computer.

The code bellow exposes the additional code required to execute the task "model" on the virtual organization (VO) "vo.iscpif.fr" of the EGEE grid. An instance of the class "GliteEnvironment" is instantiated by providing the name of the VO, the URL of the authentication service and of the URL of the information service. After that, the task is parameterized to be executed on the grid.

```
/* - Additional code for grid execution - */
GliteEnvironment env =
    new GliteEnvironment("vo.iscpif.fr",
          "voms://grid12.lal.in2p3.fr:20013"
        + "/O=GRID-FR/C=FR/O=CNRS/OU=LAL/"
        + "CN=grid12.lal.in2p3.fr",
          "ldap://topbdii.grif.fr:2170");

env.submit(model);
```

3.5. Real Explorations Using OpenMOLE

In addition to the task delegation mechanism, OpenMOLE provides a workflow engine for specifying complete scientific experiments in a naturally parallel manner and independently from a specific execution environment. OpenMole task delegation mechanism described in this paper has been involved in the explo-

ration of complex-system simulation models from various application fields, among which:

- the exploration of an individual-based bacterial biofilm model [14]: OpenMOLE allowed distributing Octave batches on a cluster operated by the PBS batch system,
- the characterization of the dynamics of a predators-prey simulation model [15] for 360 000 combinations of values for 5 input parameters of the model,
- the computation of a viability tube during the French project INCALIN[15] for a Camembert cheese mass-loss model [16].

In the detail, the computation of viability tubes aimed at studing the dynamics of a Camembert cheese mass-loss model using the viability theory [17] and the algorithm described in [18]. The model under study is a controllable model as discribed in figure 4. The model has two kinds of inputs: a state (the weight of the cheese, the micro-organisms' activity...) and a control (the conditions for temperature and humidity in the ripening chamber, which can be modified in order to modify the dynamic of the ripening process). The model computes a new state of the cheese the next day depending on the initial state and control for this day.

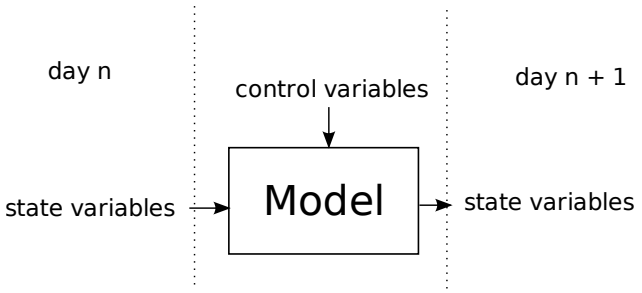

Figure 4. The Controllable Model.

The viability algorithm aims at finding all the dynamics of the model that lead, at the end of the process, to states which respect a set of constraints. An example of constraints is that the ripened cheese should not weight less than 250grammes for legal reasons and should not weigh more than 270 grammes to fit the packaging.

The global idea of the viability tube computation is to start from the last day of the process and compute the tube backwards. Let us call the last day of the ripening process the day n. First, the states of day n respecting the set of predefined constraints are computed; it is called the target of the day n. Then, the algorithm explores all possible states of the day $n-1$. For each of them, it runs the model with all the possible controls. If it finds at least one control allowing to reach the target the state is considered as viable and is added to the target the day $n-1$. The algorithm is iterated until day 1 using the successive targets.

The figure 5 shows an OpenMOLE's workflow for distributing the viability tube algorithm computation. The first task explores the space of the states for the current day, called day d. It launches an instance of the second task for each state. The second task takes a state and the target of day d as input. It explores, for the given state all the possible controls and produces the list of controls for which the resulting state of the model belongs to the target of day d. If the list is empty it means that the input state can not be considered as viable. When all the instances of the second task have been executed, the results are aggregated by the third task and the target of day $d-1$ is computed. The value of d is decreased. The process is iterated with the new target. This workflow ends when the current d is no longer greater than 0.

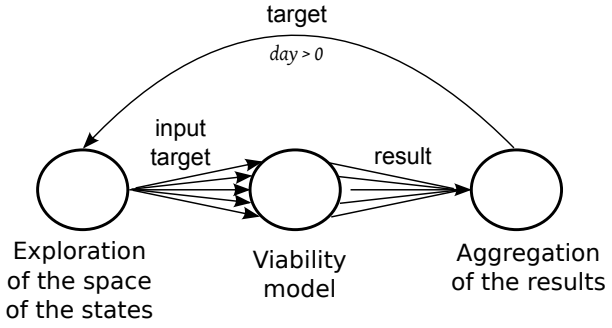

Figure 5. Example of Work ow Used for Computing a Viability Tube.

Computing a viability tube using this method involves running the model 74 million times. The total execution time is 1 year of computation on a single modern computer. Fortunately, in OpenMOLE the task executions can be easily delegated to a distributed environment. For this experiment, executions of the second task have been delegated to the EGEEgrid. The tube has been computed in only a few hours. Once the workflow is running on a single computer, the complexity of grid execution delegation is entirely hidden by

[15] http://incalin.csregistry.org

OpenMOLE, that implements algorithms to efficiently execute batch of jobs on a grid.

4. CONCLUSION AND PERSPECTIVES

This paper first exposes some of the available frameworks simplifying the access to distributed execution environments. These frameworks do not bring a solution to all the difficulties that a scientist encounters when he tries to execute his own algorithm on a distributed computation environment. In particular they don't propose methods and tools:

- to port user defined software components from one environment to another,
- to fully hide the technical and methodological aspects of computation environments.

OpenMOLE addresses both of them. By doing so, it uncouples the scientific business layer (defined in OpenMOLE's tasks) from the execution layers. For now, OpenMOLE proposes tasks to execute Java code, binary code, to manipulate files and many others... Even if those tasks might be used to describe lots of computational operations in diverse scientific fields they cannot claim to be exhaustive. The possibility of integrating new task types defined by advanced users through an OpenMOLE extension API is being worked on. It will turn OpenMOLE into a fully modular and extensible platform.

A graphical user interface (GUI) will shortly enable setting task parameters in a user-friendly environement. The GUI will also simplify the use of the workflow functionalities. As already stated, the execution of the tasks can be linked with each other through a complete workflow system in order to describe complete scientific numerical experiments. By this means, the scientific logic is uncoupled from any execution environment and ready for distributed execution. To optimize this execution, optimal tasks scheduling among all the accessible execution environments can be achieved. This work may lead to taking advantage from the accessible execution environments of different kinds and getting the best of them. For instance it would be possible to achieve fast computations by scheduling jobs optimally between a grid, with huge computational power a high latency, and a symmetric multi-processor server, with low latency but small computational power.

The last point we are working on, is the collaborative design of workflow libraries proposing meaningful workflows. Those workflows could implement various scientific algorithms such as: optimization by genetic algorithms, parallel execution of the replications of stochastic simulations or viability process in a reusable way. By providing the possibility to achieve community work within OpenMOLE, scientists could be able to work in a collaborative way using, creating, modifying and executing OpenMOLE workflows.

5. ACKNOWLEDGEMENTS

This work is supported by the European projects Patres (nest-043268)[16] and DREAM[17], as well as the projects LifeGrid (feder)[18] and INCALIN[19]. Special thanks to the g-Eclipse and the JSAGA teams for their help.

REFERENCES

[1] C. Kesselman and I. Foster, THE GRID: BLUEPRINT FOR A NEW COMPUTING INFRASTRUCTURE. Morgan Kaufmann Publishers, November 1998.

[2] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit.," Intl J. Supercomputer Applications, vol. 11, no. 2, pp. 115–128, 1997.

[3] E. Laure, S. M. Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini7, F. Prelz, J. White, M. Barroso, P. Buncic, F. Hemmer, A. D. Meglio, and A. Edlund, "Programming the grid with glite," Computational Methods In Science and Technology, vol. 12, no. 1, pp. 33–45, 2006.

[4] E. Caron and F. Desprez, "Diet: A scalable toolbox to build network enabled servers on the grid," International Journal of High Performance Computing Applications, vol. 20, no. 3, pp. 335–352, 2006.

[5] U. Egede, K. Harrisonb, R. Jonesc, A. Maierd, J. Moscickid, G. Patricke, A. Sorokof, and C. Tang, "Ganga user interface for job definition and management," in Proceedings of the Fourth International Workshop on Frontier Science: New Frontiers in Subnuclear Physics, pp. 367–374, 2005.

[6] E. Caron, V. Garonne, and A. Tsaregorodtsev, "Definition, modelling and simulation of a grid computing scheduling system for high throughput computing," Future Gener. Comput. Syst., vol. 23, no. 8, pp. 968–976, 2007.

[7] G. R. Andrews, "Paradigms for process interaction in distributed programs," ACM Comput. Surv., vol. 23, no. 1, pp. 49–90, 1991.

[16] http://www.patres-project.eu
[17] http://dreameu.vipros.eu
[18] http://www.lifegrid.fr
[19] http://incalin.csregistry.org

[8] P. Wolniewicz, N. Meyer, M. Stroinski, M. Stmpert, H. Kornmayer, M. Polak, and H. Gjermundrd, "Accessing grid computing resources with g-eclipse platform," Computational Methods in Science and Technologie, vol. 13, no. 2, pp. 131–141, 2007.

[9] M. Surridge, S. Taylor, D. D. Roure, and E. Zaluska, "Experiences with gria industrial applications on a web services grid," in Proceedings of the First International Conference on e-Science and Grid Computing, pp. 98–105, 2005.

[10] R. V. van Nieuwpoort, T. Kielmann, and H. E. Bal, "User-friendly and reliable grid computing based on imperfect middleware," in SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, (New York, NY, USA), pp. 1–11, ACM, 2007.

[11] S. Jha, H. Kaiser, A. Merzky, and O. Weidner, "Grid interoperability at the application level using saga," e-Science and Grid Computing, International Conference on, vol. 0, pp. 584–591, 2007.

[12] L. Maigne, D. Hill, V. Breton, R. Reuillon, P. Calvat, D. Lazaro, Y. Legres, and D. Donnarieix, "Parallelization of monte carlo simulations and submission to a grid environment," Parallel processing letters, vol. 14, no. 2, pp. 177–196, 2004.

[13] R. Reuillon, D. Hill, Z. E. Bitar, and V. Breton, "Rigorous distribution of stochastic simulations using the distme toolkit," IEEE Transactions on Nuclear Science, vol. 55, no. 1, pp. 595–603, 2008.

[14] J.-D. Mathias, F. Chuffart, N. Mabrouk, and G. Deffuant, "Applying a moment approximation to a bacterial biofilm individual-based model," in ICCGI '09: Proceedings of the 2009 Fourth International Multi-Conference on Computing in the Global Information Technology, (Washington, DC, USA), pp. 138–143, IEEE Computer Society, 2009.

[15] F. Chuffart, N. Dumoulin, T. Faure, and G. Deffuant, "SimExplorer: Programming Experimental Designs on Models and Managing Quality of Modelling Process," International Journal of Agricultural and Environmental Information Systems, vol. 1, p. to be published, Janvier 2010.

[16] M. Sicard, S. Martin, R. Reuillon, S. Mesmoudi, I. Alvarez, and N. Perrot, "The viability theory to control complex food processes, application to camembert cheese ripening process," **submited** to Journal of Food Engineering, 2010.

[17] J. P. Aubin, VIABILITY THEORY. Birkhuser, 1991.

[18] P. Saint-Pierre, "Approximation of the viability kernel," Applied Mathematics and Optimization, vol. 29, pp. 187–209, 1994.