Leveraging business workflows in distributed systems research for the orchestration of reproducible and scalable experiments

Tomasz Buchert, Lucas Nussbaum

MajecSTIC 2012
Lille, France, du 29 au 31 octobre 2012

Leveraging business workflows in distributed systems research for the orchestration of reproducible and scalable experiments

Tomasz Buchert[1] and Lucas Nussbaum[2]

1: INRIA Nancy - Grand Est.
2: Université de Lorraine.

Contact: tomasz.buchert@inria.fr

Résumé
Malgré une activité de recherche sur les systèmes distribués très importante et très active, les expériences dans ce domaine sont souvent difficiles à concevoir, décrire, mener et reproduire. Surmonter ces difficultés pourrait permettre à ce domaine d'être encore plus stimulé, et aux résultats de gagner en crédibilité, à la fois dans le domaine des systèmes distribués. Les facteurs principaux responsables de cette situation sont techniques (bugs logiciels, problèmes matériels), méthodologiques (mauvaises pratiques), et sociaux (réticence à partager son travail). Dans cet article, les approches existantes pour la description et la conduite d'expériences sur les systèmes distribués sont décrites, et une nouvelle approche, utilisant le *Business Process Management (BPM)*, est présentée pour répondre à leurs limitations. Puis diverses questions se posant lors de l'utilisation d'une telle approche sont discutées. Nous montrons que cette approche peut être une meilleure alternative à la manière traditionnelle de conduire des expériences, qui encourage de meilleures pratiques scientifiques, et qui résulte en une recherche et des publications de meilleure qualité. Pour finir, notre plan de travail est décrit, et des applications possibles de ce travail dans d'autres domaines sont décrites.

Abstract
While rapid research on distributed systems is observed, experiments in this field are often difficult to design, describe, conduct and reproduce. By overcoming these difficulties the research could be further stimulated and add more credibility to results in distributed systems research. The key factors responsible for this situation are technical (software bugs and hardware errors), methodological (incorrect practices), as well as social (reluctance to share work). In this paper, the existing approaches for the management of experiments on distributed systems are described and a novel approach using business process management (BPM) is presented to address their shortcomings. Then, the questions arising when such approach is taken, are addressed. We show that it can be a better alternative to the traditional way of performing experiments as it encourages better scientific practices and results in more valuable research and publications. Finally, a plan of our future work is outlined and other applications of this work are discussed.

Mots-clés : expérimentation, validation expérimentale ; démarche scientifique ; workflows scientifiques ; expériences à grande échelle

Keywords: experimentation; experimental validation; scientific method; scientific workflows; large-scale experiments

---

1. Introduction

Nowadays, a large part of research in computational sciences relies on complicated software stacks to drive experiments and obtain results. As they become more and more complex, their role in the scientific discovery process must be reconsidered. In fact, computer programs are to computational science what an experiment description is to physics, biology, geology and other

natural sciences. Without a detailed description, researchers cannot obtain statistically equivalent results and draw the same conclusions. In other words, the research is not *reproducible*.
Surprisingly, the research community has no culture of sharing computer programs [5]. The two main causes of reluctance to distribute source code are: the lack of training of scientists to write high-quality software and the fear of losing further publication opportunities. The first reason can be addressed by changes in the curriculum and the second is probably unfounded. Fortunately, some scientific publishers (e.g., Science [1]) require publication of the software used to obtain the results along with the results themselves.
Even the access to the source code may not be enough. An example is research on *distributed systems*, i.e., systems "in which the failure of a computer you didn't even know existed can render your own computer unusable" (L. Lamport [2]). The problem stems from the fact that they are nondeterministic and more complex, erroneous and difficult to control than centralized systems. *Large-scale distributed systems* span different networks and geographical entities, and consist of thousands of heterogeneous machines with different configurations. Thus, experiments on this scale suffer also from *scalability* problems, that is, problems encountered when resource limits (e.g., number of connections, network speed) of the infrastructure are exercised.
These systems cannot be controlled manually due to their complexity. To address this, *orchestration* automates control and coordination of complex computer systems to build a usable system. Orchestration of large-scale systems, including experiments involving such systems, is a very difficult problem. It becomes more important as computer systems grow in size.

The main purpose of this work is to improve current state-of-the-art methodologies and tools, making the research on large-scale distributed systems easier to carry on, more robust and ultimately, more trustworthy. To this end, we present a new approach to attack the problem of orchestrating large-scale experiments using *Business Process Management* (BPM). It consists in a workflow-based design of experiments, coupled with a software solution that supports execution of those experiments (experiment orchestration engine).

The rest of the paper is structured as follows. Section 2 is a description of goals and of the requirements that the solution must meet. In Section 3, prior work is presented and analyzed in the context of the previous section. Section 4 details our approach to the problem of reproducible and scalable experiments in large-scale distributed systems research. We discuss how Business Process Modelling can be useful, present our research plan and discuss non-academic applications of our work. Finally, in Section 5, our work is concluded.

Our further discussion is illustrated by two examples of large-scale experiments:

gLite experiment Analysis of the gLite [3] (Lightweight Middleware for Grid Computing) component: on a deployed instance of gLite (500 nodes), the scalability (ability to handle many concurrent requests) of TORQUE (batch scheduler) is assessed.

MPI experiment Performance and energy consumption of an MPI application under various parameters: MPI runtime (OpenMPI, MPICH2), number of nodes, OS configuration and version, communication medium (Ethernet vs. Infiniband), architecture (CPU vs. GPU), etc.

The first experiment is difficult to design and perform because the gLite middleware is a complicated software stack consisting of many interconnected services that are deployed on many nodes that must be set up and controlled during the execution. It is a time-consuming process during which even a single error can bring down the whole experiment.
The second experiment poses a different problem. In this large-scale experiment, the application is tested under a large set of different environment configurations. First, an optimal design of the experiment is crucial - otherwise it may take too much time. Moreover, without proper error handling the errors happening during the experiment would require manual re-executions of the experiment.

1. http://www.sciencemag.org/site/feature/contribinfo/prep/gen_info.xhtml
2. http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html#distributed-system
3. http://glite.cern.ch

2. Goals and requirements for an experiment engine

For the sake of our analysis, a computer experiment is defined as a procedure, formalized in a form of computer programs, that verifies a hypothesis posed by the experimenter. It means that through logical reasoning about the behavior of computer program execution and obtained data, the truth of the hypothesis can be determined. An experiment engine is a piece of software that helps with the design, execution and monitoring of the experiment.

A computer experiment consists of computer programs, therefore all metrics applying to computer programs and systems are meaningful and important to the experiments as well. In particular, the computer experiment requires computing power, memory, storage, network bandwidth and other resources. Hence, the execution of the experiment depends on available resources.

Now, we define two desirable properties of an experiment that we focus on in this paper:

Reproducible experiment An experiment is *reproducible* if it has an unambiguous description that allows for independent derivation of the same conclusions; note that the raw results (contrary to the conclusions) need not to be identical, they merely must fit within a statistical margin or exhibit the same property so that the conclusions are the same.

Scalable experiment A computer science experiment is *scalable* if it involves a parametrized, usually large, number of resources (machines, data, memory, etc.) and if, with the increase of these resources, the experiment execution is still correct, free of failures and can be conducted with a limited number of resources and time. In particular, the unavoidable exceptional situations must be handled properly and the amount of time needed to prepare or control the experiment must not increase drastically.

To meet the defined properties, the experiment description and its execution process must possess a certain set of features that can be grouped by the part of the experiment they help with: design (*Descriptiveness*, *Modularity*, *Reusability*, *Maintainability*, *Support for common patterns*), execution (*Snapshotting*, *Error handling*, *Integration with lower-level tools*, *Human interaction*) and monitoring (*Monitoring*, *Instrumentation*, *Data analysis*).

Descriptiveness The experiment is described in a top-down approach. The details are available, but not necessary to understand the experiment. Descriptiveness allows to ignore the details and concentrate on the experiment itself. Moreover, it lowers the bar for the newcomers.

See Figure 1 for a high-level depiction of *gLite experiment* deployment.

Modularity The building blocks of the experiment can be easily removed, replaced and added to the experimental process. That way the experiment can be logically split into modules with well-defined interfaces that can be worked on independently.

In the *MPI experiment*, for example, the underlying testbed can be replaced easily.

Reusability Well-separated parts of the experiment can be used independently from each other and reused by other researchers. This limits code duplication (and possibility of bugs), accelerates development and eases collaboration (by building on the work of others).

For instance, the deployment part of *gLite experiment* can be used to test other components.

Maintainability Incorporating changes to the implementation of the experiment or finding and fixing errors does not pose a problem in terms of technical complexity or required time. As a result, development is accelerated and debugging is easier.

In both examples of experiments, values of the parameters of the experiments (number of nodes, MPI runtime) should be easily configurable.

Support for common patterns Standard patterns specific to experimentation are available. In particular, since the large-scale experiments consist of tasks executed on thousands of nodes, parallel and distributed execution of them is well supported. Various synchronization primitives are available, some of them relaxed (e.g., using a partly successful deployment).

A typical case is running a command on all nodes in parallel.

Snapshotting A state of partially executed experiment can be saved to persistent storage and restarted later. This feature can save a lot of user's time during the design phase, when errors are common and traditionally force the user to run the experiment from the scratch.

If *gLite experiment* failed during data analysis, the platform redeployment can be skipped.

Error handling Unavoidable errors (timeouts, failures, etc.) in large-scale experiments are handled automatically by the framework. Failed experimental tasks can be restarted or migrated to a different location, making the experiment resistant to exceptional situations.

If, for instance, a node fails during *MPI experiment*, it will be replaced by another one.

Integration with lower-level tools The low-level and system specific tools (file distribution service, node reservation service, etc.) are available as a well-defined abstraction. That way a well-tested and efficient applications are available for the user as a black box.

The installation of customized operating system (e.g., Scientific Linux in *gLite experiment*) is usually offloaded to a dedicated service, for example.

Human interaction A cooperation with a human being (e.g., on-the-fly decisions, manual assessment of results) is supported during the experiment. This is a necessary requirement as some tasks cannot be automated.

In *MPI experiment*, the user can be asked for the number of nodes involved in the experiment.

Monitoring The progress and performance of the experiment execution and its constituents' state is continuously monitored. In case of error during the execution, its reason and the offending component are readily available for analysis. Information on execution performance allows to spot optimization opportunities.

Timing information on the deployment time of various parts of gLite stack could be used to pinpoint bottlenecks in *gLite experiment*, for instance.

Instrumentation It is possible to create custom measurements taken during the experiment. With that feature, nontrivial observations can be made, errors found and bottlenecks identified.

Energy consumption in *MPI experiment*, offered by external service, is a good example.

Data analysis The framework provides a service to collect, analyze and visualize experimental results. Therefore, conclusion drawing is a part of the experiment execution and the experimental process is fully automated (hence improving reproducibility).

The last part of *gLite experiment* and *MPI experiment* can integrate statistical analysis of data.

3. Related work

In Section 3.1 we discuss platforms used to run large-scale experiments. Then, in Section 3.3 and 3.4, tools for management of experiment execution, followed by tools for management of system configuration are described. Finally, Section 3.5 describes scientific workflows.
The multitude of approaches shows how difficult the problem is and how unobvious its solution must be. Even though the existing work addresses some of our requirements, our approach has new qualities (*Descriptiveness*, *Modularity*, *Reusability*, *Maintainability*, *Support for common patterns* and *Human interaction*) not featured by the prior work.

3.1. Experimental testbeds

One of the solutions to ease distributed systems research is to build dedicated testbeds (Grid'5000, Emulab, PlanetLab, FutureGrid, DAS-4). Such testbeds offer a large-scale computer infrastructure with dedicated tools to control the resources on behalf of the researchers. Consequently, the research is accelerated and more reliable, but is also tied to a particular platform, complicating reproducibility. Some testbeds allow to run experiments using custom operating systems.
Generally, testbeds do not offer an integrated solution for management of experiments. Various efforts exist to address this problem (cf. Section 3.3).

3.2. Cloud computing

In the era cloud computing another approach was proposed - running the experiments in the cloud infrastructure [3]. There are many advantages: the infrastructure can be transparently scaled and it is cheaper than self-managed computing center. Thanks to the virtualization technology, the state of the environment can be stored intact, solving the problem of software version compatibility. Finally, cloud providers may offer integrated orchestration solution that aids the process of setting up the experimental environment.

However, reproducibility in the cloud is more difficult to achieve as the resources are shared between multiple users. If precise control over the environment is needed, the cloud infrastructure may not provide such low-level access.

3.3. Tools for experiment management and control

Expo The experiment engine Expo [8] manages experiment execution on dedicated platforms. Using a domain-specific language, the experimenter can reserve nodes, check their availability, execute commands and collect results.

Unfortunately, Expo does not provide advanced workflow patterns, does not handle errors in a transparent way and does not differ substantially from writing raw scripts. Therefore, it cannot be used reliably for the management of very complicated experiments.

g5k-campaign With the g5k-campaign [4] tool, the experiment is implemented as a class in Ruby. Various steps of the experiment (i.e., resource reservation, deployment, installation, execution of the experiment and cleanup) can be customized.

The g5k-campaign tool can be used to automatize a large part of the experiment, mostly during the deployment and basic installation phases, but it has no support for advanced installations. Also, it is a Grid'5000-specific tool and cannot be used elsewhere.

ZENTURIO The next tool, ZENTURIO [6], is a system to conduct parameter studies, performance analysis and software testing on cluster and grid architectures. It employs a special language to specify a set of application parameters under study, and interfaces with the grid services to run and control the experiments. A graphical user portal enables the user to monitor the experiments, and the results can be automatically visualized afterwards.

Even though ZENTURIO helps with conducting a large number of experiments, its support for *large-scale experiments* is questionable. Moreover it treats the experiment as a black box without any special help for its design or support for advanced experimental scenarios.

DART Distributed Automated Regression Testing for Large-Scale Network Applications [2] is a tool to automatically track regressions in distributed applications. The idea is similar to unit testing in software engineering. DART helps with the design of tests and their execution. Moreover, exceptional situations (e.g., network faults) can be simulated.

Technically, DART tries to achieve a different goal than described in this paper. Nevertheless, it can be used to manage and automate executions of the distributed application. It does not, however, help with a design of the experiments and ensuring reproducibility.

Plush (Gush) To aid configuration, management and visualization of distributed applications, Plush [1] or its successor Gush can be used. Plush is scalable, failure tolerant and features monitoring and reconfiguration of the system. Its usage scenarios include short-lived, long-lived and grid applications. Plush was originally developed for PlanetLab testbed.

Plush neither focuses on the reproducibility nor supports complicated or non-standard patterns that may arise when large-scale experiments are conducted.

Weevil The research project Weevil [9] concentrates on experimenting with highly distributed systems, that is, systems with multiple interfaces accessed by many users. The solution helps with deployment and execution of the experiment and workload generation.

Weevil provides some level of automation, but the experiment is executed as a script produced by the framework. As such, the experiment execution is static and cannot accommodate for the exceptional situations. Instead, Weevil concentrates on the integrated, sophisticated workload generator that allows for design of advanced experimental scenarios.

NXE Network eXperiment Engine [5] automates the execution of networking experiments by providing an easy way to interact with experimental nodes and collect the results.

The workflow implemented in NXE is rather static and, in principle, it does not differ from a common way of writing a set of scripts in an imperative programming language. Moreover, the primary focus is on networking experiments.

4. http://g5k-campaign.gforge.inria.fr/
5. http://ens-lyon.fr/LIP/RESO/Software/NXE/

Experimentation workbench for Emulab The Emulab testbed provides a GUI interface for the users to control experiments, but it offers only a basic functionality. To address this, the experimentation workbench for Emulab [4] supports reproducible research by helping with the design and management of the experiment execution. Integrated collection of results and collaborative work on the experiment are also possible.

The solution suffers from some scalability issues (slow collection of results) and does not handle exceptional situations well. The authors also mention usability problems stemming from the implemented model of experimentation.

3.4. Tools managing system configuration

In the advent of complex computer systems, the problems with managing their configuration arise. The solution is to automate this process and many tools exist to help with that problem, among them Puppet and Chef. Using a domain specific language (DSL), a desired configuration of the nodes is described. Upon the execution of the script, the system is brought to the this state automatically. The scripts are *idempotent*: applying them many times does not result in errors.

The configuration management tools are often used to handle the initial setup of the experiment, providing reproducible configuration of the system. Unfortunately, they lack advanced control structures and an additional layer is needed to bootstrap them and apply the scripts.

3.5. Scientific workflows

Another approach to facilitate experimentation and make it reproducible is based on scientific workflows [11]. They are common in many domains of computational science (e.g., biology, physics). Tools like Taverna, Kepler, Pegasus improve the experiment design, facilitate mapping of computing resources to the workflow and transparently handle exceptional situations.

Scientific workflow tools are well suited for managing computation on *a priori* available data or data queried from public databases, but not if the source of data is the computer system itself. Therefore, they are not well suited for computer science research.

4. Research directions

4.1. Overview

In this paper, we present a new approach to attack the problem of orchestrating large-scale experiments using Business Process Management (BPM). Although traditionally used to model and optimize the processes within an organization, it can be a powerful approach to our problem. Computer systems already profit from this approach to provide robust software systems [10].

A business process can be defined as a set of activities performed to realize a business goal. Business process management is a set of concepts, methods and techniques to help with design, administration, configuration, analysis and optimization of business processes. There exist many workflow patterns that can be used to implement business processes [7]. We use the Business Process Management Notation notation, but there are alternatives like Petri Nets, Workflow Nets or Yet Another Workflow Language.

An example shown in Figure 1 is an experiment modeled using business workflows. In this process, gLite is deployed inside a computer system and one component is evaluated (cf. *gLite experiment* in Section 1). The deployment consists of few steps, some of them parallel.

Our approach consists in using the achievements of BPM and adapting them to the field of distributed systems research. In particular, we want to use BPM to design the experiments using the experience gathered by its research community. Moreover, we want to adapt software solutions developed in the BPM domain to our unique needs.

In the next section we discuss the benefits that this approach provides. The most prominent features of BPM approach are identified (*Understanding*, *Modularity*, *Monitoring* and *Workflow patterns*) and confronted with the requirements for the experiment engine identified in Section 2.

4.2. Expected benefits

The main features of BPM are listed below and confronted with the requirements in Section 2.

Understanding Business processes are abstractions created to understand the processes involved in the production. Similarly, workflow-based representation of experiments will provide

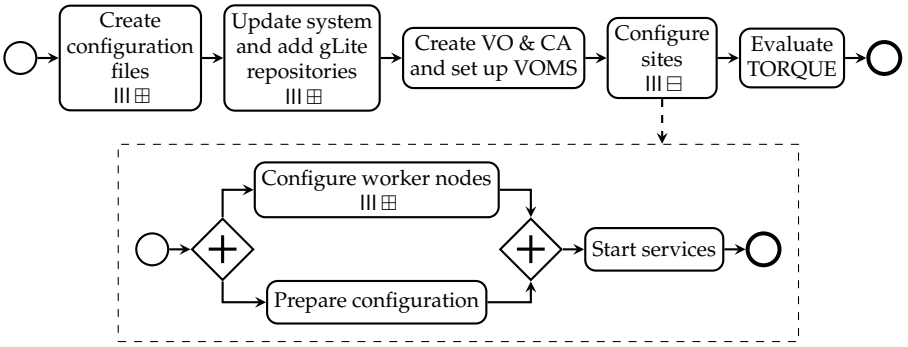

Figure 1: A workflow representation of *gLite experiment*. It consists of a workflow representing the whole process (top workflow) and a subprocess for the *Configure sites* phase (bottom workflow, executed for each site in parallel).
Activities are represented by boxes, whereas arrows enforce their order. Nodes marked with ||| symbol represent subprocesses which execute activities in parallel (⊞ and ⊟ symbols depict a collapsed and an expanded subprocess, respectively). Finally, the diamond-shaped nodes marked with a plus sign execute two paths concurrently and subsequently wait for them to finish. The abbreviations (VO, CA, VOMS) are names of gLite services, TORQUE is the batch scheduler.

better understanding of the experiment and its components by representing them in a convenient, high-level, graphical form. Then, with deeper knowledge about the experiment, the ideas to optimize it will be more evident.

This addresses *Descriptiveness*.

Modularity Business activities are independent building blocks of business processes. They can be replaced, shared between experiments and maintained independently, while unrelated activities can be executed on different physical hosts in a distributed fashion. Moreover, at every transition between activities the experiment state can be saved as a snapshot. Finally, the interface to low-level tools can be implemented as a reusable activity.

Unfortunately, sometimes it may not be possible to save a snapshot: if a state contains temporary resources or resources that have only local meaning (e.g., file descriptors) there is no reliable way to regain these resources later. Moreover, to implement snapshotting, the activities must be idempotent actions, so that they can be safely re-executed later (cf. Section 3.4).

This feature offered by BPM approach addresses *Modularity*, *Reusability*, *Maintainability*, *Snapshotting* and *Integration with lower-level tools*.

Monitoring In the business process lifecycle, monitoring is one of the steps and BPM software supports this feature. In complex configurations and workflows this may be a crucial feature to understand and find a problem. Moreover, by stretching this idea a bit further, instrumentation can be introduced as a pluggable system to collect data on execution of activities and collection of results can follow a similar principle. Another advantage is that the results are tied to the activities, so that the provenance of the results can be traced. Finally, data analysis can be represented as a pluggable activity with implementation of common tasks.

This addresses *Maintainability*, *Monitoring*, *Instrumentation* and *Data analysis*.

Workflow patterns Workflow description of business processes builds on patterns that reflect common situations. Although these patterns differ between business processes and experiments, business patterns are a good starting point and missing ones can be identified. Moreover, patterns that support error handling and task synchronization may be introduced. Finally, BPM is traditionally well-suited for processes involving human beings.

To illustrate this, let us note a common experimental pattern: running multiple instances of one activity with different parameters (e.g., *Configure sites* task in Figure 1). Surprisingly, this pattern is not well supported by the existing workflow representations and BPM software.

Hence, with additional work, the workflow patterns identified by BPM research address the requirements of *Support for common patterns*, *Error handling* and *Human interaction*.

4.3. Work plan

After a review of workflow systems, we determined that none provided the features required for our application of BPM to experimental methodology. We have tried scientific workflow systems (e.g., Taverna) and found them too data-centric and not applicable to our problem. The open-source workflow engines (e.g., ruote [6]) proved to be difficult to adapt.

As the result, our plan is to write a proof-of-concept workflow engine for experiment orchestration. Currently, we are exploring different architectures for the representation of the workflows and ways to address peculiarities of our approach. Next, we will port our experiments to this workflow engine and evaluate if it has all required properties (as described in Section 2).

4.4. Other applications

Our approach can be used to reliably automate complicated and repetitive processes involving computer systems. This may range from deterministic data-driven workflows to complicated, large-scale deployments. Such tools recently emerge (e.g., Amazon Simple Workflow Service [7]).

Another application concerns experiments with human interaction. As people can be even less reliable than computer systems, an integrated solution to handle this interaction is necessary. Moreover, crowdsourcing, i.e., delegating work to numerous, heterogeneous groups of people, is in principle distributed system built on human beings. Therefore, the management of crowdsourcing initiatives can profit from our work as well.

5. Conclusions

In this paper, we presented our preliminary work on experimental methodology in distributed systems research. To this end, we described a promising, novel approach to orchestration of experiments in large-scale systems based on Business Process Management, analyzed its advantages, disadvantages and questions that arise when such interdisciplinary approach is used. Finally, we presented our future plans and industrial applications of our work.

Bibliography

1. Jeannie Albrecht, Christopher Tuttle, Ryan Braud, Darren Dao, Nikolay Topilski, Alex C. Snoeren, and Amin Vahdat. Distributed Application Configuration, Management, and Visualization with Plush. *ACM Transactions on Internet Technology*, 11:6:1–6:41, December 2011.
2. Brent N. Chun. DART: Distributed Automated Regression Testing for Large-Scale Network Applications. In *Proc. of the 8th International Conference on Principles of Distributed Systems*, 2004.
3. Joel T. Dudley and Atul J. Butte. In silico research in the era of cloud computing. *Nature Biotechnology*, 28(11):1181–1185, 2010.
4. Eric Eide, Leigh Stoller, and Jay Lepreau. An Experimentation Workbench for Replayable Networking Research. In *Proc. of NSDI*, 2007.
5. Darrel C. Ince, Leslie Hatton, and John Graham-Cumming. The case for open computer programs. *Nature*, 482(7386):485–488, February 2012.
6. Radu Prodan and Thomas Fahringer. ZENTURIO: An Experiment Management System for Cluster and Grid Computing. In *Proc. of CLUSTER*, 2002.
7. W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distrib. Parallel Databases*, 14(1):5–51, July 2003.
8. B. Videau and O. Richard. Expo : un moteur de conduite d'expériences pour plates-forme dédiées. In *Conférence Française en Systèmes d'Exploitation (CFSE)*, 2008.
9. Yanyan Wang, Matthew J. Rutherford, Antonio Carzaniga, and Alexander L. Wolf. Automating Experimentation on Distributed Testbeds. In *Proc. of ASE*, 2005.
10. Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer Publishing Company, Incorporated, 1st edition, 2010.
11. Jia Yu and Rajkumar Buyya. A Taxonomy of Scientific Workflow Systems for Grid Computing. *SIGMOD Record*, 34:44–49, September 2005.

6. http://ruote.rubyforge.org/
7. http://aws.amazon.com/swf/