Model Predictive Analysis for Autonomic Workflow Management in Large-scale Scientific Computing Environments

Steve Nordstrom Abhishek Dubey Turker Keskinpala Rahul Datta
Sandeep Neema Ted Bapty
Institute for Software Integrated Systems
Vanderbilt University
Box 1829, Station B
Nashville, TN 37235
{steve-o, dabhishe, tkeskinpala, rdatta, sandeep, bapty}@isis.vanderbilt.edu

Abstract

In large scale scientific computing, proper planning and management of computational resources lead to higher system utilizations and increased scientific productivity. Scientists are increasingly leveraging the use of business process management techniques and workflow management tools to balance the needs of the scientific analyses with the availability of computational resources. However, the advancements in productivity from execution of workflows in a large scale computing environments are often thwarted by runtime resource failures. This paper presents our initial work toward autonomic model based fault analysis in workflow based environments.

1. Introduction

Large-scale computation systems used by the scientific community to analyze data are typically composed of thousands of computational nodes connected by state-of-the-art, high bandwidth interconnects. The correct planning and execution of analyses are crucial to the production of new scientific results. Such planning is typically done through collaborations between scientists and engineers to ensure that that the scientific needs can be met by the analysis systems. Execution is performed via automation of the resultant workflows.

1.1. Workflows

A workflow is a specification of a set of tasks or jobs to be performed, their execution order and their input/output dependencies. Workflows are typically used in business and information technology scenarios, where repetitive jobs are composed of complex sets and orderings of tasks. Examples of such scenarios might include business production streams, information flow automation and telecommunication systems. Workflow specifications are used to drive the scheduling and deployment of tasks across a variety of resources, both computational and human.

1.2. Workflow needs of the scientific computing community

To reduce the complexity inherent in analysis, planning and deployment, the scientific community is increasingly using workflow management techniques [17] [14] to handle complex data processing flows. While many workflow management systems are geared toward business environments, there are several new developments in bringing these technologies to the scientific community. Engineers and computer scientists who build systems to perform analysis of high energy physics data are investigating how to supplement workflow based tools and techniques to alleviate the complexities of job control and scheduling, and data dependency. They are also very concerned with quality of service (QoS) and fault tolerance. Scientific experiment analyses are run on high performance computing clusters with state-of-the-art interconnectivity. Typically, the allocation of cluster resources are handled by a job scheduling framework such as Portable Batch System (PBS) or Maui [3]. These systems are primarily concerned with achieving high resource utilization, typically assuming reliable hardware.

Temporal and persistent failure of certain cluster components both at the local and shared level are a fact of life, as cluster sizes increase and as hardware pushes performance to the limit. Having failure-prone components as integral parts of a workflow can lead to workflows which either can not be executed, or spend significant time in a stalled state.

Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASe'07)
0-7695-2809-0/07 $20.00 © 2007 IEEE

Typically, these failures and workflow stalls must be manually detected and resolved. Again, as cluster sizes increase, this manual intervention becomes infeasible.

Thus there is a desire for workflow based systems to automatically detect faults and mitigate errors as they occur. Such accommodations may include alteration of a workflow, restarting workflow segments, or suspension of a current workflow in favor of a workflow which can be completed with more limited resources. If properly implemented, this autonomic behavior can greatly enhance the productivity of scientific computing clusters and make scale-up of these clusters feasible. This paper discusses model based tools and techniques which are currently being developed to construct workflow environments which suit the needs of the large-scale scientific computing community.

2. Background and related work

2.1. Model based design

Model-based design has gained momentum in recent years as a sound methodology of applying computer-based modeling methods to a variety of problem domains. Model-integrated computing (MIC) [19], in particular, incorporates the creation of domain-specific modeling languages which capture relevant aspects of a complex computer-based system. These models can then be queried or transformed to produce a variety of domain specific artifacts. Examples of MIC applications are the generation of real-time schedules from software dataflow models, the creation of configuration files from system models to configure a distributed system, or the generation of source code from behavioral models which can be integrated into an existing software framework.

The creation of modeling abstractions which are specific to a given problem domain is the cornerstone of MIC. This process relies heavily on the use of domain-specific modeling languages (DSMLs) to capture relevant characteristics of an object or system of objects. The Generic Modeling Environment (GME) [13] is a configurable, domain-independent, and freely available [8] tool which provides a platform upon which to perform MIC design and development.

2.2. Workflow environments

There are many different research efforts on workflows in grid environments. In [5], Deelman et al. lay out the issues with workflow management in the Grid, in general. They mention the important aspects of workflow generation and planning. In [7], Deelman et al. focus on the automatic generation of workflows for the Grid. The generated workflows show how individual application components are brought together for the execution of a complex application. They identify mapping application requirements to an abstract workflow and mapping the abstract workflow onto the Grid as two main steps. Deelman et al. also describe the Pegasus tool that can map complex workflows onto the Grid in [6]. In [20], Linden et al. present GridAssist which is a tool Grid-based workflow management tool. GridAssist helps the user to execute workflows in a Grid environment without requiring the user to know the internal details. In [2], Amin et al. present another workflow management system called GridAnt which lets users manage complex workflows and specify complex workflow dependencies in XML. In [4], Cao et al. propose a Grid workflow management framework called GridFlow for grid computing. The system provides support for both global grid workflow management, and local grid sub-workflow scheduling.

In [11], Krishnan et al. investigate leveraging the existing Web services workflow technologies for Grid services. For this purpose, they survey different workflow specification languages for Web services and identify specific needs of Grid services. They present their own language called Grid Services Flow Language that addresses the needs of the Grid services. Another workflow specification language is described in [9] by Fahringer et al. The Abstract Grid Workflow Language (AGFL) presented in their paper, provides a way to specify Grid workflows in a high level of abstraction by hiding the implementation details of the Grid technology from the user.

In effort to enable autonomic applications on the Grid, in [1], Agarwal et al. present an autonomic component framework for Grid applications called AutoMate. In this work, the issue of autonomic application development by dynamically composing autonomic components is investigated. In [16], Nichols et al. recognize the importance of autonomic characteristic of self-healing to ensure execution of critical workflows on the grid platform. For this purpose, they propose embedding a model for dynamic fault tolerance in a mobile agent workflow management system. It is explained that this way the system can optimally configure its fault tolerance mechanisms. In a similar effort, in [15], Liu et al. present the Accord autonomic service architecture. Using Accord, autonomic services can be developed and autonomic applications can be formed by composing these autonomic services. Moreover, Accord enables the applications to adapt to the uncertainties of the Grid environment. In [10], Heinis et al. present an autonomic workflow engine which tackles the problem of deploying systems in optimal configuration. The autonomic properties of their workflow engine enable alteration of the workflow configurations with respect to the workload variations without any user intervention. In addition, the engine enables recovery of the workflow execution state.

3. Autonomic Workflow Management

Recent efforts toward autonomic workflow environments have been shown to exhibit various qualities of autonomic systems which require less user intervention in the successful execution of a workflow. However, there is still a strong desire in the scientific computing community to have more control over the corrective actions taken by the workflow management system when components begin to fail.

The self-* tenets of autonomic computing [18] serve to characterize the introspective qualities of systems which can be considered autonomic. In considering an autonomic workflow environment, an important quality which is to be desired is that of self-management. Clearly, the management of jobs and resources is pivotal to the successful completion of a workflow. For the scientific community, the number of jobs and the variety of resources make the task of managing the workflow difficult for humans. It is therefore desirable that the tools used to specify and execute a workflow display the autonomic property of self-management.

In the following sections we will present an initial design of a workflow modeling language and analysis techniques which integrate the specification of a scientific workflow with simulation and fault analysis.

3.1. WorkflowML: a workflow modeling language

The *WorkflowML* modeling language was created using GME and the MetaGME modeling framework. The language supports three main features which address the needs of the scientific computing community:

- Modeling of jobs, data stores, and desirable outputs
- Synchronization of jobs without regard to data dependencies
- Supports model based analysis techniques

It is our aim to produce a language which is capable of satisfying these criteria. In addition, we intend to extend our existing languages for scientific computing (GHOSTML and SIML) with workflow modeling capabilities. In the following section we will introduce our initial efforts toward this goal.

WorkFlowML is based around principles of synchronous dataflow (SDF) languages with a few additions, and models expressed in this language can be mapped to a directed acyclic graph (DAG). The choice to use acyclic graphs is based on simplicity; as this work progresses the language may be extended to include support for cyclic graphs. *External Data Stores* and *Internal Data Stores* are added the SDF base to allow the modeler to capture data dependencies between jobs and services which supply data for the job. *Synchronizers* are used to place an arbitrary synchronization constraint on the scheduling of jobs that are not necessarily data dependent.

A *Job* is represented as an octagon and is annotated with an integer attribute to capture the typical execution time of the job and a boolean value which indicates whether or not the job is *desired*. The desirability of a given job is an indication that the result obtained by executing a job is somehow interesting to the designer of the workflow. This information is useful in determining the completion status of a suspended workflow, or for determining whether a faulty workflow should continue to be executed.

Jobs are considered to be in one of four states, *Initial*, *Running*, *Finished*, *Faulty*. Jobs are initially in the *Initial* state. Jobs whose input, output, and scheduling dependencies are satisfied are then moved to the running state, where they execute until finished. Failed jobs are those which cannot continue due to software or resource problems. Jobs are annotated as being desired or undesired, which indicates whether the designer values the output of the job or not. The usefulness of this metric is detailed later in this chapter.

Data Stores represent external or internal data stores upon which data can be consumed or produced by Jobs. The data stores are annotated with a unique resource identifier (URI).

There exist three distinct types of dependencies which can be expressed in a WorkflowML model:

- Data: Jobs which have an inbound connection from either a data store or another job are said to be data dependent upon that store or job.
- Sequential: A job A is said to precede another job B if there exists a data path of any size through the edges of the graph from A to B. A job can not be scheduled until all jobs which precede it have completed.
- Synchronous: All jobs which have an inbound association with a synchronizer may be scheduled if and only if all jobs which have an outbound relation to that synchronizer are finished.

Formally, the domain constructs can be defined as sets and the above mentioned dependencies as relations on those sets. These definitions are stated in the following sections.

3.2. Sets

- J: Set of jobs that are part of the workflow.
- S: Set of synchronizers that are part of the workflow.
- D: Set of data stores, both external and internal that are part of the workflow.

3.3. Relations

- $R_{DJ} \subseteq D \times J \cup J \times D$: Data dependency between a job and a data store s.t. $(\forall j \in J)(\forall d \in D)((d, j) \in R_{DJ} \Rightarrow j$ depends on d for input). If $(j, d) \in R_{DJ}$, then j depends on d for output storage.
- $R_{JJ} \subseteq J \times J$: Sequential dependency relation between two jobs s.t. $(\forall j_1, j_2 \in J)((j_1, j_2) \in R_{JJ} \Rightarrow j_2$ depends on j_1). It should be noted that Sequential dependency is transitive i.e. $(j_1, j_2) \in R_{JJ} \wedge (j_2, j_3) \in R_{JJ} \Rightarrow (j_1, j_3) \in R_{JJ}$. Moreover, indirect dependency between any two jobs via data dependency or synchronous dependency implies sequential dependency. For example let $j_1, j_2 \in J$ be two jobs and $s \in S$ be a synchronizer, $(j_1, s) \in R_{SJ} \wedge (s, j_2) \in R_{SJ} \Rightarrow (j_1, j_2) \in R_{JJ}$
- $R_{SJ} \subseteq S \times J \cup J \times S$: Dependency between a job and a synchronizer s.t. $(\forall j \in J)(\forall s \in S)((j, s) \in R_{SJ} \Rightarrow s$ need j to finish before it can launch any other job $k \in J$ s.t. $(s, k) \in R_{SJ})$.

Typical workflow execution engines may provide some simple levels of fault mitigation, such as process migration or rescheduling. However, there exist certain types of resource failures such as network or database outages, thermal failures in racks of machines, or persistent software crashes which may hinder the eventual completion of the workflow. In these cases, it is necessary to determine the extent to which a workflow can be executed and provide reports to the operator so that she may make a decision to continue or stall the workflow.

To determine the extent to which a workflow can be executed, we construct a simple workflow execution engine simulator within the modeling environment which can be run in parallel with the actual system. The model is assumed to accurately represent the current state of the workflow. This can be achieved through the use of a synchronizer or observer which is monitoring the progress of the workflow execution. When a fault occurs, all available autonomic reflexes are exhausted in an attempt to mitigate this fault at the local level before the workflow prediction is performed. If the fault is unable to be mitigated by a reflex, the model is updated and a simple simulation algorithm begins in order to determine the future states of the workflow given no external intervention and no future faults.

Since jobs within the model are annotated as being desired or undesired, we can use this as a metric to determine the relative desirability of a partially completed workflow. If the workflow progresses to a state where no desirable jobs complete, it may become an easy decision to suspend the workflow.

To demonstrate the use of these features an example model is shown which is derived from an example workflow execution scenario proposed by Jim Kowalkowski and Luciano Piccoli and the Lattice Gauge Theory Computational Facility [12] at Fermi National Accelerator Laboratory.

4. Analysis

The model given is that of a simplified workflow with synchronization, sequence, and data dependency. The input to the predictive algorithm requires the DAG for the workflow. In order to make the definition of the algorithm complete, let us summarize the various sets and relations that are part of the DAG.

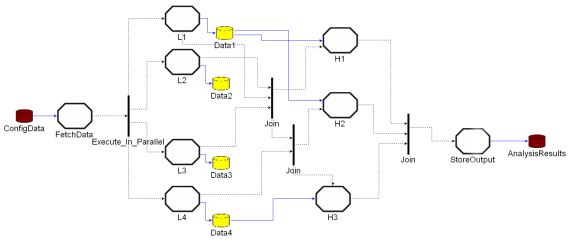

Figure 1. Initial workflow model expressed in WorkflowML.

The initial model shown in Figure 1 depicts a workflow utilizing a variety of jobs, synchronizers, and both internal and external data stores. The job labeled *StoreOutput* is the only desired job in this example. When a *Faulty* state is observed in job *L4*, the lookahead simulation begins. The initial state of the workflow simulation is shown in Figure 2. Note that jobs *L1*, and *L2*, are currently in the *Running* state, while jobs *FetchData* and *L3* are in the *Finished* state. All remaining jobs are in the *Initial* state.

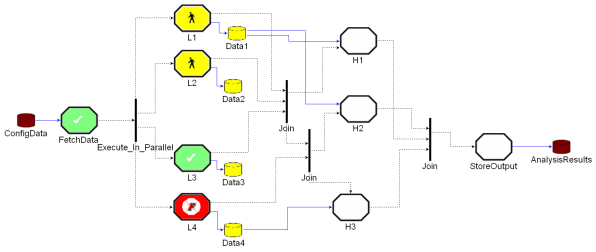

Figure 2. A faulty WorkflowML model upon which the analysis is performed, showing two jobs in the *Finished* state, two in the *Running* state, and one in the *Faulty* state.

Given that a fault has occurred, a predictive simulation is conducted in parallel with the running system by applying the lookahead algorithm given in Algorithm 1 to a copy

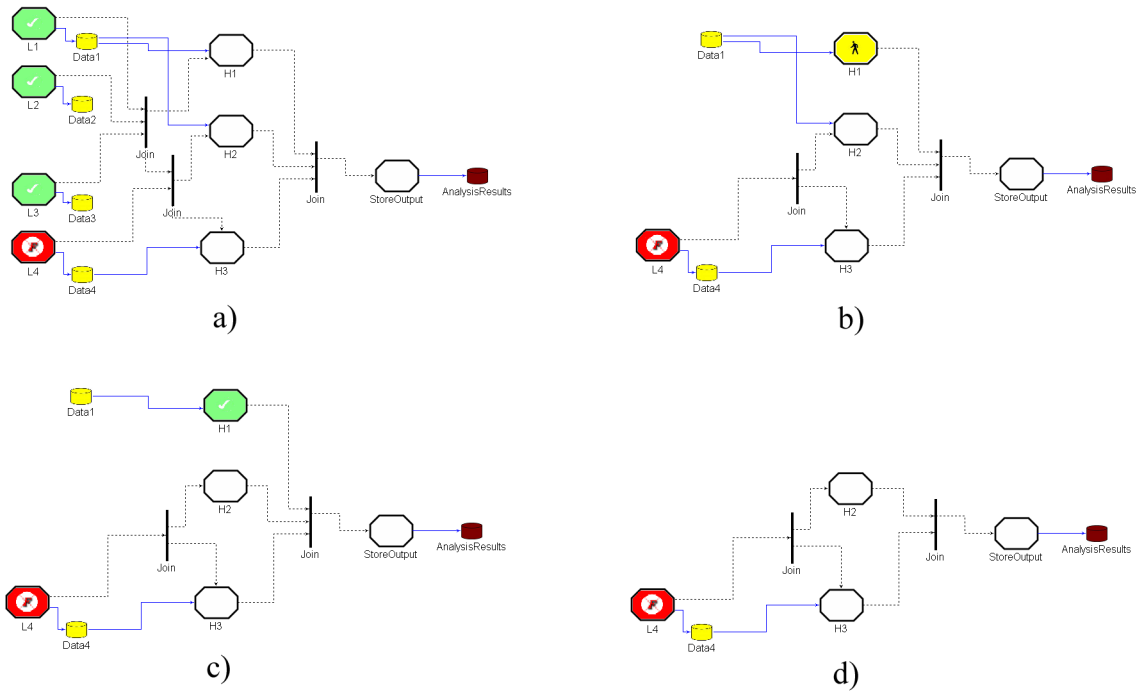

Figure 3. A visual representation of the lookahead algorithm showing a) the primary prediction, b) the secondary prediction, c) the third prediction, and d) the final stalled workflow.

Algorithm 1 Look Ahead Analysis

Input: DAG: $Nodes \subseteq J \cup S \cup D$ and $Edges \subseteq R_{DJ} \cup R_{SJ} \cup R_{JJ}$

Pre Condition: $(\exists j \in Nodes \cap J)(Faulty(j))$

Output: DAG: $Nodes$, $Edges$ {A maximal analysis of all the jobs that can not be completed.}

while $(\exists j \in Nodes \cap (J \cup S))(\neg Faulty(j))(\neg \exists j' \in Nodes \cap (J \cup S))((j', j) \in Edges)$ **do**

Mark j as *Running* {This can be used to simulate the running job}

Mark j as *Finished*

$Nodes \leftarrow Nodes - j$

$Edges \leftarrow Edges - (j, *)$ {* can match any node}

end while

---

of the initial model depicted in Figure 2. The intermediate transformed models are depicted in the subfigures a), b), c), and d) of Figure 3.

Figure 3 a) shows that the jobs *L1* and *L2* have both been moved the *Finished* state. Figures 3 b) and 3 c) show job *H2* progressing from the *Running* state to the *Finished* state. Note that since there is no representation of the passage of time in this simulation, the progression of state happens in immediate succession for all jobs whose scheduling conditions are satisfied. Finally, Figure 3 d) depicts the final remaining state of the workflow, where job *L4* remains in the *Faulty* state while jobs *H2*, *H3*, and *StoreOutput* have all remained in their *Initial* state.

The workflow in Figure 3 d) is considered stalled in that the job *H2* cannot be scheduled due to a synchronization dependency on the completion of the faulty job *L4*. Furthermore, the job *H3* cannot be scheduled due to both a synchronization and data dependency on the completion of job *L4*. Finally, the *StoreOutput* job (which is a desired job) remains unscheduled due to its synchronization dependency upon the completion of *H2* and *H3*.

One can see that a determination regarding whether or not to continue this workflow is in order. This decision gives us a chance to preempt the workflow predicted to stall and make way for the execution of some other existing workflow that is not predicted to stall given the current set of failures. This leads to a trade-off between workflows that can be completed versus those workflows which are destined to fail.

5. Conclusions and Future Work

This paper presents an ongoing development of concepts for the integration of autonomic features into workflow specification and execution. When resources are failing in large scale computing environments, tools such as these are necessary to provide scientists and system operators with decision aids, such as predictions of if their desired computations can be completed. In this regard, predictive analysis presents a key feature of autonomic workflow management tools which aim to provide self-evaluation and self-management.

Future work in this area include the integration of on line predictive analysis results to automatically stall and checkpoint workflows that may not be able to complete given the failures in the system and the currently available resource set. Also, additional checks are being investigated with the workflow submission process that can give scientist immediate feedback of the predicted completion of the workflow before it is scheduled. Additionally, preplanned mitigation actions can be integrated into the workflow, allowing an application-specific handling of faults. Finally, this set of tools can be abstracted to drive common cluster scheduling systems which do not provide the level of autonomicity desired by the community.

Acknowledgments

This work is supported by US Department of Energy grant number DE-FC02-06ER41447. The authors would like to thank Jim Kowalkowski for his concrete knowledge of software systems and his rigorous adhesion to software standards which continuously contributes to the quality of the research presented in this paper.

References

[1] M. Agarwal, V. Bhat, H. Liu, V. Matossian, V. Putty, C. Schmidt, G. Zhang, L. Zhen, M. Parashar, B. Khargharia, and S. Hariri. Automate: enabling autonomic applications on the grid. *Autonomic Computing Workshop, 2003*, pages 48–57, 2003.

[2] K. Amin, G. von Laszewski, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi. Gridant: a client-controllable grid workflow system. *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pages 10 pp.+, 2004.

[3] B. Bode, D. Halstead, R. Kendall, and Z. Lei. The portable batch scheduler and the maui scheduler on linux clusters. *Proceedings of the 4th Annual Linux Showcase and Conference*, October 2000.

[4] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd. Gridflow: workflow management for grid computing. *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 198–205, 2003.

[5] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Workflow management in griphyn. *Grid resource management: state of the art and future trends*, pages 99–116, 2004.

[6] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. *Lecture Notes in Computer Science : Grid Computing*, pages 11–20, 2004.

[7] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, V1(1):25–39, March 2003.

[8] The ESCHER research institute. www.escherinstitute.org.

[9] T. Fahringer, J. Qin, and S. Hainzer. Specification of grid workflow applications with agwl: an abstract grid workflow language. *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, 2:676–685 Vol. 2, 2005.

[10] T. Heinis, C. Pautasso, and G. Alonso. Design and evaluation of an autonomic workflow engine. *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 27–38, 2005.

[11] S. Krishnan, P. Wagstrom, and G. V. Laszewski. GSFL: A workflow framework for grid services, Aug. 14 2002.

[12] The lattice gauge theory computational facility (LQCD). http://lqcd.fnal.gov.

[13] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi. Generic modeling environment. In *WISP*, volume IEEE International Workshop on Intelligent Signal Processing, Budapest, Hungary, May 2001.

[14] Q. Li, Z. Shan, P. C. K. Hung, D. K. W. Chiu, and S. C. Cheung. Flows and views for scalable scientific process integration. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 30, New York, NY, USA, 2006. ACM Press.

[15] H. Liu, V. Bhat, M. Parashar, and S. Klasky. An autonomic service architecture for self-managing grid applications. *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on*, pages 8 pp.+, 2005.

[16] J. Nichols, H. Demirkan, and M. Goul. Autonomic workflow execution in the grid. *Systems, Man and Cybernetics, Part C, IEEE Transactions on*, 36(3):353–364, 2006.

[17] S. Shankar, A. Kini, D. DeWitt, and J. Naughton. Integrating databases and workflow systems. *SIGMOD Rec.*, 34(3):5–11, 2005.

[18] R. Sterritt. Autonomic computing. *Innovations in Systems and Software Engineering, A NASA Journal*, 1(1), April 2005.

[19] J. Sztipanovits and G. Karsai. Model-integrated computing. *Computer*, 30(4):110–111, 1997.

[20] M. ter Linden, H. de Wolf, and R. Grim. Gridassist, a user friendly grid-based workflow management tool. *Workshop on Web and Grid Services for Scientific Data Analysis (34th ICPP'2005)*, pages 5–10, June 2005.