A Workflow Platform for Simulation on Grids

Toan Nguyen, Laurentiu Trifan, Jean-Antoine Desideri

A Workflow Platform for Simulation on Grids

Toàn Nguyên, Laurentiu Trifan
Project OPALE
INRIA Grenoble Rhône-Alpes
Grenoble, France
tnguyen@inrialpes.fr, trifan@inrialpes.fr

Jean-Antoine-Désidéri
Project OPALE
INRIA Sophia-Antipolis Méditerranée
Sophia-Antipolis, France
Jean-Antoine.Desideri@sophia.inria.fr

***Abstract*—This paper presents the design, implementation and deployment of a simulation platform based on distributed workflows. It supports the smooth integration of existing software, e.g., Matlab, Scilab, Python, OpenFOAM, ParaView and user-defined programs. Additional features include the support for application-level fault-tolerance and exception-handling, i.e., resilience, and the orchestrated execution of distributed codes on remote high-performance clusters.**

Keywords-workflows; fault-tolerance; resilience; simulation; distributed systems; high-performance computing

I. INTRODUCTION

Large-scale simulation applications are becoming standard in research laboratories and in the industry [1][2]. Because they involve a large variety of existing software and terabytes of data, moving around calculations and data files is not a simple avenue. Further, software and data often reside in proprietary locations and cannot be moved. Distributed computing infrastructures are therefore necessary [6, 8].

This paper details the design, implementation and use of a distributed simulation platform. It is based on a workflow system and a wide-area distributed network. This infrastructure includes heterogeneous hardware and software components. Further, the application codes must interact in a timely, secure and effective manner. Additionally, because the coupling of remote hardware and software components are prone to run-time errors, sophisticated mechanisms are necessary to handle unexpected failures at the infrastructure and system levels. This is also true for the coupled software that contribute to large simulation applications. Consequently, specific management software is required to handle unexpected application and software behavior.

This paper addresses these issues. Section II gives a detailed overview of the implementation using the YAWL workflow management system [4]. Section III is a conclusion.

II. WORKFLOW PLATFORM

A. *The YAWL workflow management system*

Workflows systems are the support of many e-Science applications [1][6][8]. Among the most popular systems are Taverna, Kepler, Pegasus, Bonita and many others [11][15]. They complement scientific software suites like Dakota, Scilab and Matlab in their ability to provide complex application factories that can be shared, reused and evolved. Further, they support the incremental composition of hierarchic composite applications. Providing a control flow approach, they also complement the usual dataflow approach used in programming toolboxes. Another bonus is that they provide seamless user interfaces, masking technicalities of distributed, programming and administrative layers, thus allowing the users and experts to concentrate on their areas of interest.

The OPALE project at INRIA (http://www-opale.inrialpes.fr) is investigating the use of the workflow management system for distributed multidiscipline optimization [3]. The goal is to develop a resilient workflow system for large-scale optimization applications [26]. It is based on extensions to the YAWL system to add resilience and remote computing facilities for deployment on high-performance distributed infrastructures [4]. This includes large-PC clusters connected to broadband networks. It also includes interfaces with the Scilab scientific computing toolbox [16] and the ProActive middleware [17].

Provided as an open-source software, YAWL is implemented in Java. It is based on an Apache server using Tomcat and Apache's Derby relational database system for persistence. YAWL is developed by the University of Eindhoven (NL) and the University of Brisbane (Australia). It runs on Linux, Windows and MacOS platforms [25]. It allows complex workflows to be defined and supports high-level constructs (e.g., XOR- and OR-splits and joins, loops, conditional control flow based on application variables values, composite tasks, parallel execution of multiple instances of tasks, etc) through high-level user interfaces.

Formally, it is based on a sound and proven operational semantics extending the *workflow patterns* of the Workflow Management Coalition [21, 32], implemented and proved by colored Petri nets. In contrast, other workflow management systems which are based on the Business Process Management Notation (BPMN) [27] and the Business Process Execution Language (BPEL) [28] are usually not supported by a proven formal semantics. Further, they usually implement only specific and /or proprietary versions of the BPMN and the BPEL specifications. There are indeed over 73 (supposedly compliant) implementations of the BPMN, as of January 2011, with several others currently being implemented [27], in addition to more than 20 BPEL engine providers. However, BPEL supports the execution of long running processes required by simulation applications, with compensation and undo actions for exception handling and fault-tolerance, as well as concurrent flows and advance synchronization [28].

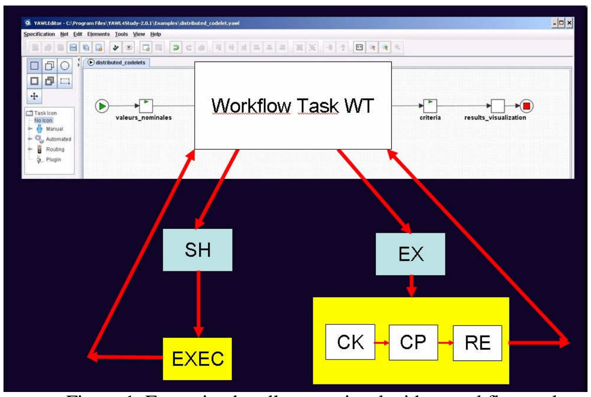

Figure 1. Exception handler associated with a workflow task

Designed as an open platform, YAWL supports natively interactions with external and existing software and application codes written in any programming languages, through shell scripts invocations, as well as distributed computing through Web Services.

It includes a native Web Services interface, custom services invocations through *codelets*, as well as rules, powerful exception handling facilities, and monitoring of workflow executions [13].

Further, it supports dynamic evolution of the applications by extensions to the existing workflows through *worklets*, i.e., on-line inclusion of new workflow components during execution [14].

It supports automatic and step-by-step execution of the workflows, as well as persistence of (possibly partial) executions of the workflows for later resuming, using its internal database system. It also features extensive event logging for later analysis, simulation, configuration and tuning of the application workflows.

Additionally, YAWL supports extensive organizations modeling, allowing complex collaborative projects and teams to be defined with sophisticated privilege management: access rights and granting capabilities to the various projects members (organized as networked teams of roles and capabilities owners) on the project workflows, down to individual components, e.g., edit, launch, pause, restart and abort workitems, as well as processing tools and facilities [25].

Current experiments include industrial testcases for automobile aerodynamics optimization, involving the connection of the Matlab, Scilab, Python, ParaView and OpenFOAM software to the YAWL platform [3]. The YAWL workflow system is used to define the optimization processes, include the testcases and control their execution: this includes reading the input data (StarCCM+ files), the automatic invocation of the external software and automatic control passing between the various application components, e.g., Matlab scripts, OpenFOAM, ParaView.

B. Exception handling

The exception handlers are automatically tested by the YAWL workflow engine when the corresponding tasks are invoked. This is standard in YAWL and constraint checking can be activated and deactivated by the users [4].

For example, if a particular workflow task WT invokes an external EXEC code through a shell script SH (Figure 1) using a standard YAWL *codelet*, an exception handler EX can be implemented to prevent from undesirable situations, e.g., infinite loops, unresponsive programs, long network delays, etc. Application variables can be tested, allowing for very close monitoring of the applications behavior, e.g., unexpected values, convergence rates for optimization programs, threshold transgressions, etc.

A set of rules (RDR) is defined in a standard YAWL *exlet* attached to the task WT and defines the exception handler EX. It is composed here of a constraint checker CK, which is automatically tested when executing the task WT. A compensation action CP triggered when a constraint is violated and a notifier RE warning the user of the exception. This is used to implement resilience [26].

The constraint violations are defined by the users and are part of the standard exception handling mechanism provided by YAWL. They can attach sophisticated exception handlers in the form of specific *exlets* that are automatically triggered at runtime when particular user-defined constraints are violated. These constraints are part of the RDR attached to the workflow tasks.

Resilience is the ability for applications to handle unexpected behavior, e.g., erratic computations, abnormal result values, etc. It is inherent to the applications logic and programming. It is therefore different from systems or hardware errors and failures. The usual fault-tolerance mechanisms are therefore inappropriate here. They only cope with late symptoms, at best.

C. *Resilience*

Resilience is the ability for applications to handle unexpected behavior, e.g., erratic computations, abnormal result values, etc. It lies at the level of application logic and programming, not at systems or hardware level. The usual fault-tolerance mechanisms are therefore inappropriate here. They only cope with very late symptoms, at best.

New mechanisms are therefore required to handle logic discrepancies in the applications, most of which are only discovered at run-time [26].

It is therefore important to provide the users with powerful monitoring features and complement them with dynamic tools to evolve the applications according to the erratic behavior observed.

This is supported here using the YAWL workflow system so called "dynamic selection and exception handling mechanism". It supports:

- Application update using dynamically added rules specifying new codes to be executed, based on application data values, constraints and exceptions.
- The persistence of these new rules to allow applications to handle correctly future occurrences of the new case.
- The dynamic extension of these sets of rules.
- The definition of the new codes to be executed using the framework provided by the YAWL application specification tool: the new codes are just new workflows included in the global composite application specification.
- Component workflows invoke external programs written in any programming language through shell scripts, custom service invocations and Web Services.

In order to implement resilience, two particular YAWL features are used:

- Ripple-down-rules (RDR) which are handlers for exception management,
- Worklets, which are actions to be taken when exceptions or specific events occur.

The RDR define the decision process which is run to decide which worklet to use in specific circumstances.

D. *Distributed workflows*

The distributed workflow is based on an interface between the YAWL engine and the ProActive middleware (Figure 2). At the application level, users provide a specification of the simulation applications using the YAWL Editor. It supports a high-level abstract description of the simulation processes. These processes are decomposed into components which can be other workflows or basic workitems. The basic workitems invoke executable tasks, e.g., shell scripts or custom services. These custom services are specific execution units that call user-defined YAWL services. They support interactions with external and remote codes. In this particular platform, the external services are invoked through the middleware interface.

This interface delegates the distributed execution of the remote tasks to the ProActive middleware [17]. The middleware is in charge of the distributed resources allocation to the individual jobs, their scheduling, and the coordinated execution and result gathering of the individual tasks composing the jobs. It also takes in charge the fault-tolerance related to hardware, communications and system failures. The resilience, i.e., the application-level fault-tolerance is handled using the rules described in the previous Sections.

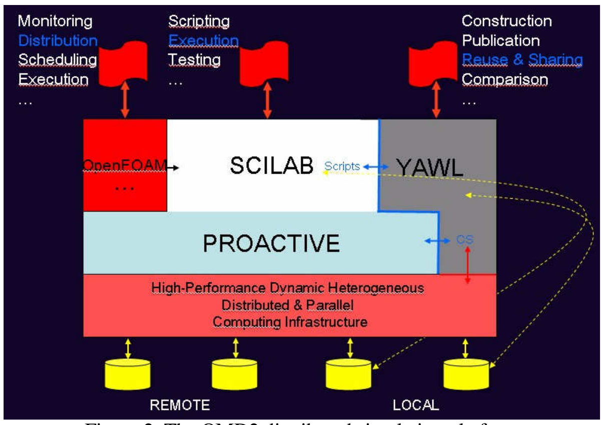

Figure 2. The OMD2 distributed simulation platform

The remote executions invoke the middleware functionalities through a Java API. The various modules invoked are the ProActive Scheduler, the Jobs definition module and the tasks which compose the jobs (Figure 3). The jobs are allocated to the distributed computing resources based upon the scheduler policy. The tasks are dispatched based on the job scheduling and invoke Java executables, possibly wrapping code written in other programming languages, e.g., Matlab, Scilab, Python, or calling other programs, e.g., CATIA, STAR-CCM+, ParaView, etc.

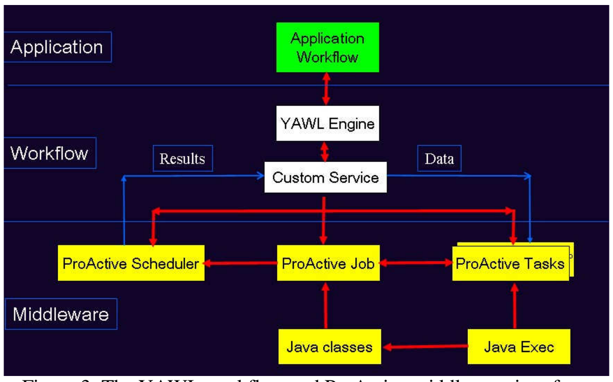

Figure 3. The YAWL workflow and ProActive middleware interface.

Optionally, the workflow can invoke local tasks using shell scripts and remote tasks using Web Services. These options are standard in YAWL.

E. Secured access

In contrast with the use of middleware, there is also a need to preserve and comply with the reservation and scheduling policies on the various HPC resources and clusters that are used. This is the case for national, e.g., IDRIS and CINES in France, and transnational HPC centers, e.g., PRACE in Europe.

Because some of the software run on proprietary resources and are not publicly accessible, some privileged connections must also be implemented through secured X11 tunnels to remote high-performance clusters (Figure 4). This also allows for fast access to software needing almost real-time answers, avoiding the constraints associated with the middleware overhead. It also allows running parallel optimization software on large HPC clusters. In this perspective, a both-ways SSH tunnel infrastructure has been implemented for the invocation of remote optimization software running on high-performance clusters and for fast result gathering.

Using the specific ports used by the communication protocol (5000) and YAWL (8080), a fast communication infrastructure is implemented for remote invocation of testcase optimizers between several different locations on a high-speed (40 GB/s) network at INRIA. This is also accessible through standard Internet connections using the same secured tunnels.

Current tests have been implemented monitoring from Grenoble in France a set of optimizers software running on HPC clusters in Sophia-Antipolis near Nice. The optimizers are invoked as custom YAWL services from the application workflow. The data and results are transparently transferred through secured SSH tunnels.

In addition t the previous interfaces, direct local access to numeric software, e.g., SciLab and OpenFOAM, is available through the standard YAWL custom services using the 8080 communication port and shell script invocations. Therefore, truly heterogeneous and distributed environments can be built here in a unified workflow framework.

F. Interfaces

To summarize, the simulation platform which is based on the YAWL workflow management system for the application specification, execution and monitoring, provides three complementary interfaces that suit all potential performance, security, portability and interoperability requirements of the current sophisticated simulation environments.

These interfaces run concurrently and are used transparently for the parallel execution of the different parts of the workflows. These interfaces are:

- The direct access to numeric software through YAWL custom services that invoke Java executables and shell scripts that trigger numeric software, e.g., OpenFOAM, and visualization tools, e.g., ParaView
- The remote access to high-performance clusters running parallel software, e.g., optimizers, through secured SSH tunnels, using remote invocations of custom services
- The access to wide-area networks through a grid middleware, e.g., ProActive, for distributed resource reservation and job scheduling

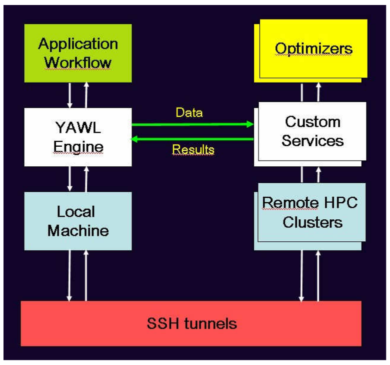

Figure 4. High-speed infrastructure for remote cluster access.

G. Service orchestration

The YAWL system provides a native Web service interface. This is a very powerful standard interface to distributed service execution, although it might impact HPC concerns. This is the reason why a comprehensive set of interfaces are provided by the platform (Section F, above).

Combined altogether and offered to the users, this rich set of functionalities is intended to support most application requirements, in terms of performance, heterogeneity and standardization.

Basically, an application workflow specifies general services orchestration. General services include here not only Web services, but also shell scripts, YAWL custom services implemented by Java class executables and high-level operators, as defined in the workflow control flow patterns of the Workflow Management Coalition [5, 21], e.g., AND-joins, XOR-joins, conditional branching, etc.

The approach implemented here therefore not only fulfills sound and semantically proved operators for task specification, deployment, invocation, execution and. synchronization. It also fulfills the requirements for heterogeneous distributed and HPC codes to be deployed and executed in a unified framework. This provides the users with high-level GUIs and hides the technicalities of distributed, and HPC software combination, synchronization and orchestration.

Further, because resilience mechanisms are implemented at the application level (Section C), on top of the middleware, network and OS fault-tolerance features, a secured and fault resilient HPC environment is provided, based on high-level constructs for complex and large-scale simulations.

The interface between the workflow tasks and the actual simulation codes can therefore be implemented as Web Services, YAWL custom services, and shell scripts through secured communication channels. This is a unique set of possibilities offered by our approach (Figure 5).

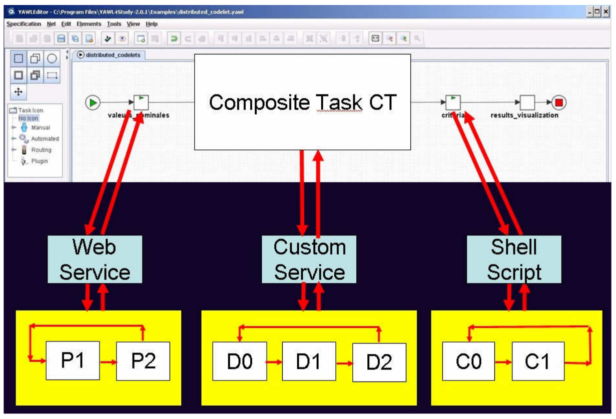

Figure 5. External services interfaces.

H. *Dataflow and control flow*

The dual requirements for the dataflow and control flow properties are preserved. Both aspects are important and address different requirements [6]. The control flow aspect addresses the need for user control over the workflow tasks execution. The dataflow aspect addresses the need for high-performance and parallel algorithms to be implemented effectively.

The control flow aspect is necessary to provide the users with global control over the synchronization and execution of the various heterogeneous and remote software that run in parallel to contribute to the application results. This is natively supported by YAWL.

The dataflow aspect is also preserved here in two complementary ways:

- the workflow data is transparently managed by the YAWL engine to ensure the proper synchronization, triggering and stopping of the tasks and complex operators among the different parallel branches of the workflows, e.g., AND joins, OR and XOR forks, conditional branching. This includes a unique YAWL feature called "cancellation set" that refers to a subset of a workflow that is frozen when another designated task is triggered [3]
- the data synchronization and dataflow scheme implemented by the specific numeric software invoked remain unchanged using a separation of concerns policy, as explained below

The various software with data dependencies that execute based on dataflow control are wrapped in adequate YAWL workflow tasks, so that the workflow engine does not interfere with the dataflow policies they implement.

This allows high-performance concerns to be taken into consideration along with the users concerns and expectations concerning the sophisticated algorithms associated with these programs.

Also, this preserves the global control flow approach over the applications which is necessary for heterogeneous software to cooperate in the workflow.

As a bonus, it allows user interactions during the workflow execution in order to cope with unexpected situations. This would otherwise be very difficult to implement because when unexpected situations occur while using a pure dataflow approach, it requires stopping the running processes or threads in the midst of possibly parallel and remote running calculations, while (possibly remote) running processes are also waiting for incoming data produced by (possibly parallel and remote) erratic predecessors in the workflow. This might cause intractable situations even if the errors are due to rather simple events, e.g., network data transfers or execution time-outs.

Note that so far, because basic tasks cannot be divided into remote components in the workflow, the dataflow control is not supported between remotely located software. This also avoids large uncontrolled data transfers on the underlying network. Thus, only collocated software, i.e., using the same computing resources or running on the same cluster, can use dataflow control on the platform. They are wrapped by workflow tasks which are controlled by the YAWL engine as standard workflow tasks.

For example, the dataflow controlled codes C0 and C1 depicted Figure 5 are wrapped by the composite task which is a genuine YAWL task that invokes a shell script to trigger them.

Specific performance improvements can therefore be expected from dataflow controlled sets of programs running on large HPC clusters. This is fully compatible with the control flow approach implemented at the application (i.e., workflow) specification level. Incidentally, this also avoids the streaming of large data collections of intermediate results through network connections. It therefore alleviates bandwidth congestion.

The platform interfaces are illustrated by Figure 5. Once the orchestration of local and distributed codes is specified at the application (workflow) level, their invocation is transparent to the user, whatever their localization.

I. *Experiments*

The current testcases include vehicle aerodynamics simulation (Figure 6) and air-conditioner pipes optimization (Figure 7). The distributed and heterogeneous platform is also tested with the Gmsh mesh generator (http://geuz.org/gmsh/) and the FAMOSA optimization suite developed at INRIA by project OPALE [34]. It is deployed on HPC clusters and invoked from remote workflows running on Linux workstations.

FAMOSA is an acronym for "*Fully Adaptive Multilevel Optimization Shape Algorithms*" and includes C++ components for:

- CAD generation,
- mesh generation,
- domain partitioning,
- parallel CFD solvers using MPI, and
- post-processors

The input is a design vector and the output is a set of simulation results. The various components are invoked by shell scripts. FAMOSA is currently tested by the PSA Automotive Company and ONERA (the French National Aerospace Research Office) for aerodynamics problem solving.

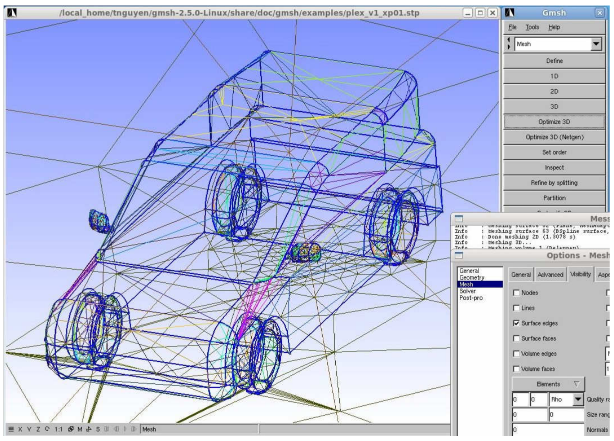

Figure 6. Vehicle mesh for aerodynamics simulation (Gmsh screenshot).

The various errors that are taken into account by the resilience algorithm include run-time errors in the solvers, inconsistent CAD and mesh generation files, and execution time-outs.

The FAMOSA components are here triggered by remote shell scripts including PBS invocations for each one on the HPC cluster. The shell scripts are called by YAWL custom service invocations from the user workflow running on the workstation.

Additionally, another experiment uses the distributed simulation platform for testing the heterogeneity of the application codes running on various hardware and software environments. It includes four remote computing resources that are connected by a high-speed network. One site is a HPC cluster. Another site is a standard Linux server. The two other sites are remote virtualized computing resources running Windows and Linux operating systems on different VirtualBox virtual machines that interface the ProActive middleware.

III. Conclusion

This paper presents an experiment for deploying a distributed simulation platform on grids. It uses a network of high-performance computers connected by a middleware layer. Users interact dynamically with the applications using a workflow management system. It allows them to define, deploy and control the application execution interactively.

In contrast with choreography of services, where autonomous software interact in a controlled manner, but where resilience and fault-tolerance are difficult to implement, the approach used here is an orchestration of heterogeneous and distributed software components that interact in a dynamic way under the user control [29]. This allows the dynamic interaction in case of errors and erratic application behavior. This approach is also fully compatible with both the dataflow and control flow approaches which are often described as poorly compatible [30, 31, 32] and are extensively used in numeric software platforms.

Because of the heterogeneity of the software and resources, the platform also combines secured access to remote HPC clusters and local software in a unified workflow framework.

This approach is also proved to combine in an elegant way the dataflow control used by many HPC software and the control flow approach required by complex and distributed application execution and monitoring.

A significant bonus of this approach is that the users can define and handle application failures at the workflow specification level. This means that a new abstraction layer is introduced to cope with application-level errors at run-time. Indeed, these errors do not necessarily result from programming and design errors. They may also result from unforeseen situations, data values and boundary conditions that were not envisaged at first. This is often the case for simulations, due to their experimental nature, e.g., discovering the behavior of the system being simulated.

This provides support for resiliency using an asymmetric checkpoint mechanism. This feature allows for efficient handling mechanisms to restart only those parts of the applications that are characterized by the users as necessary for overcoming erratic behavior.

Further, this approach can be evolved dynamically, i.e., when the applications are running. This uses the dynamic selection and exception handling mechanism in the YAWL workflow system. It allows for new rules and new exception handling to be added on-line if unexpected situations occur.

Acknowledgment

This work is supported by the French National Research Agency ANR (*Agence Nationale de la Recherche*), grant ANR-08-COSI-007, OMD2 project (*Optimisation Multi-Discipline Distribuée)*.

References

[1] Y. Simmhan, R. Barga, C. van Ingen, E. Lazowska and A. Szalay “Building the Trident Scientific Workflow Workbench for Data Management in the Cloud”. In proceedings of the *3rd Intl. Conf. on Advanced Engineering Computing and Applications in Science*. ADVCOMP’2009. Sliema (Malta). October 2009. pp 41-50.

[2] A. Abbas, High Computing Power: A radical Change in Aircraft Design Process, In proceedings of the *2nd China-EU Workshop on Multi-Physics and RTD Collaboration in Aeronautics*. Harbin (China) April 2009.

[3] T. Nguyên and J-A Désidéri, Dynamic Resilient Workflows for Collaborative Design, In proceedings of the 6th *Intl. Conf. on Cooperative Design, Visualization and Engineering*. Luxemburg. September 2009. Springer-Verlag. *LNCS 5738*, pp. 341–350 (2009)

[4] A.H.M ter Hofstede, W. Van der Aalst, M. Adams and N. Russell, Modern Business Process Automation: YAWL and its support environment, *Springer* (2010).

[5] N. Russel, A.H.M ter Hofstede and W. Van der Aalst. Workflow Control Flow Patterns. A Revised View. Technical Report. University of Eindhoven (NL). 2006.

[6] E. Deelman and Y. Gil., Managing Large-Scale Scientific Workflows in Distributed Environments: Experiences and Challenges, In proceedings of the 2nd IEEE Intl. Conf. on e-Science and the Grid. Amsterdam (NL). December 2006. pp 131-139.

[7] SUN VirtualBox, User Manual, 2010. http://www.virtualbox.org.

[8] M. Ghanem, N. Azam, M. Boniface and J. Ferris, Grid-enabled workflows for industrial product design, In proceedings of the 2nd Intl. Conf. on e-Science and Grid Computing. Amsterdam (NL). December 2006. pp 88-92.

[9] G. Kandaswamy, A. Mandal and D.A. Reed, Fault-tolerant and recovery of scientific workflows on computational grids, In proceedings of the 8th Intl. Symp. On Cluster Computing and the Grid. 2008. pp 777-782.

[10] H. Simon. "Future directions in High-Performance Computing 2009- 2018". Lecture given at the ParCFD 2009 Conference. Moffett Field (Ca). May 2009.

[11] J. Wang, I. Altintas, C. Berkley, L. Gilbert and M.B. Jones, A high-level distributed execution framework for scientific workflows, In proceedings of the *4th IEEE Intl. Conf. on eScience*. Indianapolis (In). December 2008. pp 156-164.

[12] D. Crawl and I. Altintas, A Provenance-Based Fault Tolerance Mechanism for Scientific Workflows, In proceedings of the *2nd Intl. Provenance and Annotation Workshop*. IPAW 2008. Salt Lake City (UT). June 2008. Springer. LNCS 5272. pp 152-159.

[13] M. Adams, A.H.M ter Hofstede, W. Van der Aalst and N. Russell, Facilitating Flexibility and Dynamic Exception Handling in Workflows through Worklets, Technical report, Faculty of Information Technology, Queensland University of Technology, Brisbane (Aus.), October 2006.

[14] M. Adams and L. Aldred, The worklet custom service for YAWL, Installation and User Manual, Beta-8 Release, Technical Report, Faculty of Information Technology, Queensland University of Technology, Brisbane (Aus.), October 2006.

[15] L. Ramakrishnan et al., VGrADS: Enabling e-Science workflows on grids and clouds with fault tolerance. Proc. ACM SC'09 Conf. Portland (Or.), November 2009. pp 145-152.

[16] M. Baudin, Introduction to Scilab", Consortium Scilab. January 2010. Also: http://wiki.scilab.org/

[17] F. Baude et al., Programming, composing, deploying for the grid. in "GRID COMPUTING: Software Environments and Tools", Jose C. Cunha and Omer F. Rana (Eds), Springer Verlag, January 2006.

[18] http://edition.cnn.com/2009/TRAVEL/01/20/mumbai.overview last accessed: 07/07/2010.

[19] J. Dongarra et al. "The International Exascale Software Project Roadmap". University of Tennessee EECS Technical report UT-CS-10-654. May 2010. Available at: http://www.exascale.org/

[20] R. Gupta, et al. "CIFTS: a Coordinated Infrastructure for Fault-Tolerant Systems". Proc. 38th Intl. Conf. Parallel Processing Systems. Vienna (Au). September 2009.pp 145-154.

[21] The Workflow Management Coalition. http://www.wfmc.org

[22] D. Abramson, B. Bethwaite et al. "Embedding Optimization in Computational Science Workflows". Journal of Computational Science 1 (2010). Pp 41-47. Elsevier.

[23] A.Bachmann, M. Kunde, D. Seider and A. Schreiber. "Advances in Generalization and Decoupling of Software Parts in a Scientific Simulation Workflow System". Proc. 4th Intl. Conf. Advanced Engineering Computing and Applications in Sciences. Florence (I). October 2010. pp 133-139.

[24] R. Duan, R. Prodan and T. Fahringer. "DEE: a Distributed Fault Tolerant Workflow Enactment Engine for Grid Computing". Proc. 1st. Intl. Conf. on High-Performance Computing and Communications. Sorrento (I). LNCS 3726. September 2005. pp 265-278.

[25] http://www.yawlfoundation.org/software/documentation.The YAWL foundation. 2010.

[26] T. Nguyên, L. Trifan and J-A Désidéri. A Distributed Workflow Platform for Simulation. Proc. 4th Intl. Conf on Advanced Engineering Computing and Applications in Sciences. Florence (I). October 2010. pp 321-329.

[27] Object Management Group / Business Process Management Initiative. BPMN Specifications. http://www.bpmn.org, last accessed: 12/01/2011.

[28] OASIS Web Services Business Process Execution Language. http://www.oasisopen.org/committees/tc_home.php?=wg_abbrev=wsbpel last accessed: 12/01/2011.

[29] Sherp G., Hoing A., Gudenkauf S., Hasselbring W. and Kao O. Using UNICORE and WS-BPEL for Scientific Workfow execution in Grid Environments. Proc. EuroPAR 2009. LNCS 6043. . Springer. 2010. pp 455-461.

[30] Ludäscher B.,Weske M., McPhillips T. and Bowers S. Scientific Workflows: Business as usual ? Proc. BPM 2009. LNCS 5701. Springer. 2009. pp 351-358.

[31] Montagnat J., Isnard B., Gatard T., Maheshwari K. and Fornarino M. A Data-driven Workflow Language for Grids based on Array Programming Principles. Proc. SC 2009 4th Workshop on Workflows in Support of Large-Scale Science. WORKS 2009. Portland (Or). ACM 2009. pp 235-242.

[32] Yildiz U., Guabtni A. and Ngu A.H. Towards Scientific Workflow Patterns. Proc. SC 2009 4th Workshop on Workflows in Support of Large-Scale Science. WORKS 2009. Portland (Or). ACM 2009. pp 121-129.

[33] Plankensteiner K., Prodan R. and Fahringer T. Fault-tolerant Behavior in State-of-the-Art Grid Workflow Management Systems. CoreGRID Technical Report TR-0091. October 2007. http://www.coregrid.net

[34] Duvigneau R. and Chandrashekaran P. A three-level parallelization strategy for robust design in aerodynamics. Proc. 20th Intl. Conf. on Parallel Computational Fluid Dynamics. May 2008. Lyon (F). pp 101-108.