A Dynamic Workflow Simulation Platform

Toan Nguyen, Laurentiu Trifan

A Dynamic Workflow Simulation Platform

Laurentiu Trifan
INRIA
655 av. de l'Europe
Montbonnot
38334 Saint Ismier (France)
Email: laurentiu.trifan@inria.fr

Toàn Nguyên
INRIA
655 av. de l'Europe
Montbonnot
38334 Saint Ismier (France)
Email: toan.nguyen@inria.fr

***Abstract*—In numeric optimization algorithms errors at application level considerably affect the performance of their execution on distributed infrastructures. Hours of execution can be lost only due to bad parameter configurations. Though current grid workflow systems have facilitated the deployment of complex scientific applications on distributed environments, the error handling mechanisms remain mostly those provided by the middleware. In this paper, we propose a collaborative platform for the execution of scientific experiments in which we integrate a new approach for treating application errors, using the dynamicity and exception handling mechanisms of the YAWL workflow management system. Thus, application errors are correctly detected and appropriate handling procedures are triggered in order to save as much as possible of the work already executed.**

Keywords: HPC, Fault Tolerance, Service Oriented Architectures, Large Scale Scientific Computing

I. Introduction

Numeric optimization applications are continuously improving and their complexity is growing very fast. This produces a considerable boost in research areas like aerodynamics, electromagnetism, structural mechanics and biology. The price to pay is an increased demand for computational power. Cluster and grid systems and the evolving cloud infrastructures represent viable answers to these demands. But the distributed and heterogeneous nature of their components, combined with the growing complexity of the multidiscipline applications, render them hard to manage, especially for inexperienced scientists. To address all these issues, the concept of scientific workflow system has been proposed [1].

But with the evolution of exascale systems, composed of hundreds of thousands of cores, failures augment in variation and size and the Mean Time Between Failure (MTBF) parameter will be in the range of minutes [7]. Thus, a critical characteristic of a workflow system is its resilience. When executing a scientific workflow experiment on a grid infrastructure, failures can occur due to numerous reasons: resource unavailable, hardware failure, system out of memory, dynamic resource relocation, usually detected by the middleware, but also application errors like faulty mesh operations, bad input parameters, solver errors (lack of convergence, time-outs), which are more difficult to detect and treat. Without the capacity to address such errors dynamically, one must either interrupt the execution and manually adapt the workflow for new updated scenarios or even worse, restart the workflow execution completely [6]. This means a significant time loss and poor execution performance.

Even though there is currently an acceptable offer of distributed workflow systems [1], [2], [3], [4] with different fault tolerant mechanisms [1], [8], [9], very few have addressed this aspect from a dynamic perspective [5]. We present in this paper a new approach to resilience based on dynamicity and exception handling techniques. The workflow system on which we drive our research is called YAWL (Yet Another Workflow Language) [10], characterized by its extensible architecture, intelligent dynamicity and exception handling services [19], [20].

The rest of this paper is organized as follows: in Section 2, we present some related work concerning fault-tolerance in scientific workflow systems. Section 3 presents the general architecture of the YAWL workflow management system with specific details on its service-oriented characteristics. Section 4 describes in more details the dynamicity and exception handling mechanisms developed for YAWL and how we use these mechanisms to meet the resilience requirements. A prototype collaborative platform that we develop is presented in Section 5, with some extension proposals in Section 6. At the end, in Section 7, some conclusions are presented.

II. Related Work

Dynamicity is implemented in various forms in some of the current scientific workflow systems, and in others it lacks completely [5]. For example, in Kepler [31], a workflow is considered static, so it must be completely specified at design time. Askalon, one of the most complete workflow system on the market, mentions run-time steering and dynamicity [11], but this functionality is not detailed in any of the resources we found. Pegasus works on an abstract to concrete workflow approach, leaving no opportunity for dynamic changes since the users must operate on an entire workflow statically [12].

As mentioned in [29], faults in workflow distributed platforms occur at different levels: Hardware, Operating System (OS), Middleware, Task, Workflow or User level. If low-level errors like hardware (machine crashes, network connectivity), OS (memory and disk limitations) or middleware (non-responsive services, authentication, job submission problems) can be detected by most of the workflow systems (70% on

average), it is not the same for higher levels errors (uncaught exceptions, missing shared libraries, data movements, infinite loops, user-definable exceptions and assertions), only 40% of them being detected on average. Weaker results have been registered concerning the recovery aspects, especially for higher levels, where less than 20% of the systems can recover from errors produced at those levels. A lot is still to be achieved concerning error prediction and prevention, features that are almost non-existent in the current versions of grid workflow systems.

We think that with its context based dynamicity and exception handling techniques [19], [20], YAWL is a viable candidate to address the issues presented above and which are not taken into account by current systems. Adopting existing fault-management techniques, like light- and heavy-weight checkpointing or restart procedures [1], [13], [8], [14], [15], we want to enhance resiliency by dynamically detecting application faults and apply context-based treatments so that computation loses are reduced to minimum.

III. YAWL General Architecture

Designed after a rigorous analysis of existing workflow systems, YAWL aims at covering functional aspects left unimplemented by previous workflow systems. Having a workflow language formally based on Petri nets and implementing the well-known workflow patterns [16], YAWL extends these concepts with dedicated constructs to deal with patterns like cancellation, synchronization of active branches only and multiple concurrently executing instances of the same task. Even more, YAWL is based on a service-oriented architecture (Figure 1) that greatly contributes to its extensibility.

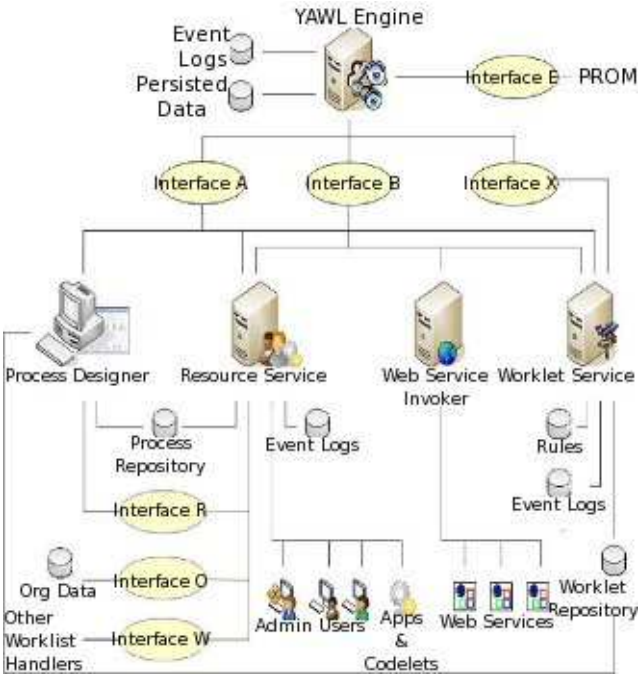

Fig. 1. YAWL General Architecture

Beside the built-in services available in the standard distribution of YAWL, users can also implement their own services using the concept of Custom Service [17]. The development of a Custom Service must follow a fixed interface through which it will communicate with the YAWL engine using HTTP messages. It can be deployed locally or remotely with respect to the engine's location and the registration is done by specifying basic authentication credentials and a location in the form of a base "URL". A Custom Service is responsible for the execution of all workitems with which it is associated, that are being checked out of the engine and then checked back in (Figure 2). Between these two stages, the developer of the service has full control over the functionality implemented by the service.

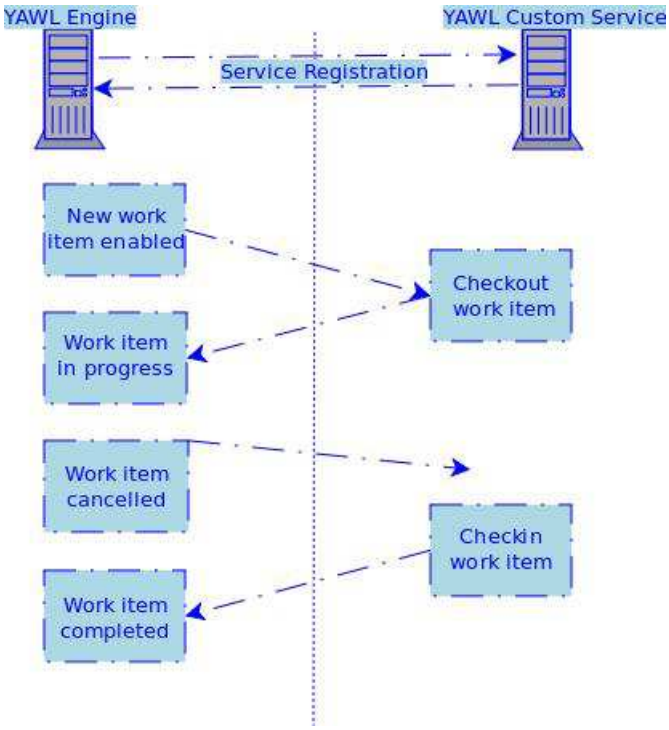

Fig. 2. YAWL Custom Service

IV. Dynamicity and Exception Handling in YAWL

A. Theoretical View

YAWL departs from other service-oriented workflow systems by two important built-in services: the "Worklet Dynamic Process Selection Service" [18], [19] and the "Exception Handling Service" [20]. The former implements dynamicity requirements by providing each task of a process instance with the ability to be linked to an extensible repertoire of actions, called worklets (Figure 3-up). Being self-contained workflow processes, these worklets allow the right actions to be executed at run-time, depending on a specific run-time context. The context capture and the selection of the associated worklet is realized by the evaluation of a hierarchical set of rules, which are modified Ripple Down Rules (RDR) (Figure 3-down), an

enactment of a popular knowledge representation system [21].

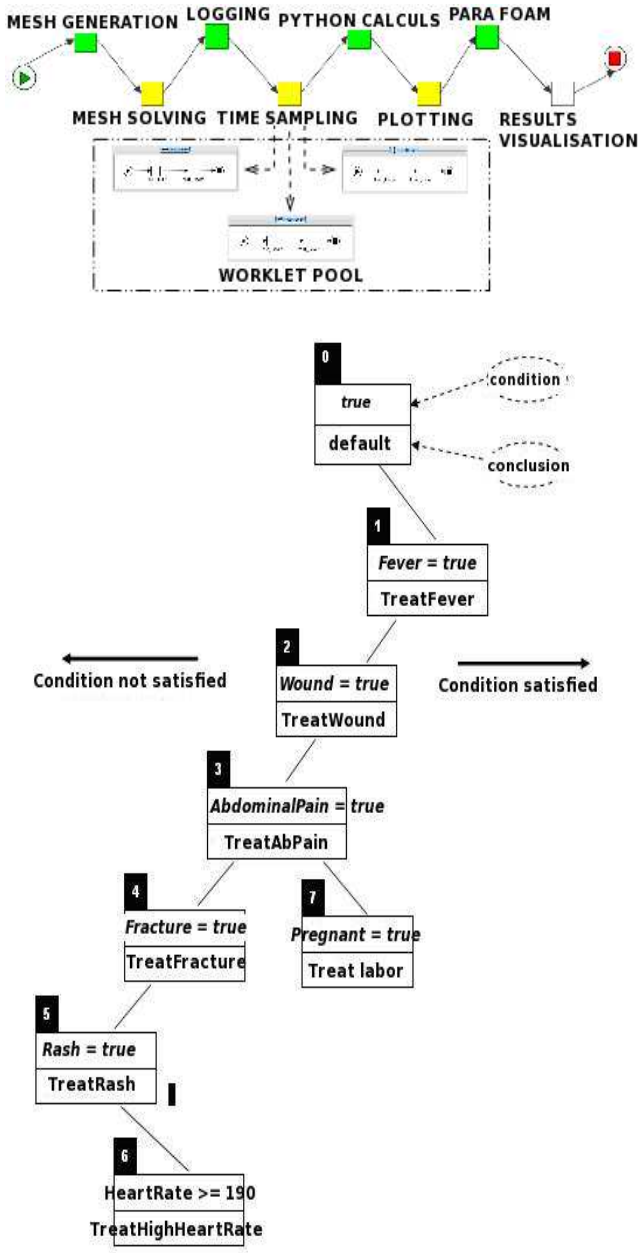

Fig. 3. Dynamic Process Selection Service & Ripple Down Rules

The exceptions are handled by the "Exception Handling Service". It allows the users to define specific exception handling procedures, called "exlets" (Figure 4), for up to 10 different exception variants, ranging from data typing to externally triggered exceptions. An exlet contains one or more exception handling primitives, which are defined and executed sequentially. These primitives analyze whether it is the task that generated the exception or the whole process, and perform actions like restart, cancellation, forced continuation or forced failure. Further, an exlet can also contain in its exception handling chain compensation actions that will be chosen from the same repertoire of actions, as described above for the Selection Service. Again, the selection of the appropriate exlet is accomplished through modified RDRs.

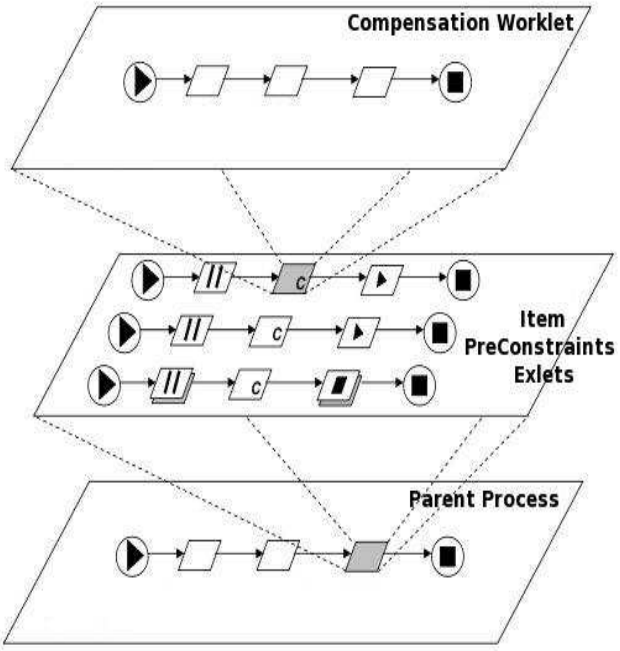

Fig. 4. Exception Handling Service

B. A Timeout Example

In order to illustrate the theoretical concepts presented earlier we will use a simple timeout example. Figure 5 shows a workflow specification called "Timer_Test" aiming at executing a shell script. Beside the usual variables specific for a shell execution application like "dir" to specify the directory location of the shell script and "command" to identify the script, we added a new variable called "nrRestarts" that will count the number of times a task is restarted. Every variable has a usage attribute stating if it can be written to - "input" or read from - "output"(Figure 6). The executing task is "Timer_Task" that has an associated Yawl Custom Service in charge of the actual execution. The following task, "Timer_Check", will compare the value of the "nrRestarts" variable to a fixed threshold and while it is under this threshold the "Timer_Task" will be re-executed. Otherwise, the application will continue its normal flow by showing the final execution result and finish. The transfer between values of variables belonging to different tasks is intermediated by net-level variables using XQuery expressions (Figure 6). Beside the Yawl Custom Service, the "Timer_Task" is also provided with a timer of 10s. When this timer expires a timeout exception is triggered by the system. A RDR file is created for the "timer_test.yawl" called "timer_test.xrs". In this file is specified that if a timeout exception occurs for the "Timer_Task" task then an exlet is enabled that will suspend the curent workitem, execute a compensation worklet and then continue the execution of

the workitem (Figure 5 - second level). The compensation worklet is "TimeOutWorklet.yawl" that contains only one task called "TimeOutTreatment" in which the "nrRestarts" variable is incremented by 1 ((Figure 5 - third level)). The "TimeOutTreatment" must be associated with a second Yawl Custom Service that will perform the incrementation. This way after a while the counter will reach the threshold value, quitting the loop and finishing the execution. The timeout occurrence is forced by using a call to the "wait" fuction inside the shell script.

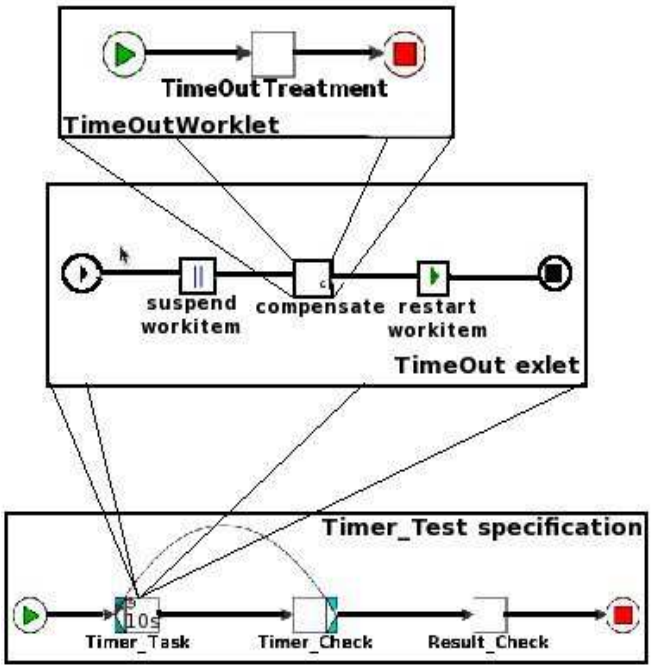

Fig. 5. Time Out Exception Handling Example

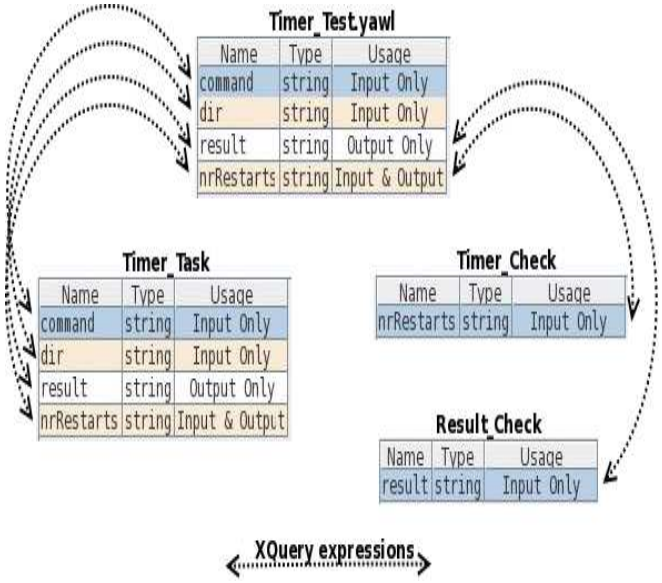

Fig. 6. Yawl Data Transfer

V. A Collaborative Platform Prototype

Starting from the service-oriented architecture of YAWL, we have built a collaborative platform in order to execute numeric optimization applications. They are modeled as YAWL workflow specifications executing on distributed resources. The main features that we expect from this platform are:

- Non-intrusive - the platform must not modify the user codes to be executed.
- Transparent access to remote code and data.
- Standardized interfaces with numeric tools - through script invocations, web services and custom services.
- Resiliency - fault-tolerance at application level.

For testing purposes, we used test-cases (Figure 7) provided by an industry partner from the OMD2 project [30]. The cases are geometry modeling and optimization applications with different levels of complexity and computing demands. The first two represent a 2D and 3D air conditioner pipes with computational time less than 1 minute and 10 minutes, respectively, based on a standard 32 computers cluster. The third one is an engine cylinder head with a CPU time of 100 hours and the last one represents the aerodynamics of a van with a computational time in the order of 1000 hours of CPU. The current state of the collaborative platform prototype is

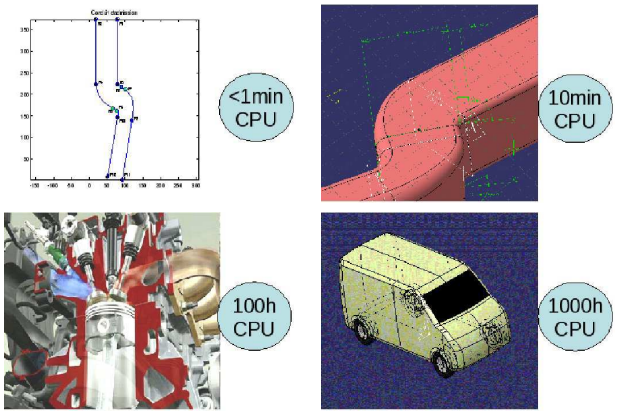

Fig. 7. Test Cases

presented in Figure 8. First, we installed the YAWL engine on a standard computer and then we developed a YAWL Custom Service that we installed on front-ends for clusters from INRIA centres, in Sophia-Antipolis and Grenoble. In the YAWL engine, we register the deployed services using their URLs. Then we split the second test-case in atomic tasks and we assign them to the available custom services for execution. On the front-ends we create the submission scripts, one for each task, invoking the corresponding executable code. Of course, all the needed software tools, like OpenFOAM, Python, GnuPlot, Matlab, are previously installed on the clusters. So basically, when the workflow specification is started, the tasks are executed in the specified order by exporting the corresponding work item to the attached YAWL Custom Service. The latter reads the input data (command to execute,

directory of submission script), launches the command and sends the results back to the workflow engine.

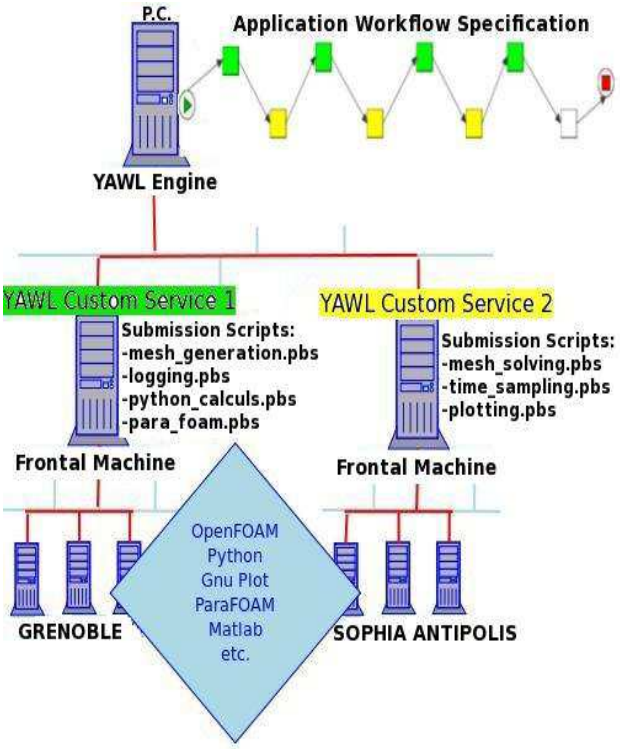

Fig. 8. Collaborative Platform Prototype

If errors like bad input parameters, missing data or faulty mesh operations are absent, corresponding to correctly formed test-cases, we may however encounter errors during the workflow execution. Most noticeable would be broken links to the OpenFOAM libraries, unavailable computing nodes or infinite loops in the submission scripts. In the absence of a resiliency mechanism, we would have to restart execution from the beginning. By detecting these errors using the YAWL exception mechanism, we can apply an exception handling procedure that can either restart only the problematic task or assign it to a service located on a different computing node. The result of the test-case execution is shown in Figure 9 using the ParaView tool.

VI. Extensions and Further Improvements

A. Middleware Interface

ProActive(PA) [22] is an open-source middleware, offered as a Java library, aiming to simplify the programming of multi-threaded, parallel and distributed applications for grids, multi-core clusters and data centers. With a small set of primitives, ProActive provides an API allowing the development of parallel applications on distributed systems using the deployment framework. This framework, composed of the PA Scheduler [23] and the PA Resource Manager [24], plays an important part in the final collaborative platform, as it handles the actual scheduling of the tasks on computing resources along with

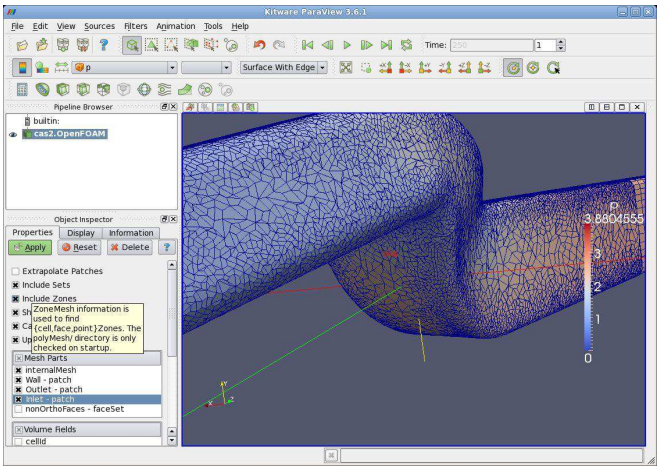

Fig. 9. Test Case 2 Result

the management of the distributed resources. Beside this, its role is also to detect errors at system and hardware level and either handle them locally, if possible, or send them to the workflow level. In Figure 10 is a schema describing the interconnection between YAWL and the PA Scheduler through a Custom Service. Currently, we are interfacing YAWL and ProActive in the prototype platform.

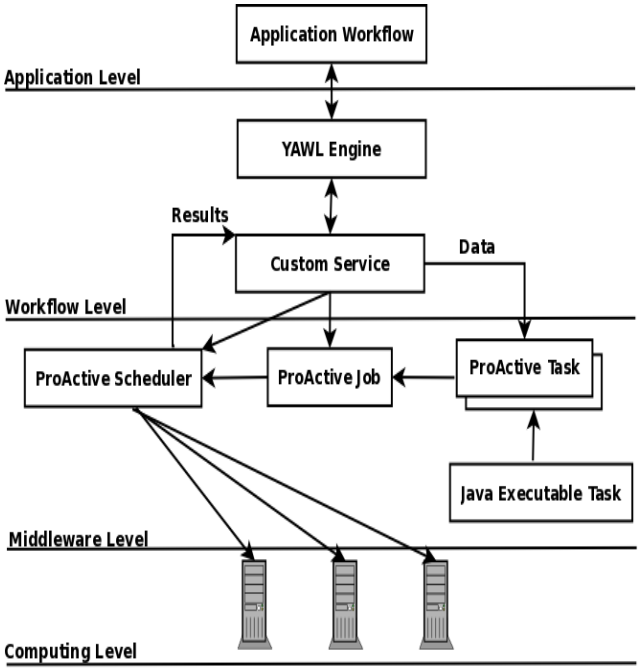

Fig. 10. Interaction Between ProActive and YAWL

B. Virtual Infrastructure Deployment

The PA Resource Manager supports several type of computing infrastructures. The one that can leverage our future large-scale applications is the Virtual Infrastructure. First of all, using an infrastructure based on virtual machines allows

rapid deployment tests without the need for real distributed platforms. Even more, to deploy the workflow platform on a real distributed infrastructure, like Grid5000 [25], it will be a lot easier to start virtual machines with all the required services already installed and configured using system images.

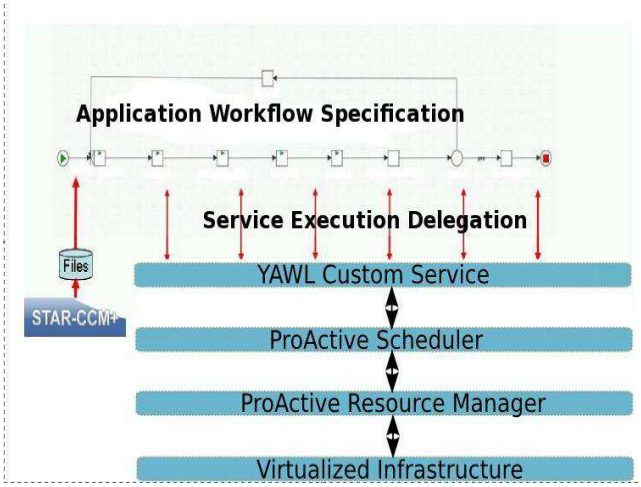

Fig. 11. PA Resource Manager - Virtual Infrastructure

VII. Conclusion

In this paper, we present a method for handling errors in numeric optimization applications by making use of the dynamicity and exception handling capabilities offered by the YAWL workflow management system. We propose a prototype of a collaborative platform that ensures the communication between the different levels of the execution: application, middleware and computing resources. We have tested this prototype on 2D and 3D geometry optimization test-cases. Current work is performed to integrate the concepts of worklets and exlets in the platform in order to make it resilient.

References

[1] Jia Yu and Rajkumar Buyya, *A Taxonomy of Workflow Management Systems for Grid Computing*, J. Grid Comp. Volume 3, Numbers 3-4, 171-200, September 2005.

[2] S. Gudenkauf, W. Hasselbring, A. Hing, G. Scherp, O. Kao, *Using UNICORE and WS- BPEL for Scientific Workflow Execution in Grid s Environments*, Proceedings of 5th UNICORE Summit 2009 in conjunction with EuroPar 2009, Delft, The Netherlands.

[3] Jianwu Wang, Ilkay Altintas, Chad Berkley, Lucas Gilbert, Matthew B. Jones, *A High- Level Distributed Execution Framework for Scientific Workflows*, Proceedings of the 2008 Fourth IEEE International Conference on eScience.

[4] DIET User's Manual, Ch. 14, Workflow management in Diet, April 2010.

[5] Manuel Caeiro-Rodriguez, Thierry Priol, Zsolt Nmeth, *Dynamicity in Scientific Workflows*, CoreGRID Technical Report, 31 August 2008.

[6] M.Adams, T.A. H.M. Hofstede, D.Edmond, W.M.P. van der Aalst, *Implementing dynamic flexibility in workflows using worklets*, BPM Center Report BPM-06-06, BPMcen- ter.org, Tech. Rep., 2006.

[7] International EXASCALE Software Project, ROADMAP 1.0,May 2010.

[8] Yang Xiang, Zhongwen Li, Hong Che, *Optimizing Adaptive Checkpointing Schemes for Grid Workflow Systems*, Fifth International Conference on Grid and Cooperative Computing Workshops, 2006.

[9] Rubing Duan, Radu Prodan, Thomas Fahringer, *DEE: A Distributed Fault Tolerant Workflow Enactment Engine for Grid Computing*, The 2005 International Conference on High Performance Computing and Communications (HPCC-05), Copyright (C) Springer-Verlag, LNCS 3726, Proceedings.September 21 - 25, 2005, Sorrento, Italy.

[10] YAWL Official Website.

[11] Askalon Official Website.

[12] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi, Miron Livny, *Pegasus: Mapping Scientific Workflows onto the Grid*, Lecture Notes in Computer Science, 2004, Volume 3165/2004, 131-140.

[13] Xavier Besseron, Thierry Gautier, *Optimised Recovery with a Coordinated Checkpoint/Rollback Protocol for Domain Decomposition Applications*, Communications in Computer and Information Science, 2008.

[14] Nichamon Naksinehaboon, Yudan Liu , Chokchai (Box) Leangsuksun, Raja Nassar, Mihaela Paun, Stephen L. Scott, *Reliability-Aware Approach: An Incremental Checkpoint/Restart Model in HPC Environments*, Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID), May 2008.

[15] Oliner A, Rudolph L, Sahoo R, *Cooperative checkpointing theory*, Parallel and Distributed Processing Symposium, 2006, IPDPS 2006.

[16] Workflow Patterns.

[17] YAWL Technical Manual, 2010.

[18] W.M.P van der Aalst, M. Adams, A.H.M. ter Hofstede, M. Pesic, and H. Schonenberg, *Flexibility as a Service*, BPM Center Report BPM-08-09, BPMcenter.org, 2008.

[19] Michael Adams, Arthur ter Hofstede, David Edmond, W.M.P. van der Aalst, *Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows*, Proceedings of the 14th International Conference on Cooperative Information Systems (CoopIS 06), Montpellier, France, November, 2006.

[20] Michael Adams, Arthur H.M. ter Hofstede, Wil M.P. van der Aalst, David Edmond, *Dynamic, Extensible and Context-Aware Exception Handling for Workflow*, Proceedings of the 15th International Conference on Cooperative Information Systems (CoopIS 2007), Vilamoura, Algarve, Portugal, November 2007.

[21] P.Compton, G.Edwards, B.Kang, L.Lazarus, R.Malor, T.Menzies, P.Preston. A.Srinivasan, S.Sammut, *Ripple down rules: possibilities and limitations*, 6th Banff AAAI Knowledge Acquisition for Knowledge, 1991.

[22] ProActive official website.

[23] The OASIS Research Team and ActiveEon Company, *"ProActive Scheduling" - Manual*, INRIA, Universit Nice, CNRS, January 2011.

[24] The OASIS Research Team and ActiveEon Company, *"ProActive Resource Manager" - Manual*, INRIA, Universit Nice, CNRS, January 2011.

[25] Grid 5000 Official Website.

[26] Zong Wei Luo, *Checkpointing for workflow recovery*, ACM Southeast Regional Conference, 2000.

[27] Toàn Nguyên, Laurentiu Trifan, Jean-Antoine Désidéri, *Resilient Workflows for Computational Mechanics Platforms*, WCCM/APCOM, Sydney, July 2010.

[28] A. Rozinat, M. T. Wynn, W. M. P. van der Aalst, A. H. M. ter Hofstede, and C. J. Fidge, *Workflow Simulation for Operational Decision Support using YAWL and ProM*, BPM Center Report BPM-08-04, BPMcenter.org, 2008.

[29] Kassian Plankensteiner, Radu Prodan, Thomas Fahringer, Attila Kertesz, Peter Kacsuk, *Fault-tolerant behavior in state-of-the-art Grid Workflow Management Systems*, CoreGRID Technical Report Number TR-0091 October 18, 2007.

[30] OMD2 Official Website.

[31] Bertram Ludaescher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, Yang Zhao, *Scientific Workflow Management and the Kepler System*, Concurrency and Computation: Practice & Experience, 18(10):1039-1065, 2006.