A Distributed Workflow Platform for High-Performance Simulation

Toàn Nguyên
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 an approach to design, implement and deploy 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 are often stored in proprietary locations and cannot be moved. Distributed computing infrastructures are therefore necessary [6][8].

This article explores 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 is prone to run-time errors, sophisticated mechanisms are necessary to handle unexpected failures at the infrastructure and system levels [19]. This is also true for the coupled software that contribute to large simulation applications [35]. Consequently, specific management software is required to handle unexpected application and software behavior [9][11][12][15].

This paper addresses these issues. Section II is an overview of related work. Section III is a general description of a sample application, infrastructure, systems and application software. Section IV addresses fault-tolerance and resiliency issues. Section V gives an overview of the implementation using the YAWL workflow management system [4]. Section VI is a conclusion.

II. RELATED WORK

Simulation is nowadays a prerequisite for product design and for scientific breakthrough in many application areas ranging from pharmacy, biology to climate modeling that also require extensive simulation testing. This requires often large-scale experiments, including the management of petabytes volumes of data and large multi-core supercomputers [10].

In such application environments, various teams usually collaborate on several projects or part of projects. Computerized tools are often shared and tightly or loosely coupled [23]. Some codes may be remotely located and non-movable. This is supported by distributed code and data management facilities [29]. And unfortunately, this is prone to a large variety of unexpected errors and breakdowns [30].

Most notably, data replication and redundant computations have been proposed to prevent from random hardware and communication failures [42], as well as failure prediction [43], sometimes applied to deadline-dependent scheduling [12].

System level fault-tolerance in specific programming environments are proposed, e.g., CIFTS [20], FTI [48]. Also, middleware usually support mechanisms to handle fault-tolerance in distributed job execution, usually calling upon data replication and redundant code execution [9][15][22][24].

Also, erratic application behavior needs to be supported [45]. This implies evolution of the simulation process in the event of such occurrences. Little has been done in this area [33][46]. The primary concerns of the designers, engineers and users have so far focused on efficiency and performance [47] [49] [50]. Therefore, application unexpected behavior is usually handled by re-designing and re-programming pieces of code and adjusting parameter values and bounds. This usually requires the simulations to be stopped and restarted.

A dynamic approach is presented in the following sections. It support the evolution of the application behavior using the introduction of new exception handling rules at run-time by the users, based on occurring (and possibly unexpected) events and data values. The running workflows do not need to be suspended in this approach, as new rules

can be added at run-time without stopping the executing workflows.

This allows on-the-fly management of unexpected events. This approach also allows a permanent evolution of the applications that supports their continuous adaptation to the occurrence of unforeseen situations [46]. As new situations arise and data values appear, new rules can be added to the workflows that will permanently take them into account in the future. These evolutions are dynamically hooked onto the workflows without the need to stop the running applications. The overall application logics is therefore maintained unchanged. This guarantees a constant adaptation to new situations without the need to redesign the existing workflows. Further, because exception-handling codes are themselves defined by new ad-hoc workflows, the user interface remains unchanged [14].

III. TESTCASE APPLICATION

A. *Example*

This work is performed for the OMD2 project (*Optimisation Multi-Discipline Distribuée*, i.e., Distributed Multi-Discipline Optimization) supported by the French National Research Agency ANR.

An overview of two running testcases is presented here. It deals with the optimization of an auto air-conditioning system [36]. The goal of this particular testcase is to optimize the geometry of an air conditioner pipe in order to avoid air flow deviations in both pressure and speed concerning the pipe output (Figure 1). Several optimization methods are used, based on current research by public and industry laboratories.

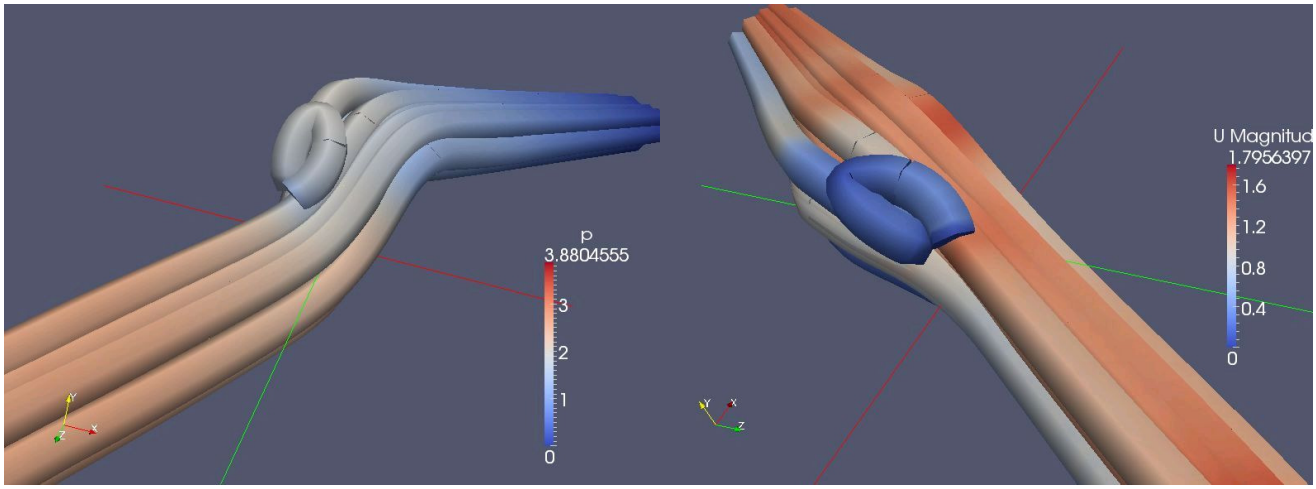


Figure 1. Flow pressure (left) and speed (right) in an air-conditioner pipe (ParaView screenshots).

This example is provided by a car manufacturer and involves several industry partners, e.g., software vendors, and academic labs, e.g., optimization research teams (Figure 1).

The testcases are a dual faceted 2D and 3D example. Each facet involves different software for CAD modeling, e.g. CATIA and STAR-CCM+, numeric computations, e.g., Matlab and Scilab, and flow computation, e.g., OpenFOAM and visualization, e.g., ParaView (Figure 12, at the end of this paper).

The testcases are deployed on the YAWL workflow management system [4]. The goal is to distribute the testcases on various partners locations where the software are running (Figure 2). In order to support this distributed computing approach, an open source middleware is used [17].

A first step is implemented using extensively the virtualization technologies (Figure 3), i.e., Oracle VM VirtualBox, formerly SUN's VirtualBox [7]. This allows hands-on experiments connecting several virtual guest computers running heterogeneous software (Figure 10, at the end of this paper). These include Linux Fedora Core 12, Windows 7 and Windows XP running on a range of local workstations and laptops (Figure 11, at the end of this paper).

B. *Application Workflow*

In order to provide a simple and easy-to-use interface to the computing software, a workflow management system is used (Figure 2). It supports high-level graphic specification for application design, deployment, execution and monitoring. It also supports interactions among heterogeneous software components. Indeed, the 2D example testcase described in Section III.A involves several codes written in Matlab, OpenFOAM and displayed using ParaView (Figure 7). The 3D testcase involves CAD files generated using CATIA and STAR-CCM+, flow calculations using OpenFOAM, Python scripts and visualization with ParaView. Extensions allow also the use of the Scilab toolbox.

Because proprietary software are used, as well as open-source and in-house research codes, a secured network of connected computers is made available to the users, based on existing middleware (Figure 8).

This network is deployed on the various partners locations throughout France. Web servers accessed through the SSH protocol are used for the proprietary software running on dedicated servers, e.g., CATIA v5 and STAR-CCM+.

An interesting feature of the YAWL workflow system is that composite workflows can be defined hierarchically [13]. They can also invoke external software, i.e., pieces of code written in whatever language suits the users. They are called by custom YAWL services or local shell scripts. Remote Web services can also be called.

YAWL thus provides an abstraction layer that helps users design complex applications that may involve a large number of distributed components (Figure 6). Further, the workflow specifications involve possible alternative execution paths, as well as parallel branches, conditional branching and loops. Combined with the run-time addition of code with the corresponding dynamic selection procedures as well as new exception handling procedures (see Section IV), a very powerful environment is provided to the users.

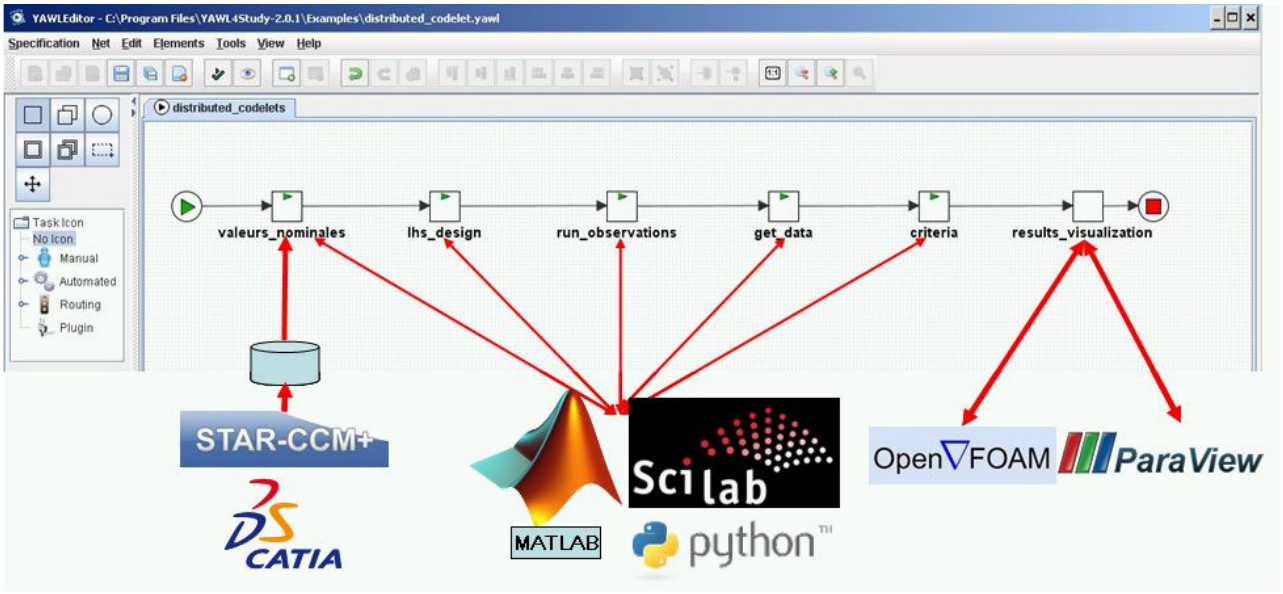


Figure 2. The YAWL workflow interface to the 2D testcase.

IV. RESILIENCE

A. *Rationale*

Resilience is defined here as the ability of the applications to handle unexpected situations. Usually, hardware, communication and software failures are handled using fault-tolerance mechanisms [15]. This is the case for communication software and for middleware that take into account possible computer and network breakdowns at run-time. These mechanisms use for example data and packet replication and redundant code execution to cope with these situations [5].

However, when unexpected situations occur at run-time, very few options are usually offered to the application users: ignore them or abort the execution, reporting the errors and analyze them, to later modify and restart the applications.

B. *Exception Handling*

Another alternative is proposed here. It is based on the dynamic selection and exception handling mechanism featured by YAWL [13].

It provides the users with the ability to add at run-time new rules governing the application behavior and new pieces of code that will take care of the new situations.

For example, it allows for the selection of alternative code, based on the current unexpected data values. The application can therefore evolve over time without being stopped. It can also cope later with the new situations without being altered. This refinement process is therefore lasting over time and the obsolescence of the code greatly reduced.

The new codes are defined and inserted in the application workflow using the standard specification approach used by YAWL (Figure 7). This is implemented by YAWL so-called exlets that are in charge of exception and error handling. They can be inserted and invoked at run-time in cas of task failure.

For example (Figure 24, at the end of this paper) if a workflow is specified as a sequence of tasks T0, T1 and T2 and that a failure occurs for task T1, an exlet is automatically invoked and takes the form of a dynamic insertion of a set of tasks that cope for the error (Error Handler, Restore and Ignore). It is based on a pre-defined or dynamically provided scenario by the user. The Error Handler task then triggers the Restore task or the Ignore task, based on appropriate

decisions made, depending on parameters values or user interactions. In case the Restore task is invoked, the scenario then backtracks the execution of the workflow to the nearest checkpoint CHKPT before the failed task T1. In contrast, if the decision is made to ignore the error, the control is passed to the task immediately following T1 in the original scenario, i.e., T2.

Because it is important that monitoring long-running applications be closely controlled by the users, this dynamic selection and exception handling mechanism also requires a user-defined probing mechanism that provides them with the ability to suspend, evolve and restart the code dynamically.

For example, if the output pressure of an air-conditioning pipe is clearly off limits during a simulation run, the user must be able to suspend it as soon as he is aware of that situation. He can then take corrective actions, e.g., suspending the simulation, modifying some parameters or value ranges and restarting the process.

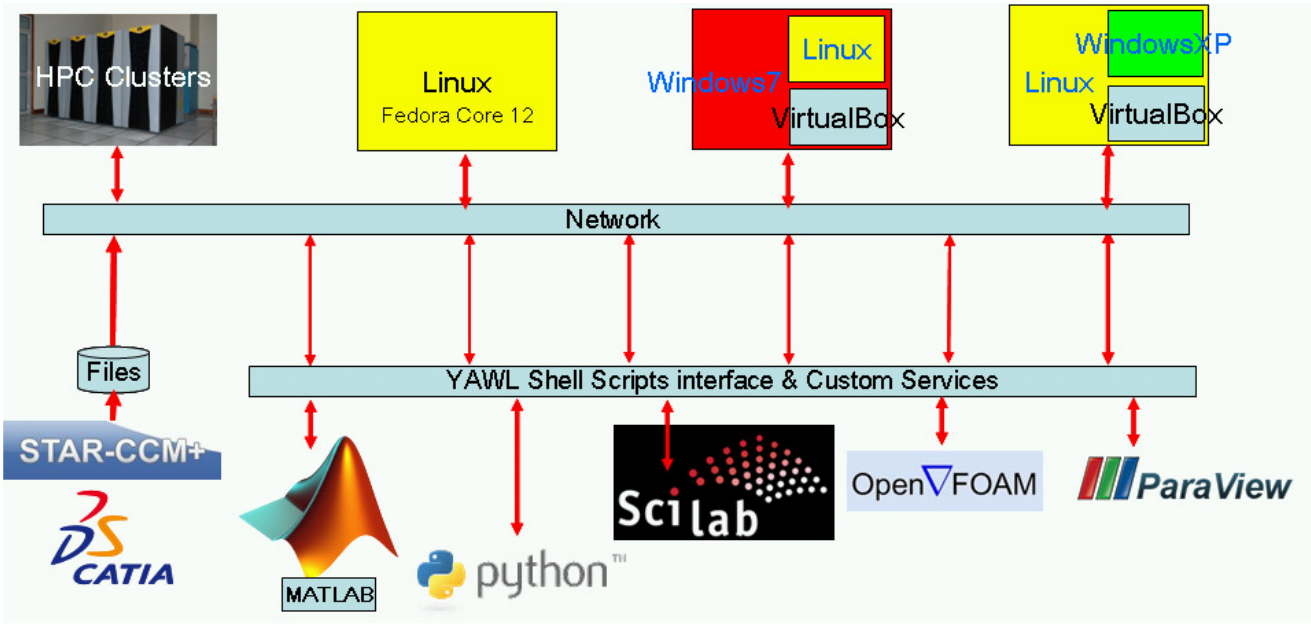


Figure 3. The virtualized infrastructure.

C. *Fault-tolerance*

The fault-tolerance mechanism provided by the underlying middleware copes with job and communication failures. Job failures or time-outs are handled by reassignment and re-execution. Communication failures are handled by re-sending appropriate messages. Also, hardware breakdowns are handled by re-assigning running jobs to other resources, implying possible data movements to the corresponding resources. This is standard for most middleware [17].

D. *Assymetric Checkpoints*

Asymmetric checkpoints are defined by the users at significant execution locations in the application workflows. They are used to avoid the systematic insertion of checkpoints at all potential failure points. They are user-defined at specific critical locations, depending only on the application logic. Clearly, the applications designers and users are the only ones that have the expertise necessary to insert the appropriate checkpoints. In contrast with middleware fault-tolerance which can re-submit jobs and resend data packets, no automatic procedure can be implemented here. It is therefore based on a dynamically evolving set of heuristic rules.

As such, this approach significantly reduces the number of necessary checkpoints to better concentrate on only those that have a critical impact on the applications runs.

For example (Figure 4):

- The checkpoints can be chosen by the users among those that follow long-running components and large data transfers.
- Alternatively, those that precede series of small components executions.

The base rule set on which the asymmetric checkpoints are characterized is the following:

- R1: no output backup for specified join operations
- R2: only one output backup for fork operations
- R3: no intermediate result backup for user-specified sequences of operations
- R4: no backup for user-specified local operations
- R5: systematic backup for remote inputs

This rule set can be evolved by the user dynamically, at any time during the application life-time, depending on the specific application requirements.

V. IMPLEMENTATION

A. *The YAWL workflow management system*

Workflows systems are the support for many e-Science applications [6][8][26]. Among the most popular systems are Taverna, Kepler, Pegasus, Bonita and many others [11][15]. They complement scientific software environments 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 [40] is investigating the use of the YAWL workflow management system for distributed multidiscipline optimization [3]. The goal is to develop a resilient workflow system for large-scale optimization applications. It is based on extensions to the YAWL system to add resilience and remote computing facilities for deployment on high-performance distributed infrastructures. This includes large-PC clusters connected to broadband networks. It also includes interfaces with the Scilab scientific computing toolbox [16] and the middleware [17].

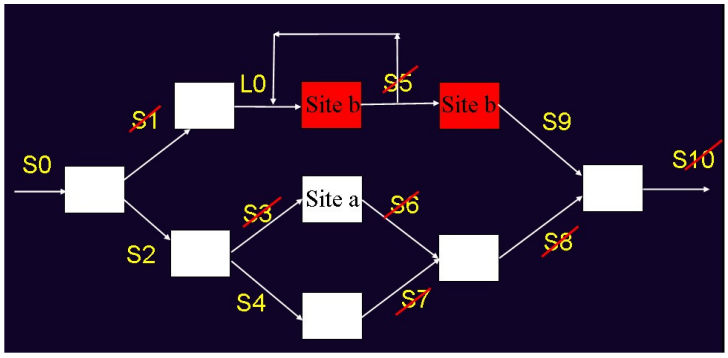


Figure 4. Asymmetric checkpoints.

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 (Figure 5). 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 (Figure 10, at the end of this paper).

Formally, it is based on a sound and proven operational semantics extending the *workflow patterns* of the Workflow Management Coalition [21][32]. It is implemented and proved by colored Petri nets. This allows for sophisticated verifications of workflow specifications at design time: fairness, termination, completeness, deadlocks, etc (Figure 5-left).

In contrast, workflow 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 (Figure 17, at the end of this paper). There are indeed over 73 (supposedly compliant) implementations of the BPMN, as of February 2011, and several others are currently being implemented [27]. In addition, there are more than 20 existing BPEL engines. 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 advanced synchronization mechanisms [28].

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 (Figure 6).

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 (Figure 5-right) [25].

Current experiments include industrial testcases, 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 (Figure 11, at the end of this paper).

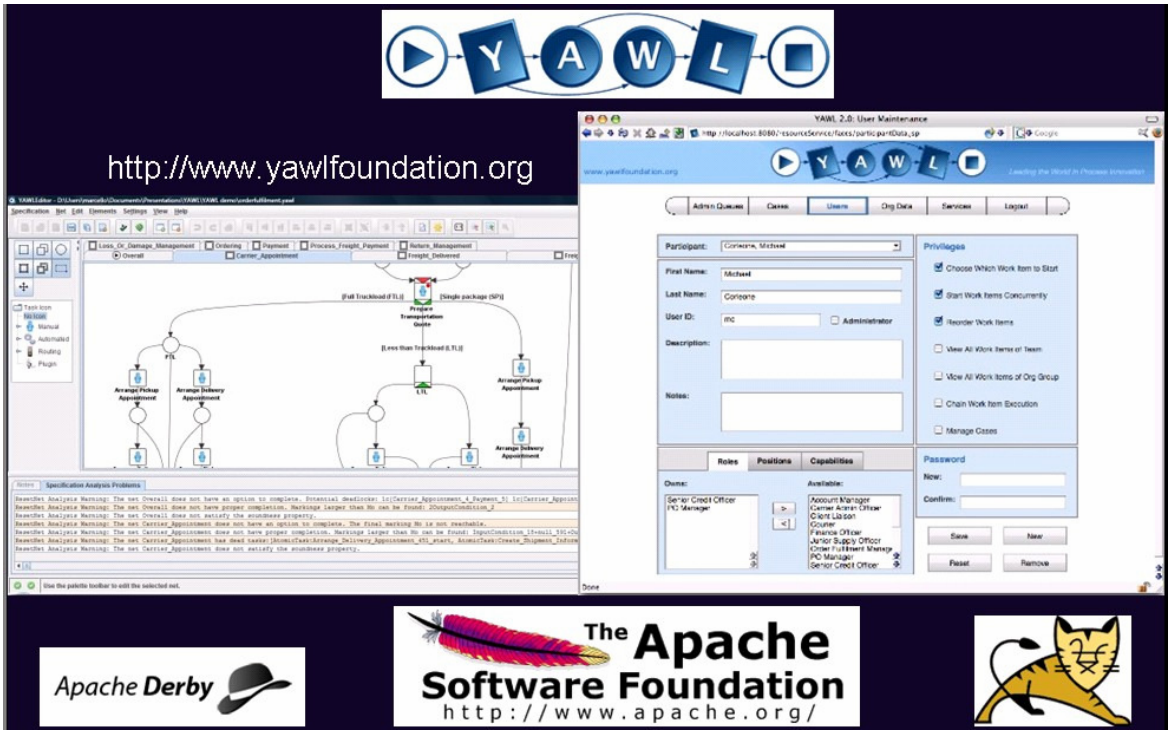


Figure 5. The user interfaces: YAWL Editor (left) and YAWL Control Center (right).

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 7) 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 (Section V. C.).

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.

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:

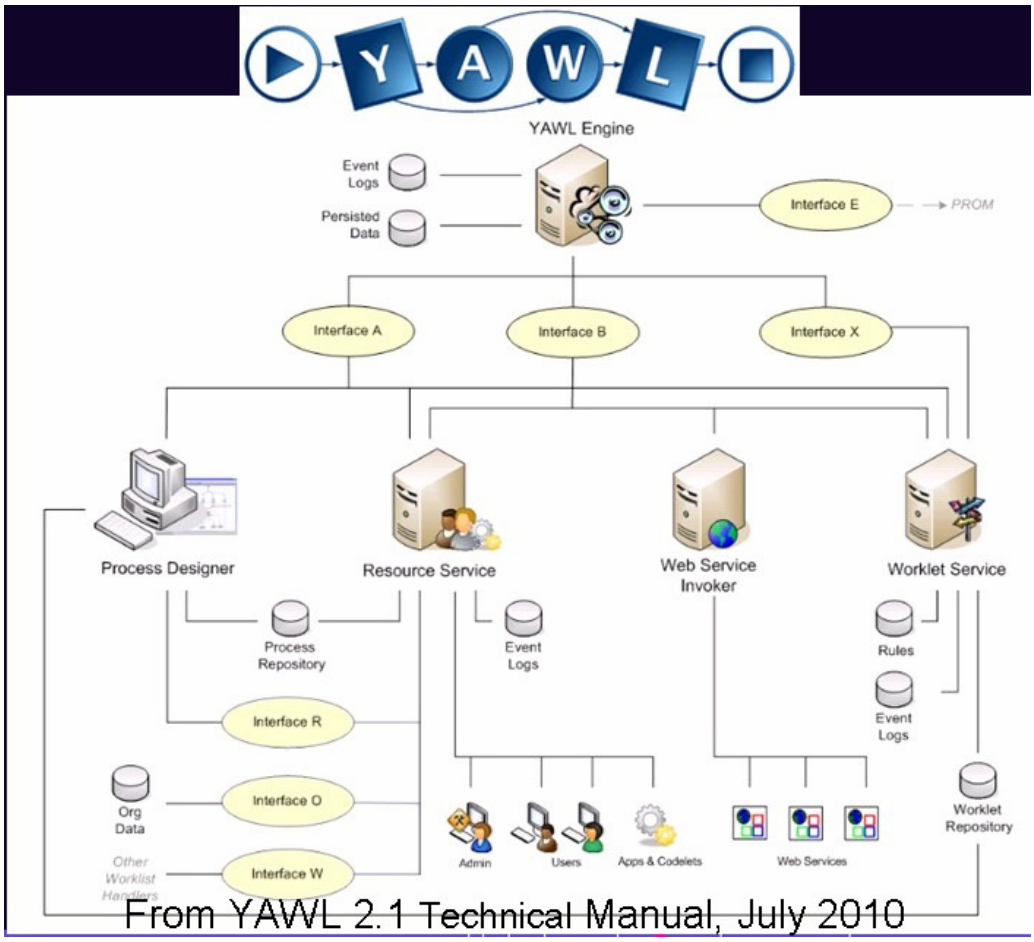


Figure 6. YAWL architecture.

- 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 new workflows included in the global application workflow 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 particular workflow 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 underlying middleware (Figure 8). 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 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.

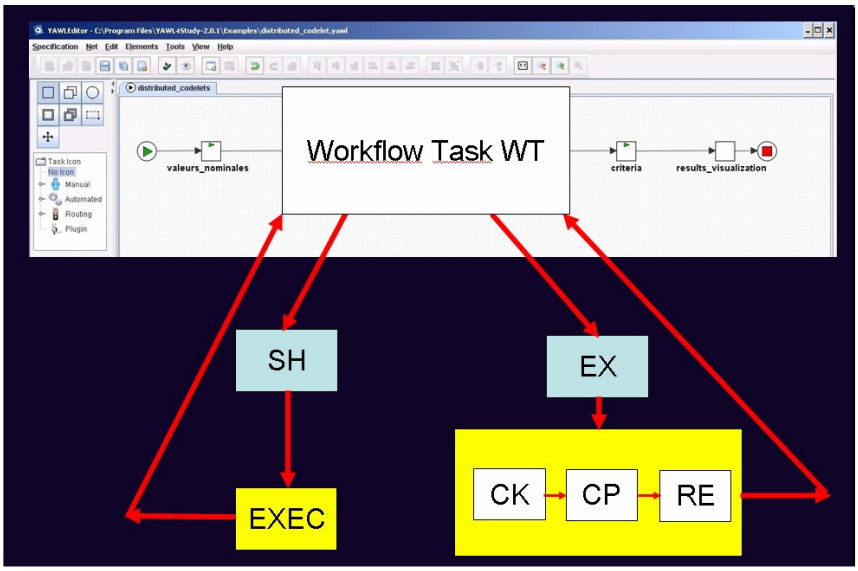


Figure 7. Exception handler associated with a workflow task.

The remote executions invoke the middleware functionalities through a Java API. The various modules invoked are the middleware Scheduler, the Jobs definition module and the tasks which compose the jobs. 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.

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 13). 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 (10 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 always

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 (Figure 14). 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 (Figure 2)
- The remote access to high-performance clusters running parallel software, e.g., optimizers, through secured SSH tunnels, using remote invocations of custom services (Figure 13)
- The access to wide-area networks through a grid middleware, e.g., Grid5000, for distributed resource reservation and job scheduling (Figure 9)

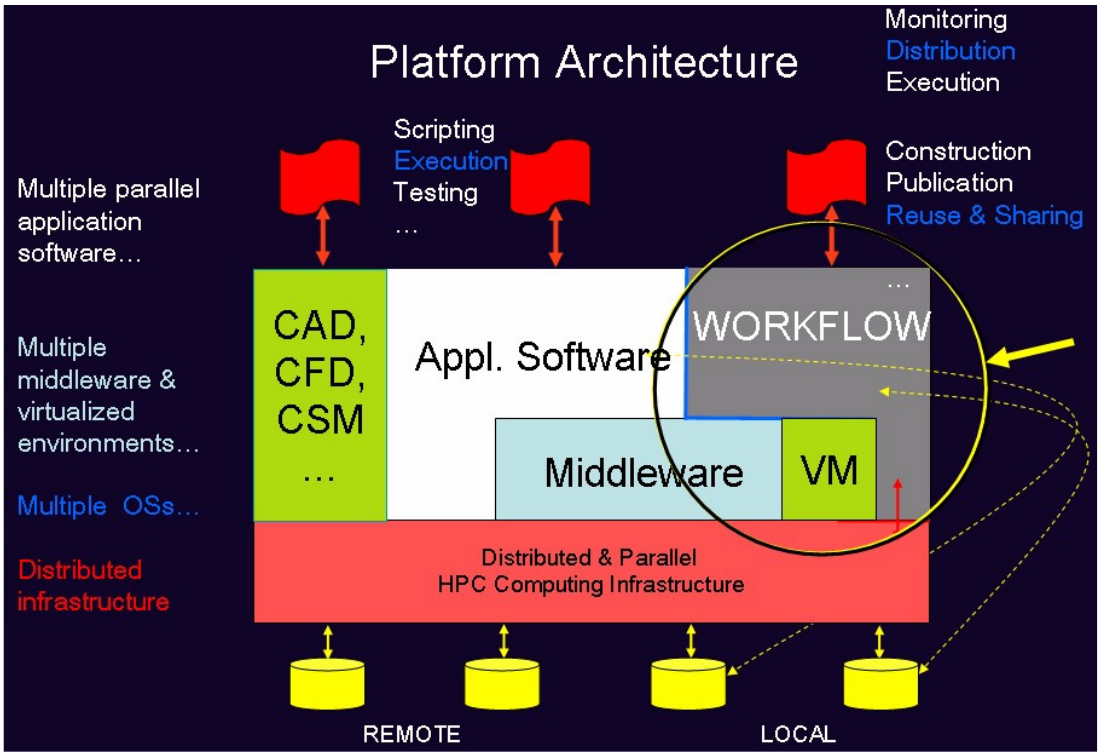


Figure 8. The distributed simulation platform.

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 branchings, 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 stringent 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 [41].

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 14).

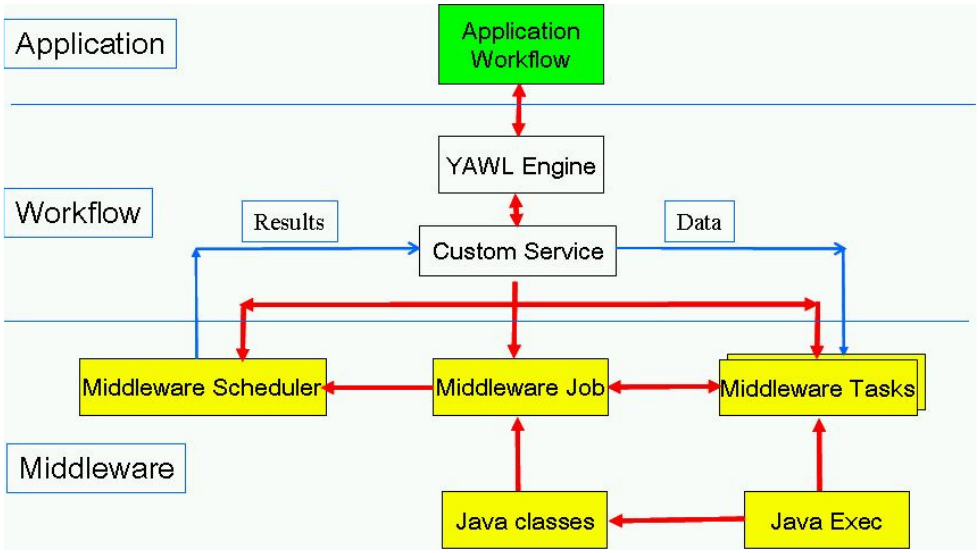


Figure 9. The YAWL workflow and middleware interface.

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 required in order to provide the users with control over the synchronization and execution of the various heterogeneous and remote software that run in parallel and contribute to the application results. This aspect is exemplified in the previous sections (Secttion III) where multiple software contribute to the application results and visualization. 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 branchings. 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 dataflow dependencies 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 (Section IV). 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.

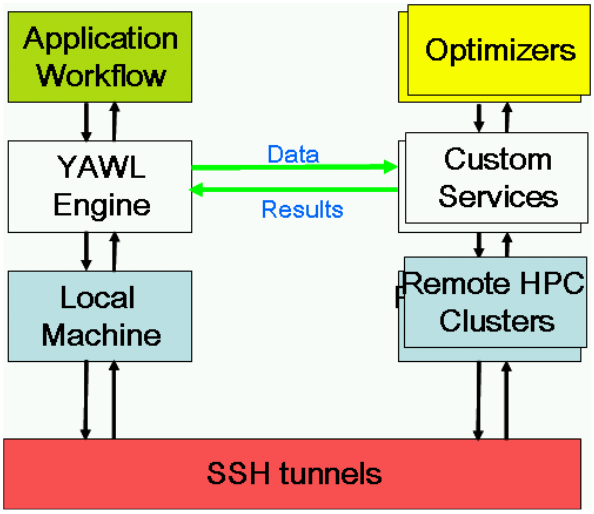


Figure 13. High-speed infrastructure for remote cluster access.

For example, the dataflow controlled codes D0, D1 and D2 depicted Figure 15 are wrapped by the composite task CT which is a genuine YAWL task that invokes a shell script SH 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 16, at the end of this paper. 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. Other experiments

This distributed and heterogeneous platform is also tested with the FAMOSA optimization suite developed at INRIA by project OPALE [34]. It is deployed on a HPC cluster and invoked from a remote workflow running on a Linux workstation (Figure 18, at the end of this paper).

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 (Figure 19, at the end of this paper). The components also include other software for mesh generation, e.g., Gmsh [37], partitioning, e.g., Metis [38] and solvers, e.g., Num3sis [39]. They are remotely invoked from the YAWL application workflow by shell scripts (Figure 18).

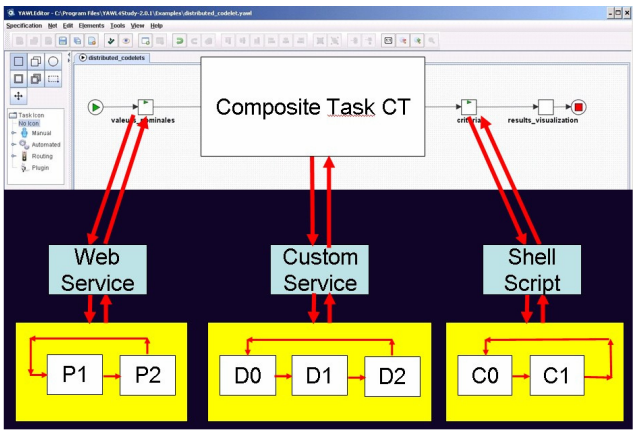


Figure 14. External services interfaces.

FAMOSA is currently tested by an auto manufacturer (Figure 21, at the end of this paper) and ONERA (the French National Aerospace Research Office) for aerodynamics problem solving (Figure 25 and 26).

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 running 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 (Figure 18).

Additionally, another experiment described by Figure 20 illustrates the distributed simulation platform used 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 (Site 4). Another site is a standard Linux server (Site 1). The two other sites are remote virtualized computing resources running Windows and Linux operating systems on different VirtualBox virtual machines that interface the underlying middleware (Sites 3 an 4). This platform has been tested against the testcases described in Section III.

VI. CONCLUSION

The requirements for large-scale simulation make it necessary to deploy various software components on heterogeneous distributed computing infrastructures [10, 44]. These environments are often required to be distributed among a number of project partners for administrative and collaborative purposes.

This paper presents an experiment for deploying a distributed simulation platform. 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 executions 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, in order to contribute to the application results [29]. This allows the dynamic interaction with the users 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.

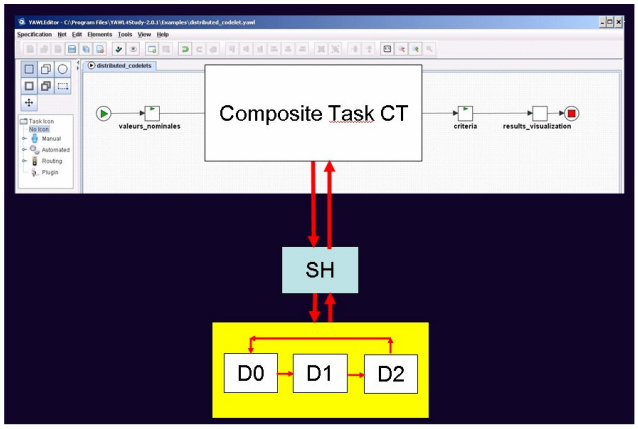


Figure 15. Dataflow tasks wrapped by a composite YAWL task.

The underlying interface to the distributed components is a middleware providing resource allocation and job scheduling [17]. Because of the heterogeneity of the software and resources used, the platform also combines secured access to remote HPC clusters and local software in a unified workflow framework (Figure 20, at the end of this paper).

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 besides fault-tolerance provided by the middleware, which handles communication, hardware and job failures, the users can define and handle application logic 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 limit conditions that could not be envisaged. 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 to resiliency using an asymmetric checkpoint mechanism. This feature allows for efficient handling mechanisms to restart only those parts of an application that are characterized by the users as necessary for overcoming erratic behavior.

Further, this approach can be evolved dynamically, i.e., when 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 at run-time.

ACKNOWLEDGMENT

This work is supported by the European Commission FP7 Cooperation Program "Transport (incl. aeronautics)", for the *GRAIN* Coordination and Support Action *("Greener Aeronautics International Networking")*, grant ACS0-GA-2010-266184. It is also supported by the French National Research Agency ANR (*Agence Nationale de la Recherche*), OMD2 project (*Optimisation Multi-Discipline Distribuée)*, grant ANR-08-COSI-007, program COSINUS (*Conception et Simulation*).

REFERENCES

[1] T. Nguyên and J-A Désidéri, "A Distributed Workflow Platform for Simulation", Proc. 4th Intl. Conf on Advanced Engineering Computing and Applications in Sciences (ADVCOMP2010), Florence (I), October 2010, pp. 375-382.

[2] A. Abbas, "High Computing Power: A radical Change in Aircraft Design Process", Proc. 2nd China-EU Workshop on Multi-Physics and RTD Collaboration in Aeronautics, Harbin (China) April 2009, pp. 115-122.

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

[4] W. Van der Aalst et al., "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", Proc. 2nd IEEE Intl. Conf. on e-Science and the Grid, Amsterdam (NL), December 2006, pp. 211-222.

[7] Oracle VM VirtualBox, User Manual, 2011. http://www.virtualbox.org Retreived: January, 2012.

[8] M. Ghanem, N. Azam, M. Boniface, and J. Ferris, "Grid-enabled workflows for industrial product design", Proc. 2nd Intl. Conf. on e-Science and Grid Computing, Amsterdam (NL), December 2006, pp. 325-336.

[9] G. Kandaswamy, A. Mandal, and D.A. Reed., "Fault-tolerant and recovery of scientific workflows on computational grids", Proc. 8th Intl. Symp. On Cluster Computing and the Grid, Lyon (F), May 2008, pp. 167-176.

[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", Proc. 4th IEEE Intl. Conf. on eScience. Indianapolis (In), December 2008, pp. 233-246.

[12] D. Crawl and I. Altintas, "A Provenance-Based Fault Tolerance Mechanism for Scientific Workflows", Proc. 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. Ramakrishna, et al., "VGrADS: Enabling e-Science workflows on grids and clouds with fault tolerance", Proc. ACM/IEEE Intl. Conf. High Performance Computing, Networking, Storage and Analysis (SC09), Portland (Or.), November 2009, pp. 475-483.

[16] M. Baudin, "Introduction to Scilab", Consortium Scilab, January 2010, Also: http://wiki.scilab.org/ Retreived: January, 2012.

[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 Retreived: January, 2012.

[19] J. Dongarra, P. Beckman, et al., "The International Exascale Software Roadmap", vol. 25, n.1, 2011, International Journal of High Performance Computer Applications, ISSN 1094-3420, pp, 89-96, Available at: http://www.exascale.org/ Retreived: January, 2012.

[20] R. Gupta, P. Beckman, et al., "CIFTS: a Coordinated Infrastructure for Fault-Tolerant Systems", Proc. 38th Intl. Conf. Parallel Processing Systems, Vienna (Au), September 2009, pp. 289-296.

[21] The Workflow Management Coalition. http://www.wfmc.org Retreived: January, 2012.

[22] D. Abramson, B. Bethwaite, et al., "Embedding Optimization in Computational Science Workflows", Journal of Computational Science 1 (2010), Pp 41-47, Elsevier, pp. 89-95.

[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. 179-186.

[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. 231-240.

[25] http://www.yawlfoundation.org/software/documentation. The YAWL foundation, 2010, Retreived: January, 2012.

[26] Y.Simmha, R. Barga, C. van Ingen, E. Lazowska, and A. Szalay, "Building the Trident Scientific Workflow Workbench for Data Management in the Cloud", Proc. 3rd Intl. Conf. on Advanced Engineering Computing and Applications in Science (ADVCOMP2009), Sliema (Malta), October 2009, pp. 179-186.

[27] Object Management Group / Business Process Management Initiative, BPMN Specifications, http://www.bpmn.org, Retreived: January, 2012.

[28] Emmerich W., B. Butchart, and L. Chen, "Grid Service Orchestration using the Business Process Execution Language (BPEL)", Journal of Grid Computing, vol. 3, pp 283-304, Springer.2006, pp. 257-262.

[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. 135-140.

[30] B. Ludäscher, M. Weske, T. McPhillips, and S. Bowers, "Scientific Workflows: Business as usual ?" Proc. BPM 2009, LNCS 5701, Springer, 2009, pp. 345-352.

[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. 4th Workshop on Workflows in Support of Large-Scale Science, WORKS 2009, Portland (Or), ACM 2009, pp. 123-128.

[32] Yildiz U., Guabtni A. and Ngu A.H., "Towards Scientific Workflow Patterns", Proc. 4th Workshop on Workflows in Support of Large-Scale Science, WORKS 2009, Portland (Or), ACM 2009, pp. 247-254.

[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.

[34] Duvigneau R., Kloczko T., and Praveen C.., "A three-level parallelization strategy for robust design in aerodynamics", Proc. 20th Intl. Conf. on Parallel Computational Fluid Dynamics, May 2008, Lyon (F), pp. 379-384.

[35] E.C. Joseph, et al., "A Strategic Agenda for European Leadership in Supercomputing: HPC 2020", IDC Final Report of the HPC Study for the DG Information Society of the EC, July 2010, Available at: http://www.hpcuserforum.com/EU/ Retreived: January, 2012.

[36] P.E. Gill, Murray W. and Wright M.H., Practical Optimization, Elsevier Academic Press, 2004.

[37] Gmsh. https://geuz.org/gmsh/ Retreived: January, 2012.

[38] Metis. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview Retreived: January, 2012.

[39] Num3sis. http://num3sis.inria.fr/blog/ Retreived: January, 2012.

[40] OPALE project at INRIA. http://www-opale.inrialpes.fr and http://wiki.inria.fr/opale Retreived: January, 2012.

[41] L. Trifan and T. Nguyên, "A Dynamic Workflow Simulation Platform", Proc. 2011 Intl. Conf. on High-Performance Computing & Simulation, Istanbul (TK), July 2011, pp. 115-122.

[42] Plankensteiner K., Prodan R., and Fahringer T., "A New Fault-Tolerant Heuristic for Scientific Workflows in Highly Distributed Environments based on Resubmission impact", Proc. 5th IEEE Intl. Conf. on e-Science, Oxford (UK), December 2009, pp 313-320.

[43] Z. Lan and Y. Li, "Adaptive Fault Management of Parallel Applications for High-Performance Computing", IEEE Trans. On Computers, vol. 57, no. 12, December 2008, pp. 337-344.

[44] S. Ostermann, et al., "Extending Grids with Cloud Resource Management for Scientific Computing", Proc. 10th IEEE/ACM Intl. Conf. on Grid Computing, 2009, pp. 457-462.

[45] E. Sindrilaru, A. Costan, and V. Cristea,. "Fault-Tolerance and Recovery in Grid Workflow Mangement Systems", Proc. 4th Intl. Conf. on Complex, Intelligent and Software Intensive Systems, Krakow (PL), February 2010, pp. 85-92.

[46] S. Hwang and C. Kesselman, "Grid Workflow: A Flexible Failure Handling Framework for the Grid", Proc. 12th IEEE Intl. Symp. on High Performance Distributed Computing, Seattle (USA), 2003, pp. 235-242.

[47] The Grid Workflow Forum: http://www.gridworkflow.org/snips/gridworkflow/space/start Retreived: January, 2012.

[48] Bautista-Gomez L., et al., "FTI: high-performance Fault Tolerance Interface for hybrid systems", Proc. ACM/IEEE Intl. Conf. for High Performance Computing, Networking, Storage and Analysis (SC11), pp. 239-248, Seattle (Wa.)., November 2011.

[49] Bogdan N. and Cappello F., "BlobCR: Efficient Checkpoint-Retart for HPC Applications on IaaS Clouds using Virtual Disk Image Snapshots", Proc. ACM/IEEE Intl. Conf. High Performance Computing, Networking, Storage and Analysis (SC11), pp. 145-156, Seattle (Wa.)., November 2011.

[50] Moody A., G.Bronevetsky, K. Mohror, B. de Supinski. "Design, Modeling and Evaluation of a Scalable Multi-level checkpointing System", Proc. ACM/IEEE Intl. Conf. for High Performance Computing, Networking, Storage and Analysis (SC10), pp. 73-86. New Orleans (La.), November 2010.

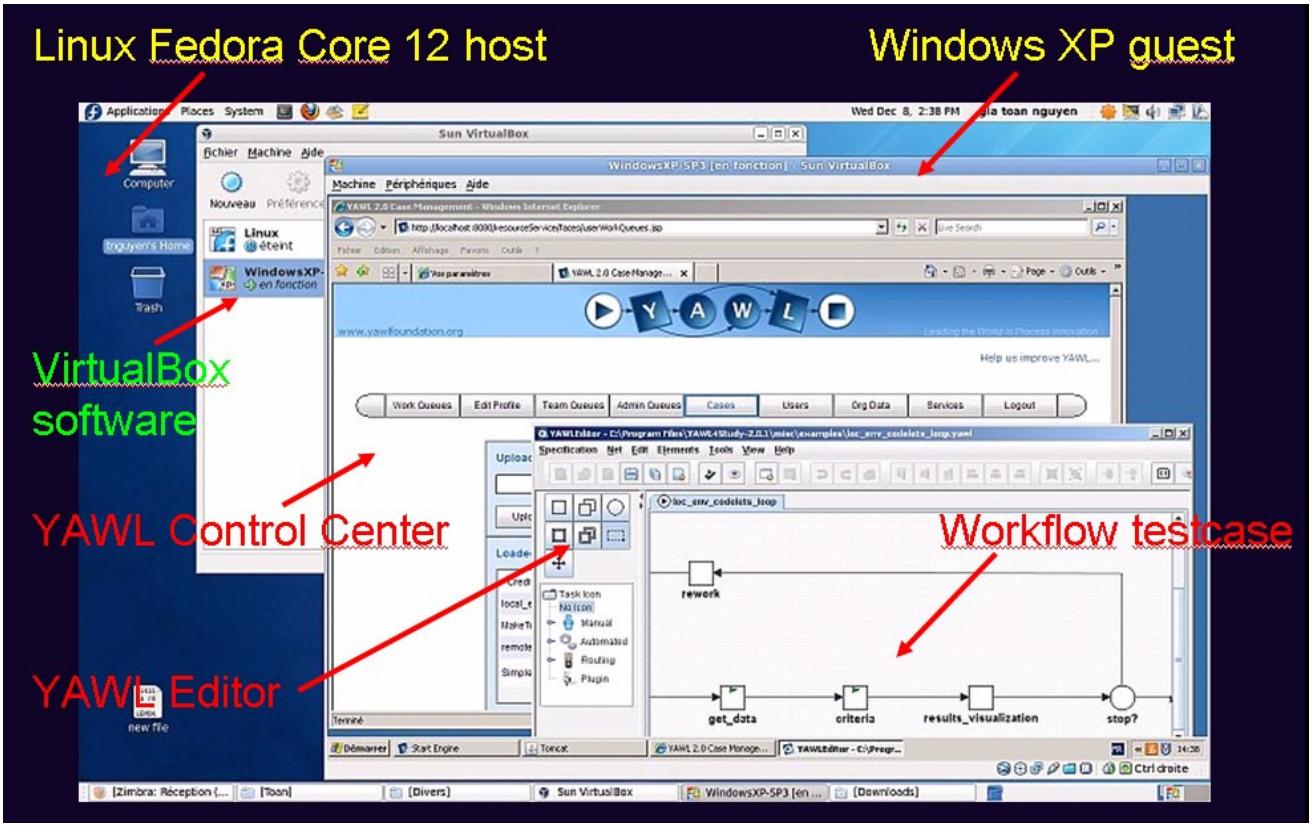


Figure 10. The YAWL testcase and workflow editor deployed on a virtual machine: Linux Fedora Core 12 host running VirtualBox and Windows XP guest.

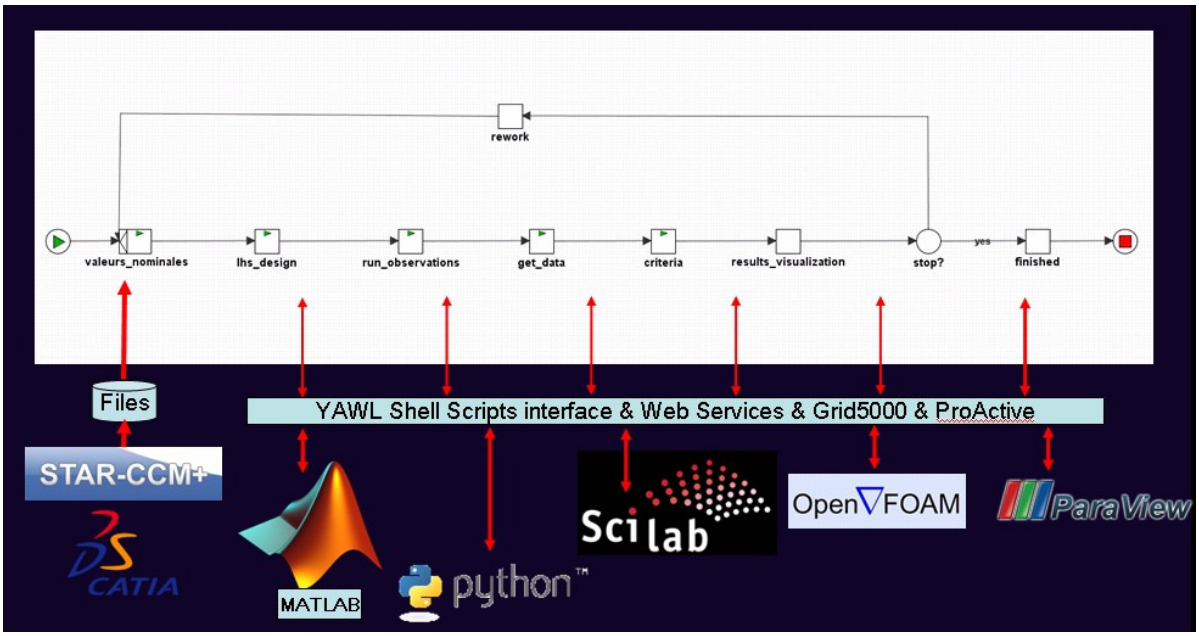


Figure 11. Parameter sweeping and YAWL interface to remote simulation codes.

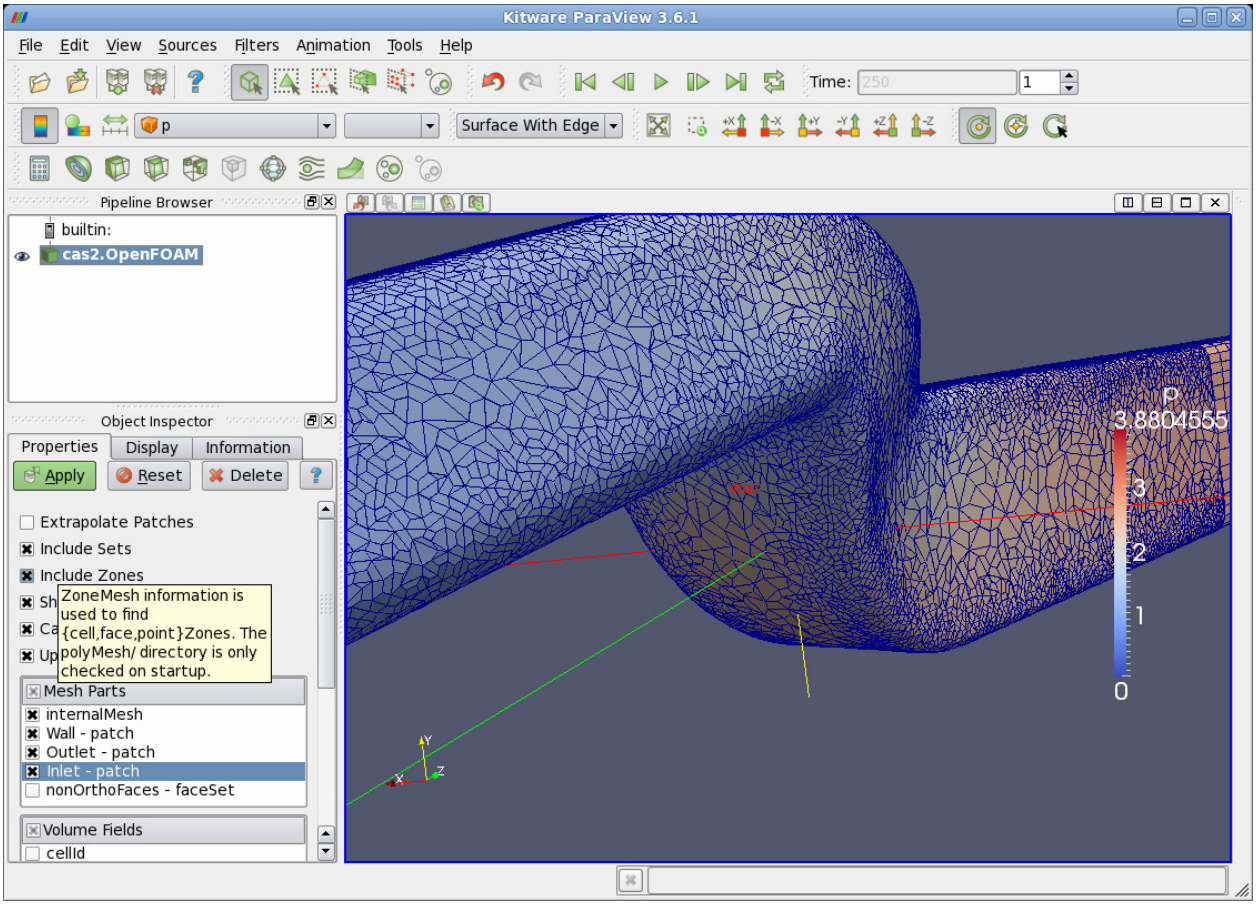


Figure 12. The 3D testcase visualization (ParaView screenshot).

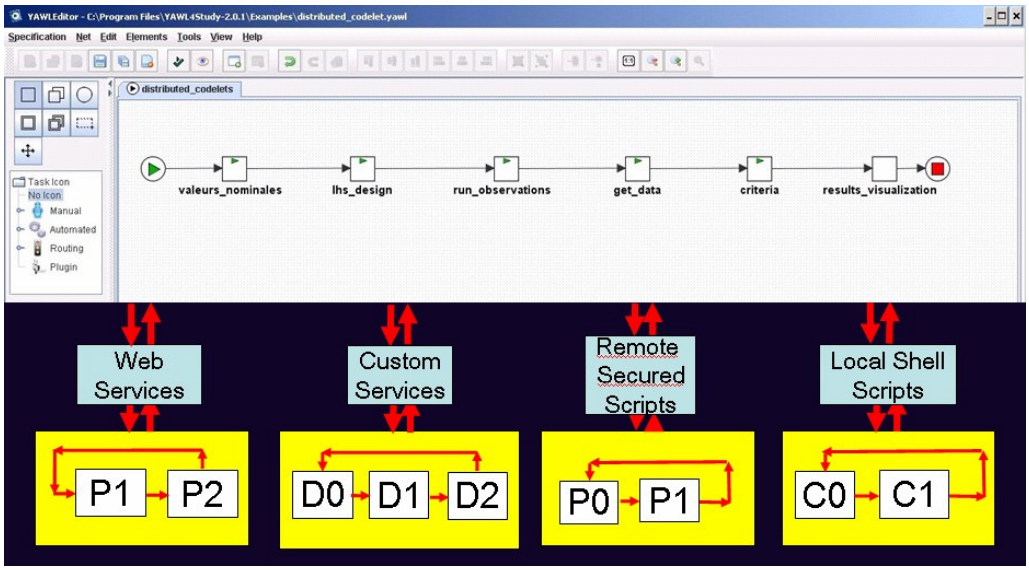


Figure 16. The platform interfaces to local and distributed codes.

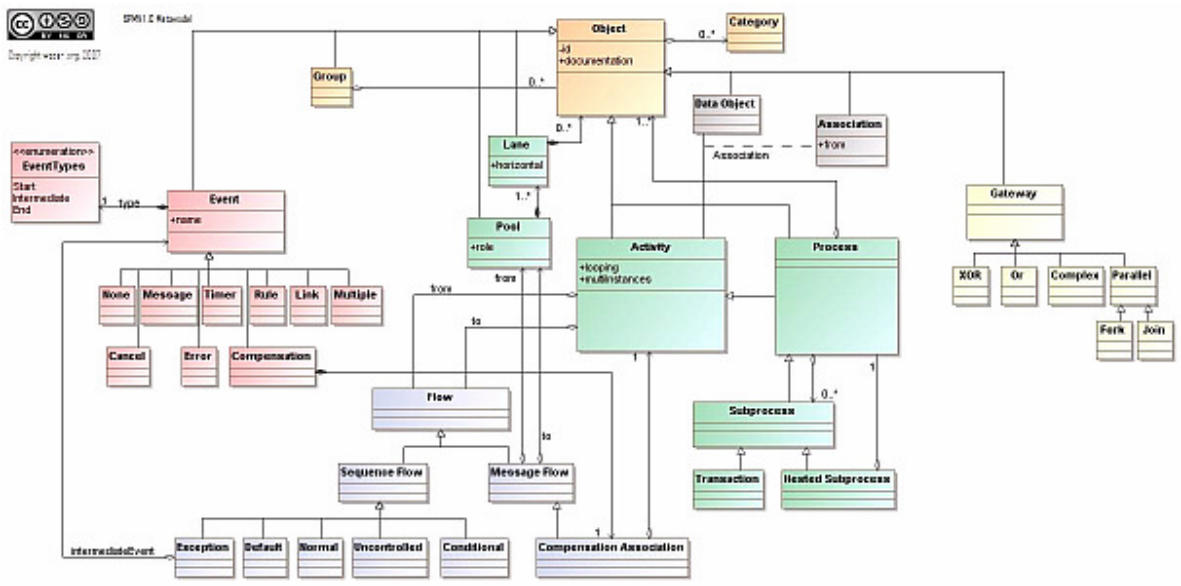


Figure 17. BPMN metamodel - © 2007 OMG.

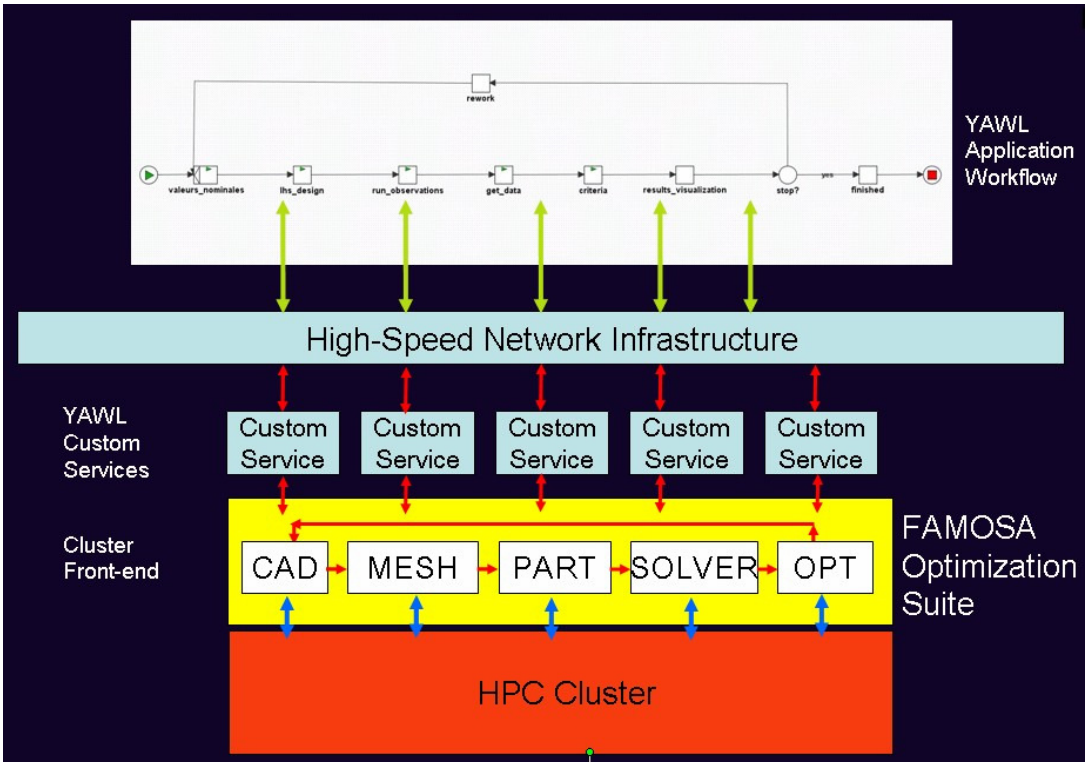


Figure 18. A distributed optimization experiment.

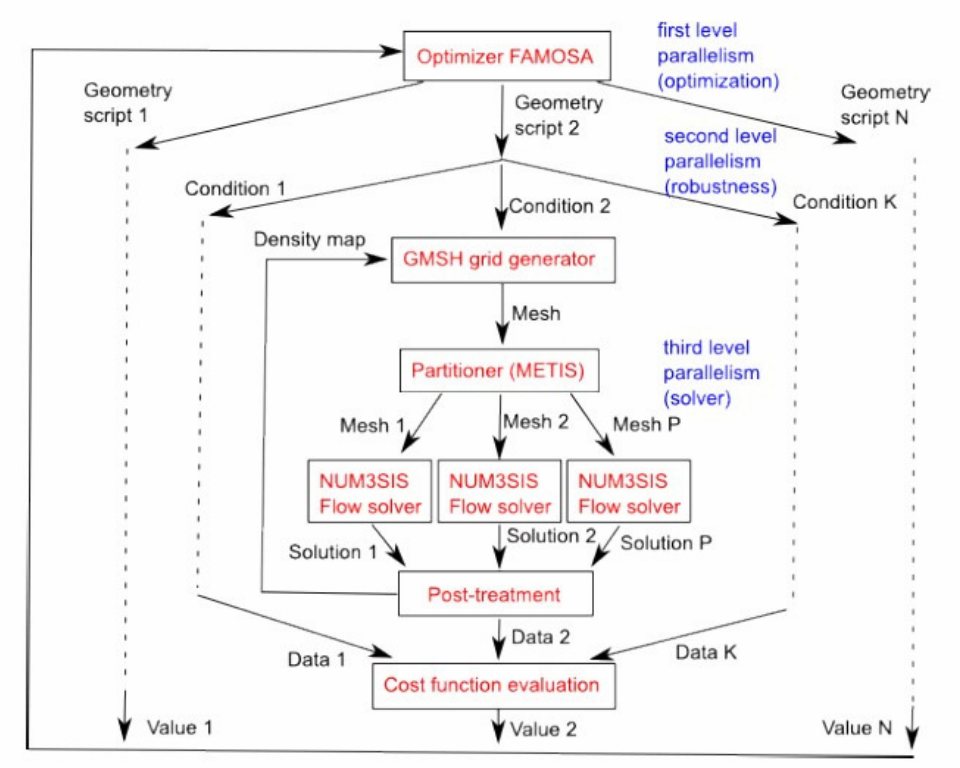


Figure 19. The FAMOSA optimization suite - © 2010 Régis Duvigneau – Project OPALE.

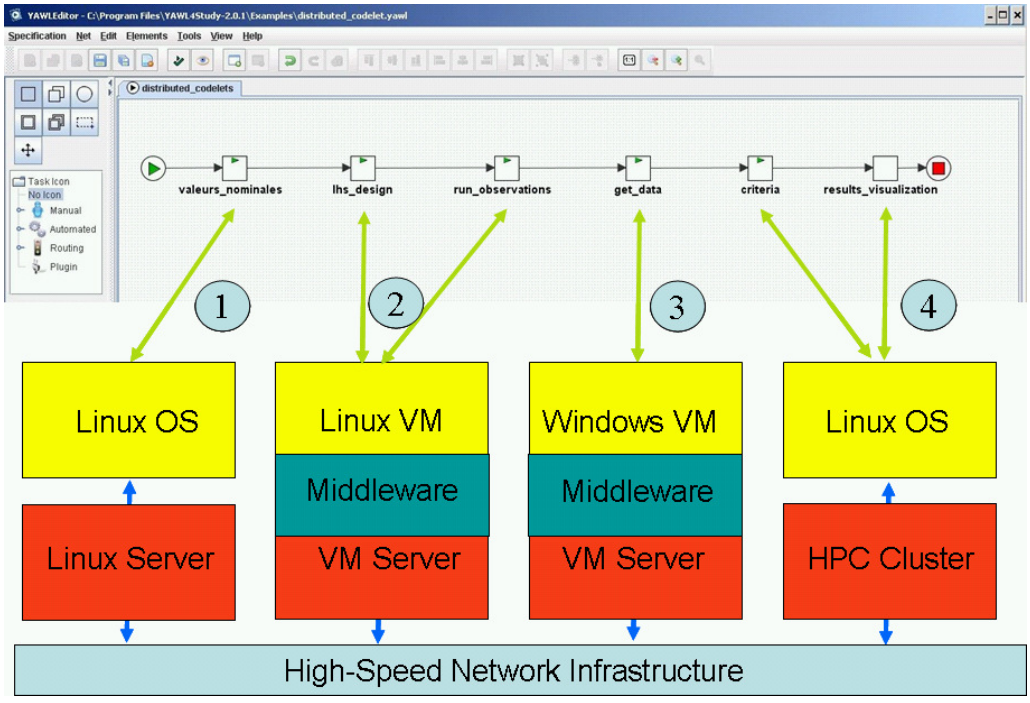


Figure 20. The distributed simulation platform.

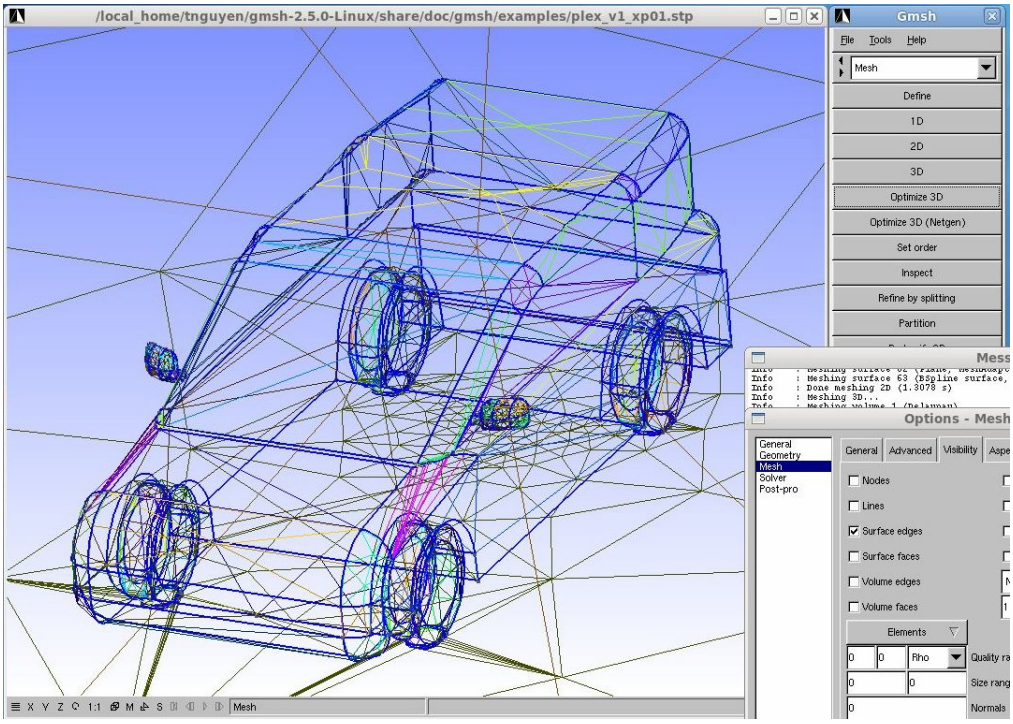


Figure 21. Mesh for vehicle aerodynamics simulation (Gmsh screenshot).

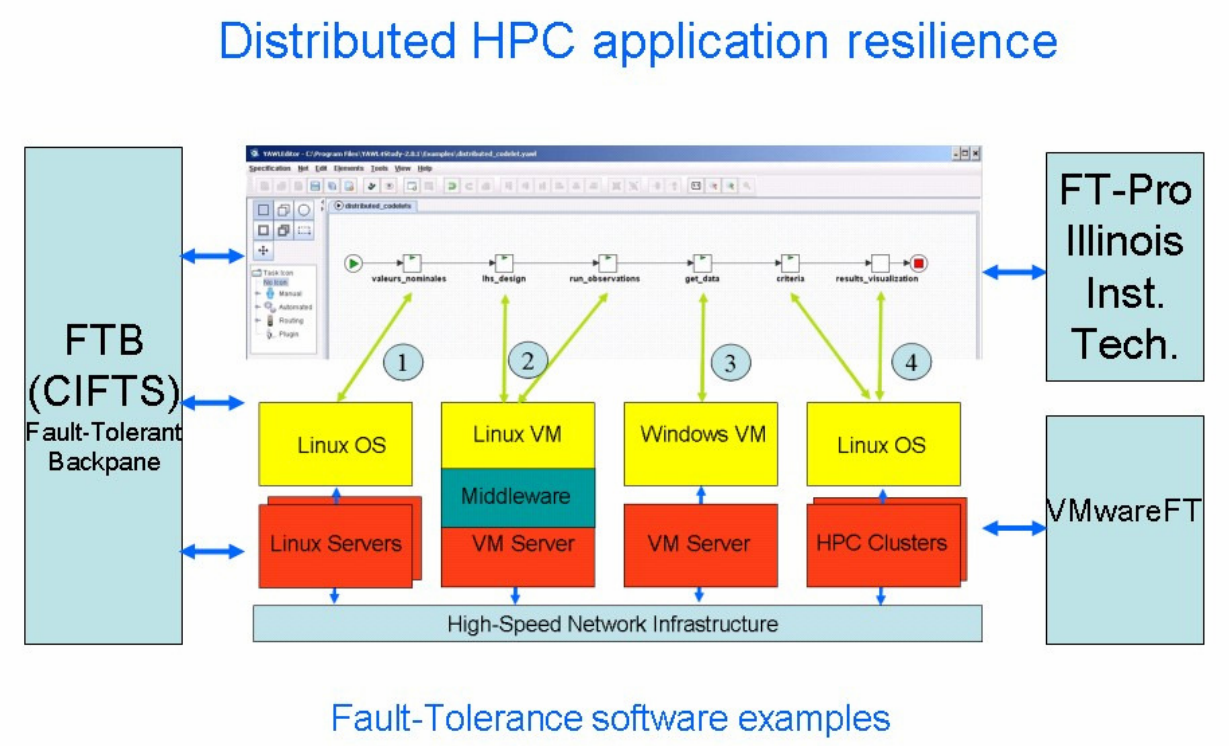


Figure 22. Examples of resilience and fault-tolerance software.

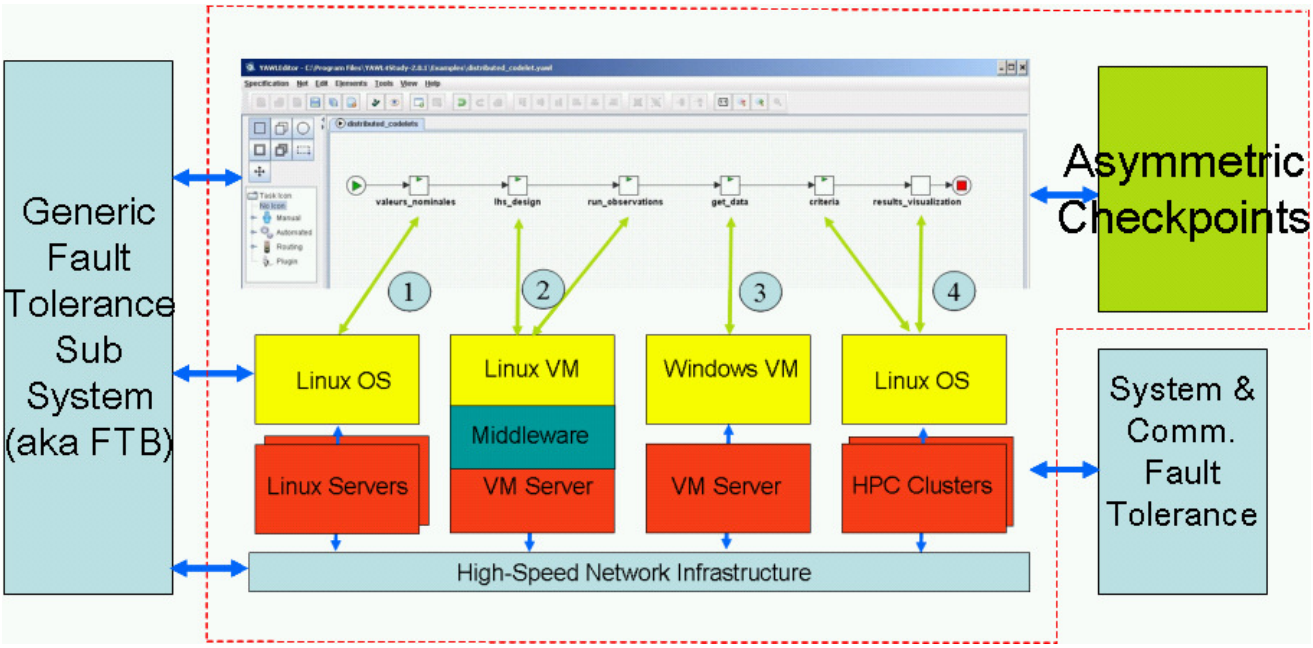


Figure 23. Asymmetric checkpoints software for application resilience.

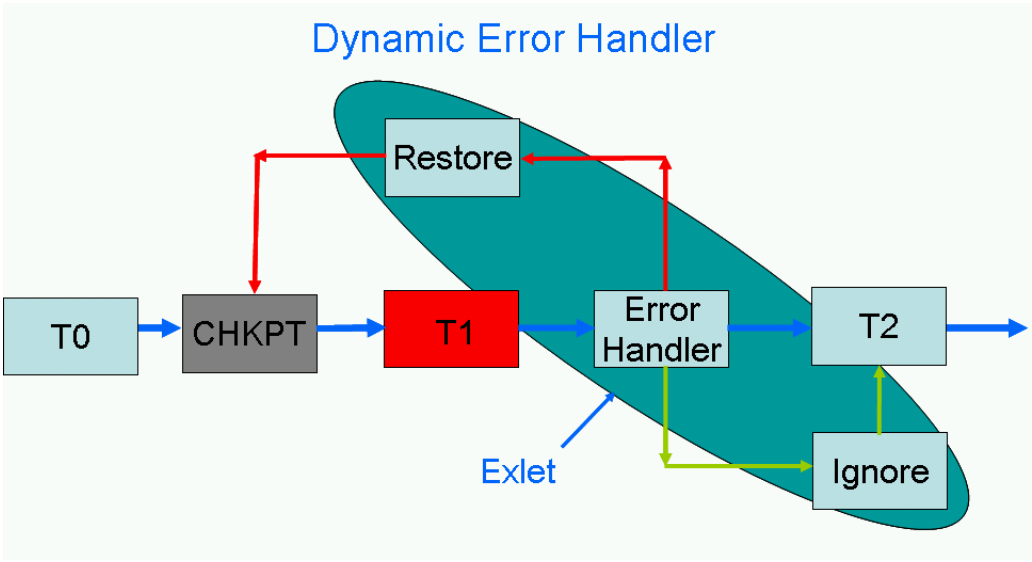


Figure 24. Error handler for task T1 error: YAWL exlet.

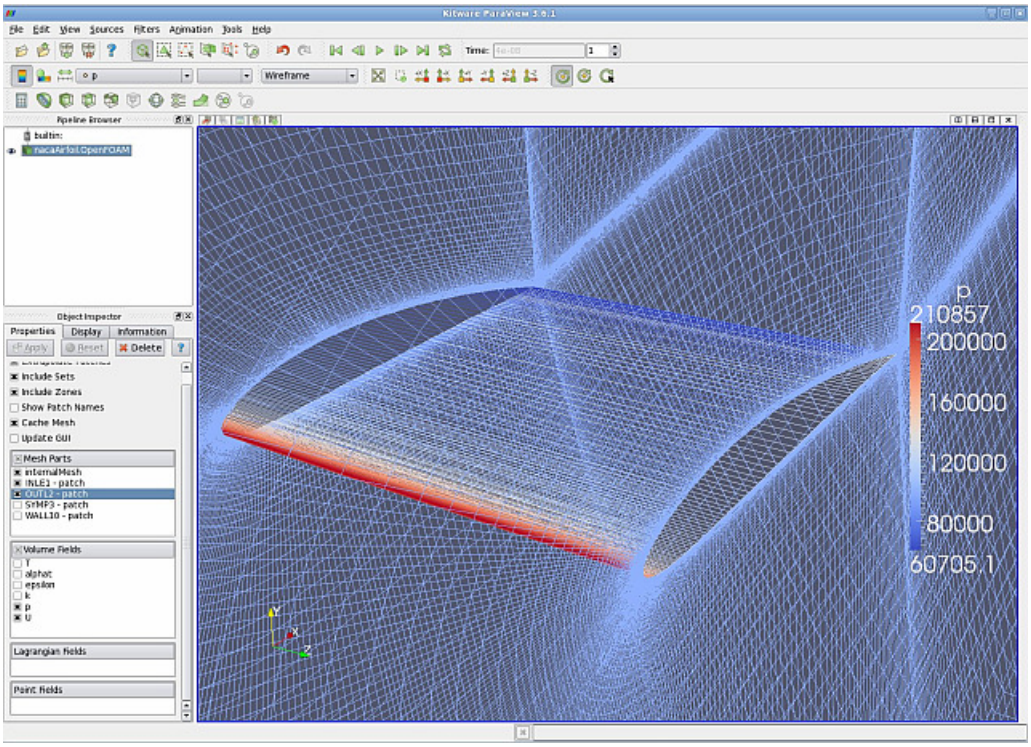


Figure 25. Pressure on a NACA airfoil (OpenFOAM testcase).

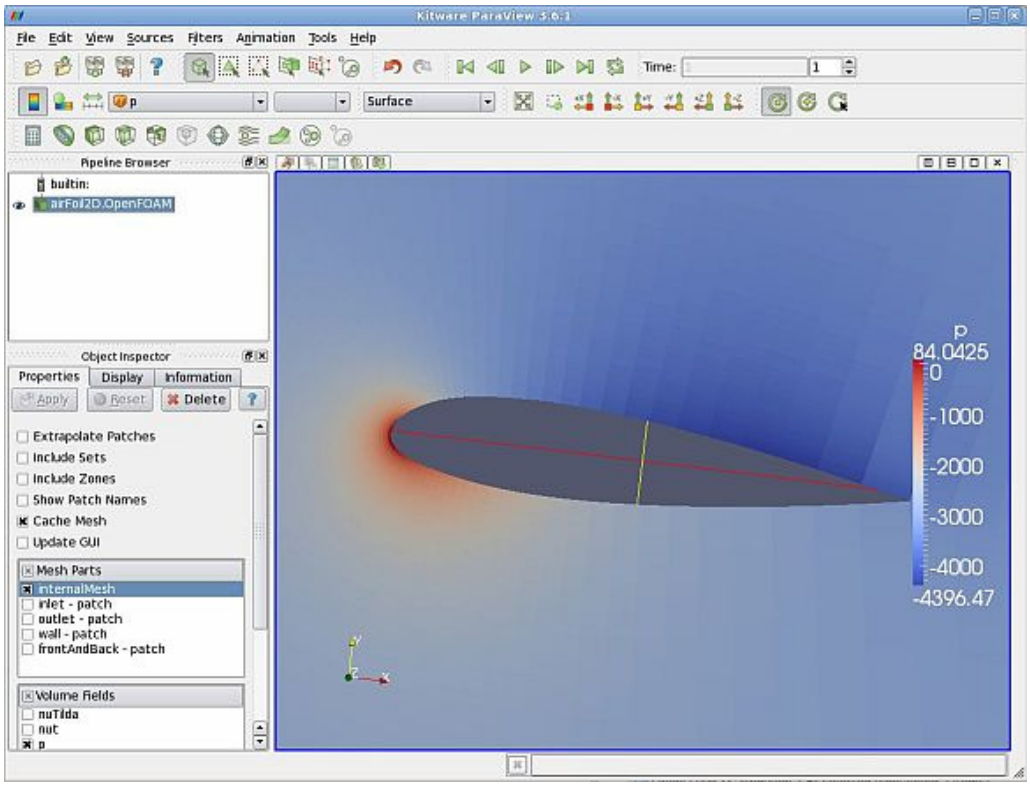


Figure 26. Pressure over a 2D airfoil (OpenFOAM testcase).