Resilient Workflows for High-Performance Simulation Platforms

Toan Nguyen, Laurentiu Trifan, Jean-Antoine Désidéri

Resilient Workflows for High-Performance Simulation Platforms

Toàn Nguyên, Laurentiu Trifan and Jean-Antoine Désidéri
INRIA, 655, Av. de l'Europe, Montbonnot, FR-38334 Saint-Ismier (France)
Toan.Nguyen@inrialpes.fr

ABSTRACT

Workflows systems are considered here to support large-scale multiphysics simulations. Because the use of large distributed and parallel multi-core infrastructures is prone to software and hardware failures, the paper addresses the need for error recovery procedures. A new mechanism based on asymmetric checkpointing is presented. A rule-based implementation for a distributed workflow platform is detailed.

KEYWORDS: Fault-tolerant computing; Large-scale scientific computing; parallelization of simulation; workflow systems.

1. INTRODUCTION

During the last decade, the e-science sector has shown a growing interest in workflows [1, 4, 14, 16]. It has extensively used a dataflow approach for the processing of large numeric data sets [5, 6, 7, 17].

Large-scale multiphysics applications, e.g., aircraft flight dynamics simulation that takes into account aerodynamics and structural loads, are considered today fundamental by aircraft manufacturers in order to gain leading position on highly competitive innovative markets world-wide. The same goes for mobile phones manufacturers.

Not only are organizational problems put forward, because of the risk-sharing partnerships that are often implemented, but technological and scientific challenges are addressed because verification and validation of numeric models are necessary in order for virtual prototypes to allow drastic reduction in time-to-market design [2, 8].

Multiphysics approaches are considered here to better combine and synchronize the intricate relationships between the various disciplines that contribute to the integration of complex new products, e.g., acoustics, electromagnetics and fluid dynamics in aircraft design [7, 5].

High-performance computing also opens new perspectives for complex products definition, design and tuning to market needs [9]. However, high-performance computing platforms also raise new challenges to computer scientists to fulfill the design bureaus requirements, e.g., the management of petascale volumes of data, the management of distributed teams collaborating on large and complex virtual prototypes, using various remote computerized tools and databases, etc [1, 11].

This paper focuses on the design of distributed workflows systems that are used to define, deploy, configure, execute and monitor complex simulation and optimization applications. It emphasizes the need for resilient workflows. It does not consider hardware and system-level fault-tolerance. In a way similar to [7], it focuses on a generic approach to handle application-level failures, namely the implementation of resilient workflows. In that sense, it copes with the byzantine application processes as described elsewhere in the literature [6].

Section 2 deals with dynamic workflows. Section 3 presents resiliency for workflows, including fault-tolerance, resiliency, and checkpointing issues: two approaches are described, namely bracketing checkpoints and asymmetric cascading checkpoints. Section 4 deals with implementation issues in connection with an ongoing project on distributed multidiscipline optimization platforms. Section 5 is a conclusion.

2. RESILIENT WORKFLOWS

Multiphysics design includes several disciplines and various tools that pertain to each particular expertise

involved. This includes CAD tools, meshers, solvers, analyzers and optimizers, which in turn are used to modify the meshes in iterative and incrementally optimized design processes (Figure 1).

Pause, resume, abort facilities are required in distributed workflow systems to update input parameters for the simulation and optimization processes.

This calls for dynamic logging mechanisms, interleaved checkpoints management, distributed pause, resume and abort mechanisms. They can be used also as building blocks to support resiliency.

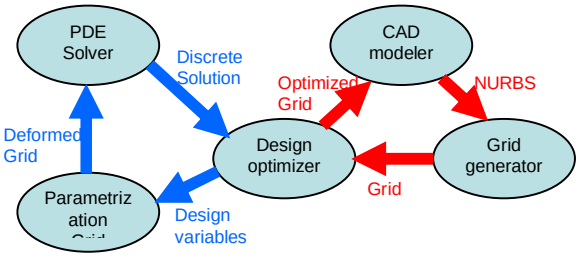

Figure 1. Optimization Workflow.

Past research in distributed systems tells us that distributed recovery algorithms are implemented using partial order among checkpoints.

This guarantees that the executing application codes can be paused and resumed after dynamic parameter updates by the users. It also guarantees that the executing applications can be restored after system or application failures. The whole workflow systems, including the running applications, are then qualified here *resilient workflows*. This departs from fault-tolerance, which is restricted here only to hardware, system and communication failures. In this case, it concerns *fault-tolerant workflows*.

In case of erratic application behavior, it is also clear that the users can invoke these services to abort them as well as pause and later update the execution parameters to restart the simulation processes.

The infrastructure required to implement these services can benefit from the appropriate functionalities developed in existing grid middleware, e.g., Globus, UNICORE, gLite [12, 13]. Also, high-performance visualization tools like parallel display walls can be interfaced with the workflow systems, e.g., CUDA programming tools on GPU-clusters, to compare various design alternatives in real-time.

2.1. Fault-Tolerant Workflows

Because distributed systems are potentially faced with unexpected hardware and software failures, adequate mechanisms have been devised to handle recovery of running systems, software and applications.

Checkpoint and restart mechanisms are usually implemented using the local ordering of the running processes. This implies that the safe execution of all the running processes is not guaranteed, i.e., there is no way a randomly aborted distributed process can be restored in a consistent state and resume correctly. The solution would be to use a global synchronization and clock, which is practically unfeasible and very constraining.

Design, simulation and optimization applications bear specificities that require less stringent mechanisms than transaction systems. Design is a stepwise process that does not require global synchronizing, except when, and only when, dynamic update propagation is required. This can be executed during limited time periods and does not impair the usual stepwise approach.

The same goes for simulation and optimization, where long duration processes are executed, which can invoke many composite components. These components may be invoked by sub-workflows. They bear a similar nature: global synchronization is not required, only synchronization for composite sub-workflows with their running components. Even so, asynchronous executions using pipelining of intermediate results can be devised.

For example, the wing optimization workflow depicted in Figure 2 can use the following checkpoints:

- C0 and C1 to save the paraOMD2meters and the optimization results
- C2 and C3 to save the individual solutions (alternatively, C'3 can save the results)
- C4 and C5 to save the individual solutions geometry variants (the forms)
- C6 and C7 to save the various flight regimes results (C'7 if the database is saved)
- C8 and C9 to save the results of the various solvers executions (and C'9 to save the database)

They are called here *bracketing checkpoints* (Section 3.3).

Should some random hardware and software failure occur, it is easy to see that each optimized solution (called here "Individual") computed so far is saved, corresponding to every geometry (called here "Form"), every flight "Regime" and every "Solver" computation is

saved. This minimizes the process of resuming the optimization workflow when aborted due to some external cause. This is an implementation of fault-tolerance.

For example, the checkpoint C6 supports the resuming of the composite and parallel sub-workflow "Regimes". The latter can be restarted entirely or partially if some of its component "Solvers" resumed correctly. Their results are checkpointed by C9 and alternatively C'9 if they are stored in the database DB_Perf (Figure 2).

2.2. Resiliency

Resiliency differs from fault-tolerance because it is related to the ability of the applications to survive to unpredictable behavior.

In contrast with fault-tolerant workflows which can survive hardware and system failures, using ad-hoc bracketing by checkpointing mechanisms (Section 3.3), resilient workflows need to be aware of the application structure to implement automated survival procedures. These procedures can use the bracketing of sub-workflows also, but in addition, they need specific logging of the workflow component operations and parameters to restore incrementally previous states and resume partially their operations (Figure 3).

2.3. Bracketing Checkpoints

Thus, checkpoints must be inserted in the workflow composite hierarchy. They can bracket critical parts of the hierarchy, e.g., the most demanding CPU components (unsteady flow calculations over a 3D wing model, for example) and the following optimization components which might be less CPU demanding, but are fundamental to the application because they allow for the comparison of various optimized solutions. This scheme is called *bracketing checkpoints.*

Further, parallel branches of the workflow that failed need later to be re-synchronized with the branches that resumed correctly. This requires that the results of the successful branches are stored for further processing with the failed branches results, if they resume correctly later. Otherwise, these results are discarded if the failed branches never succeed. Because there is no awareness of the successful branches on the possible failures of parallel branches in the workflow, time-out and synchronization signals must be exchanged on a regular basis to notify each branch of the current state of the others: alive or not responding.

2.4. Resiliency Procedure

An iterative process is implemented that chooses a particular checkpoint and executes several steps forward. If the application does again behave erratically, it is supposed to be stopped by the user. The checkpoints are then chosen further backward in time in the workflow execution and the application is then again partially resumed with updated parameters or pre-specified user operations (Figure 3). This resiliency mechanism iterates until the workflow resumes correctly or is aborted.

For example, if the "Regimes" component workflow fails for some unpredictable reason, the resiliency process will restore the workflow state at checkpoint C6 (Figure 3). This means that all solvers calculations for the current individual solution will be restarted. Note that some particular solver computations that resumed correctly so far for the current individual solutions are already saved at checkpoint C9.

Should this process fail again for some reason, the resiliency mechanism will step backwards to checkpoint C4. This means that the whole geometry calculations for the current individual solution will be restarted. Note that the computations already finished for other individuals are not affected and have been saved at checkpoints C9, C7, C5 or C3 if no synchronization barriers have been defined.

Should again the whole "Regime" component workflow fail, the resiliency mechanism will step backwards again to checkpoint C2, which means that the whole geometry, regimes and solvers computations will be restarted for the current individual solution being processed.

2.5. Asymmetric Checkpoints

Because bracketing checkpoints might also incur a large overhead when used in composite workflows, their occurrence must be fine-tuned to each particular application workflow.

For example, the checkpoints C0, C2, C4 and C6 which store the state and data relevant to the component workflows "Optimize", "Individuals", "Forms" and "Regimes" in Figure 3 are redundant with the checkpoints C1, C3, C5 and C7.

Indeed, should a failure occur in a component workflow, e.g. “Regime”, the preceding checkpoint C6 will be used to restore the application in a safe state. It is therefore redundant to insert the checkpoint C8, except if the “Configuration” and “Read_DB” tasks are critical.

An appropriate placement mechanism must therefore be implemented to optimize the recovery procedure and minimize the checkpoints overhead for running applications.

An asymmetric scheme has been designed to handle this problem. Opening checkpoints, e.g., checkpoints inserted prior to critical tasks (e.g., C2, C4, C6) are paired with closing checkpoints of component workflows that are not immediate children of the parent component workflow. This avoids redundant checkpoints.

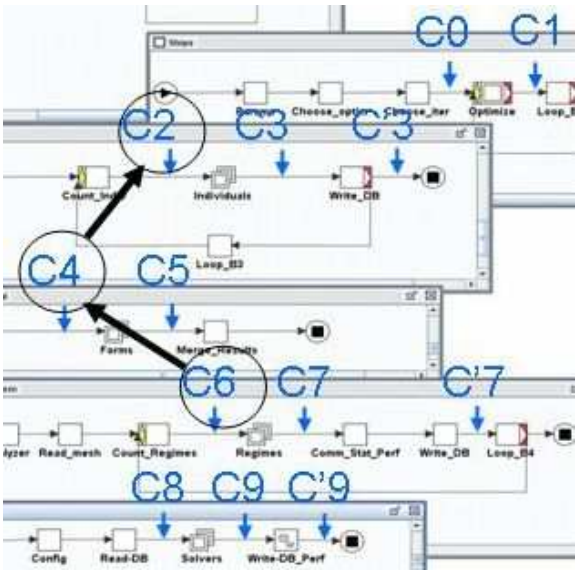

Figure 3. Resilient Workflow: Iterative Recovery.

This scheme is called *asymmetric cascading checkpoints*. It is particularly useful when multiple instances of component workflows are defined, e.g., in the example above, the “Regime” task is defined as a multiple instance composite component. This means that there may exist several instance of the “Regime” component workflow executing at the same time in parallel for a specific Form instance of workflow (and for a specific “Individual” solution). In the example above, the tasks “Forms”, Individuals are also multiple instance composite components.

2.6. Heuristic Rules

We assume in the following that “join” operations are those that require several input datastreams to execute. Similarly, we assume that “fork” operations are those that output their results on several datastreams. They model generic tasks that execute application codes. We also consider “remote” and “local” operations. We do not distinguish between parallel and sequential implementations of the operations.

Further, we consider in the following that the “specified” operations are those operations or workflow tasks that are marked by the application designers or the users as requiring a specific treatment in the following heuristic procedure.

The specific characterization of the marked tasks is implemented by raising an exception that invokes a specific treatment that departs from the standard heuristic rules. An example of such exception is the backup of a particular intermediate result after processing by a large CPU intensive task or the back up of the result of a task producing petascale volumes of data. Workflow management systems usually provide powerful exception handling functionalities that can be used to implement this kind of “specified” operations management, e.g., YAWL [14].

This enables the designers and users to adapt the execution of the workflow depending on their specific knowledge and expertise. This is a prerequisite for the effective implementation of the workflows based on previous runs and casestudies involving petaflops and petabytes of data. Some automated learning procedure could eventually be designed to support this kind of feedback.

The recovery procedure implements a heuristic approach based on the following rules:
- 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

To improve performance, these rules can be tuned by the application designers to fit their specific requirements. This includes specified operations that are deemed CPU intensive and data transfer intensive.
They can also be altered or ignored by load-balancing strategies if appropriately authorized by the designers

and users, and if global and local policies make this mandatory, e.g., preemptive local strategies.

Based on these rules, the example illustrates the asymmetric cascading checkpoints on an unfolded workflow (Figure 4). Two remote execution sites are considered: Site a (white colored tasks) and Site b (red colored tasks).

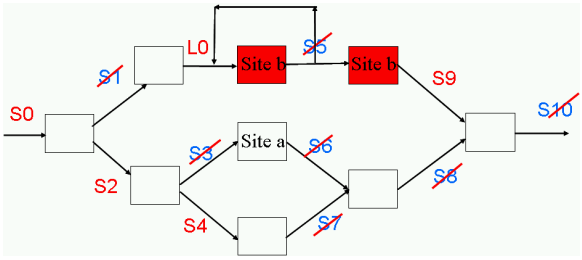

Figure 4. Asymmetric Checkpoints.

When it is not modified and tuned by the designers, the result of the asymmetric cascading checkpoints procedure results in seven unnecessary checkpoints which are deleted, thus leaving five remaining checkpoints: S0, S2, S4, S9 and L0.

3. IMPLEMENTATION

The approach implemented here uses the YAWL workflow management system. It is one of the few workflow systems to be defined with a sound formal semantics [18]. It is designed to combine grid and distributed computing through a middleware with scientific computing using a mathematical problem solving environment. It thus provides an e-Science infrastructure as a high-performance platform for large-scale distributed data and CPU intensive applications. Validation of the platform is through industrial testcases concerning car aerodynamics and engine valves and pipes optimization.

The approach wraps the existing applications codes, e.g., optimizers and solvers, with Web services that are invoked for remote execution when required. When software components are local, they are invoked through shell scripts that in turn trigger the appropriate software. This script invocations are natively implemented in the YAWL workflow system for automated execution of application tasks. Parameters passing to the software invoked are defined by standard YAWL protocols. Results are similarly transferred back to YAWL by the application software through scripts callbacks for later use by other tasks in the workflow. Dynamic interactions through user-definable forms are also standard in YAWL, which support parameter initializations, dynamic interactions with the executing applications and ad-hoc execution features, e.g., dynamic introduction of new exceptions, flow control, etc .

This is extended to the checkpointing and resiliency procedures which are defined by standard YAWL workflow tasks. They are inserted appropriately in the workflow definition, in compliance with the specific scheme adopted, i.e., bracketing scheme or asymmetric cascading scheme (Section 2).

The invocations of the various tasks in the workflow by other tasks are specific YAWL invocations through shell scripts if the tasks are local. They are invoked by Web Services if the tasks are remote. Data and task parameters and descriptions are uniformly exchanged as XML schemas.

A platform supporting these features is developed for the OMD2 project [21] by a consortium that includes twelve academic and industry partners, including a major international car manufacturer leading the project. OMD2 is an acronym for Distributed Multi-Discipline Optimization.

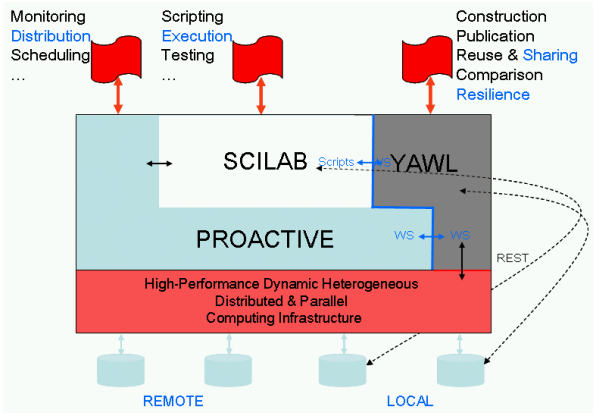

Figure 5. OMD2 Distributed Optimization Platform.

The goal is to develop a high-performance distributed environment for simulation and multidiscipline optimization in complex design projects. The distributed platform uses the ProActive middleware for resource allocation and scheduling of tasks [19]. The tasks invoke software codes that collaborate and include Matlab, Python and Scilab scripts [20], the OpenFOAM software for solver components and mesh generation, the

ParaView software for data visualization and manipulation, optimization software developed by the project partners, as well as commercial CAD tools, e.g., CATIA v5 and STAR-CCM+ (Figure 6).
YAWL is used for defining incrementally composite workflows, as well as the sharing and reuse of the various software that form the applications. These can interact with the users through sophisticated exception handling mechanisms and interact with each other using Web services (Figure 5). This is also used for implementing the resilience and fault-tolerance features described in the previous sections (Section 2).
Because the workflow engine supports natively dynamic interactions with software through Web Services and shell scripts, it communicates with the ProActive engine using specific services for distributed resource allocation and scheduling. Similarly, interactions with OpenFOAM, ParaView, Python, the Matlab and Scilab numeric computation software are based on shell script invocations for the execution of all local application codes and YAWL users.

4. CONCLUSION

Distributed infrastructures exhibit potential hazards to the executing processes, due to unexpected hardware and software failures. This is endangered by the use of distributed high-performance environments that include very large clusters of multi-processors nodes. This requires fault-tolerant workflows systems. Further, erratic application behavior requires dynamic user interventions, to adapt execution parameters for the executing codes and to run dynamic application re-configurations. This requires resilient workflow systems.

This implies applications roll-back to appropriate checkpoints, and the implementation of survivability procedures, including fault-tolerance to external failures and resiliency to unexpected application behavior.
Asymmetric cascading checkpoints are presented here to effectively support the resiliency procedure. In order to minimize the overhead incurred by the checkpointing and logging of the workflow operations, a heuristic is presented that uses tunable rules to adapt the resiliency procedure to the application requirements and to comply with the computing infrastructures.
Oen issues are currently under investigation: the impact of the rule ordering on the resilience performance in case of application restart, the impact of user defined rules inserted in the default rule set, the impact of application characteristics (CPU and data intensive) on the expected resilience overhead/application performance ratio.

ACKNOWLEDGEMENTS

This work is the result of the authors' contributions to: the AEROCHINA2 project of the EC, supported by the "Transport including Aeronautics" program of the FP7, contract n°ACS7-GA-2008-213599 and the OMD2 project ("Optimisation Multi-Disciplines Distribuée") of the French National Research Agency (ANR), grant n°ANR-08-COSI-007-07, program COSINUS ("Conception et Simulation").

REFERENCES

[1] Y.Simmhan et al. "Building the Trident Scientific Workflow Workbench for Data Management in the Cloud". Proc. 3rd Intl. Conf. on Advanced Engineering Computing and Applications in Science. ADVCOMP'2009. Sliema (Malta). October 2009.

[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.

[3] IEEE TCSC Workflow Management in Scalable Computing Environments.
http://www.swinflow.org/tcsc/wmsce.htm

[4] T. Nguyên, J-A Désidéri "Dynamic Resilient Workflows for Collaborative Design". Proc. Intl. Conference on Cooperative Design, Visualization and Engineering. CDVE2009. Luxemburg. LNCS 5738. Springer. September 2009.

[5] B. Abou El Majd, J.-A.Désidéri, and R. Duvigneau. "Multilevel strategies for parametric shape optimization in aerodynamics". European Journal of Computational Mechanics 17, 1–2 (2008).

[6] D. Mogilevsky et al. "Byzantine anomaly testing in Charm++: providing fault-tolerance and survivability for Charm++ empowered clusters". Proc. 6th IEEE Intl. Symp. On Cluster Computing and Grid Workshops. CCGRIDW'06. Singapore. May 2006.

[7] G. Kandaswamy et al. "Fault-tolerant and recovery of scientific workflows on computational grids". Proc. 8th Intl. Symp. Cluster Computing and the Grid. 2008.

[8] V. Selmin. "Advanced Tools for Multidisciplinary Analysis and Optimisation". Proc. 2nd China-EU Workshop on Multi-Physics and RTD Collaboration in Aeronautics. Harbin (China) April 2009.

[9] Perrier P. "Virtual flight tests". PROMUVAL Seminar. National Technical University of Athens. November 2005.

[10] T. Nguyên. "A perspective on High-Performance Collaborative Platforms for Multidiscipline Simulation and Optimization". Proc. 2nd China-EU Workshop on Multi-Physics and RTD Collaboration in Aeronautics. Harbin (China). April 2009.

[11] Y. Simmhan et al. "GrayWulf: Scalable Softawre Architecture for Data Intensive Computing". Proc 42nd Hawaii Intl. Conf. on System Sciences. January 2009.

[12] Proc. 1st Int'l Workshop on Workflow Systems in Grid Environments (WSGE06). October 2006. Changsha(China).

[13] Proc. 2nd Int'l Workshop on Workflow Management and Application in Grid Environments (WaGe07). August 2007. Xinjiang (China).

[14] W. van der Aalst et al. "Design and implementation of the YAWL sytem". Proc. 16th Intl. Conf. on Advanced Information Systems Engineering. CaiSE'2004. 2004.

[15] M. Adams et al. "YAWL User Manual". Version 2.0f. The YAWL Foundation. September 2009.

[16] Y. Gil et al. "Examining the challenges of Scientific Workflows". IEEE Computer. December 2007.

[17] E. Deelman et al. Proc. NSF Workshop on the Challenges of Scientific Workflows. Arlington (Va.). May 2006.

[18] N. Russel et al.. "Workflow control flow patterns. A revised view". Tech. report. University of Eindhoven (NL). 2006.

[19] F. Baude et al. "Programming, composing, deploying for the grid". In Grid Computing: software environments and tools. J. C Cunha and O. F. Rana (Eds). Springer Verlag. January 2006.

[20] S. Campbell et al. "Modeling and simulation in SciLab/Scios". Springer. ISBN: 978-0-387-27802-5. 2006. Also: http://wiki.scilab.org/

[21] R. Filomeno Coelho, P. Breitkopf et al. MULTIDISCIPLINARY DESIGN OPTIMIZATION IN COMPUTATIONAL MECHANICS. Wiley/ISTE. New-York. 2010.

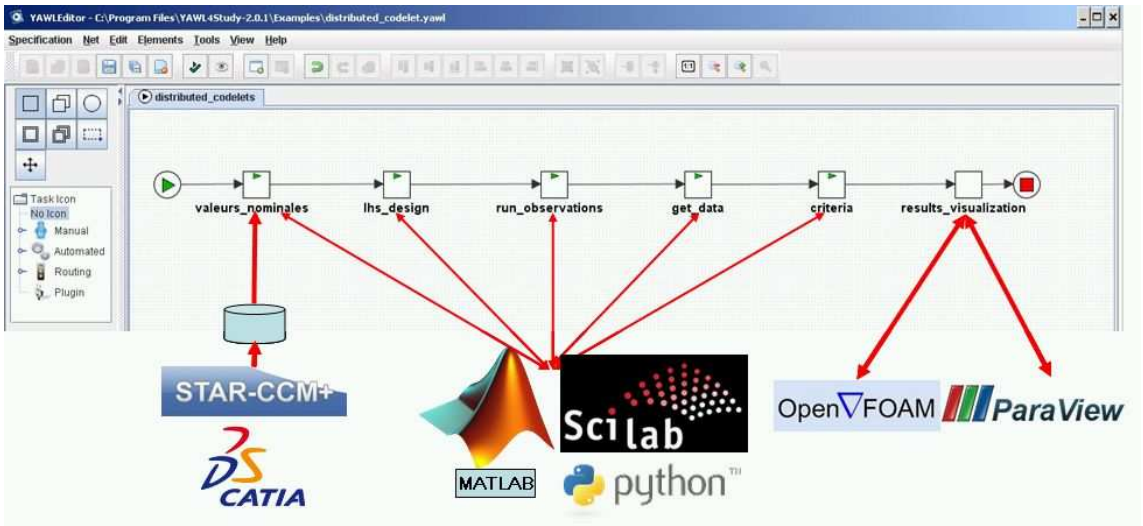

Figure 6. Workflow Interactions for OMD2 Testcase.