VIP ANR-09-COSI-03
Virtual Imaging Platform

D2.1.1: Prototype integration of simulators within VIP

(in alphabetical order)

Hugues Benoit-Cattin	CREATIS	yougz@creatis.insa-lyon.fr
Sorina Camarasu-Pop	CREATIS	pop@creatis.insa-lyon.fr
Patrick Clarysse	CREATIS	clarysse@creatis.insa-lyon.fr
Guillaume Cottez	CREATIS	cottez@creatis.insa-lyon.fr
Denis Friboulet	CREATIS	friboulet@creatis.insa-lyon.fr
Alban Gaignard	MODALIS (I3S)	alban.gaignard@i3s.unice.fr
Bernard Gibaud	IRISA (INRIA Rennes)	Bernard.Gibaud@irisa.fr
Tristan Glatard	CREATIS	glatard@creatis.insa-lyon.fr
Patrick Hugonnard	CEA-Leti	patrick.hugonnard@cea.fr
Carole Lartizien	CREATIS	lartizien@creatis.insa-lyon.fr
Laurent Leduvehat	CREATIS	leduvehat@creatis.insa-lyon.fr
Gabriel Levy	CREATIS	gabriel.levy@insa-lyon.fr
Hervé Liebgott	CREATIS	liebgott@creatis.insa-lyon.fr
Johan Montagnat	MODALIS (I3S)	johan@i3s.unice.fr
Joachim Tabary	CEA-Leti	joachim.tabary@cea.fr
Sébastien Valette	CREATIS	valette@creatis.insa-lyon.fr

Abstract

This document presents the current status of simulator integration in the VIP. It is a deliverable of VIP task 2.1, "syntactic integration". Simulator integration is based on workflow descriptions. Workflows consist of several workflow components that can be pre-processing, core simulation or post-processing workflows. The workflow development framework (language and execution environment) is presented. Then, a core simulation workflow component is described for each project simulator, namely Sindbad (CT), Sorteo (PET), FIELD-II (US) and SIMRI (MR). Performance figures obtained on the European Grid Initiative infrastructure (EGI) are reported. First sketches to object preparation, parameters generation, final parameters assembling and post-processing workflow components are finally discussed.

Contents

1 Introduction

The integration of image simulators in the Virtual Imaging Platform is based on workflows. Workflows have been used for several years both to describe business processes[1] and to execute applications on grid infrastructures [6]. Workflows are directed graphs whose edges represent control or data dependencies between *activities* (executable processes). They are interesting application representations for VIP because they provide a structured description of the processes. Such structure can be exploited by humans and programs to perform a variety of operations, such as:

- automatic parallelization, for execution on a distributed infrastructure (see task T2)
- logging of data provenance and annotation of produced data (task T1)
- automatic manipulation and assembly of workflow components (task T1.2.b)

The core computation of the simulation is only a part of a complete simulation experiment. First, the object model has to be selected and prepared for the simulation. Then, simulation parameters have to be tuned. Finally, various kinds of post-processing can be selected. Several alternates exist for each of these steps and some of them can be reused across various simulations.

The workflow designer developed in T1.2.b will be able to retrieve these workflow components and assemble them appropriately. It will exploit semantic annotations on the workflow components and activities. To prepare the creation of the ontology, the semantic analysis (project task T1.1.a) started to define the terminology used to refer to workflows, tools, models and simulation parameters. A *simulation workflow* is the minimal entity that can be executed by the platform. It is an aggregation of simulation workflow components (WFCs). It produces *simulated data* from an *object model* and a set of *simulation parameters*. The main classes of workflow components identified so far are

[1]see in particular activities of the workflow management coalition at http://www.wfmc.org/

pre-processing WFCs, *core simulation* WFCs and *post-processing* WFCs. A pre-processing WFC can be an *object preparation* WFC, a *parameter generation* WFC or a *final parameter assembling* WFC. Image reconstruction is an example of post-processing WFC.

Section 2 of this deliverable presents core simulation workflow components for all the project simulators, namely Sorteo (PET), Sindbad (CT), FIELD-II (US) and SIMRI (MR). For each simulator the workflow structure and the main activities are described. A brief performance study on the European Grid Initiative infrastructure (EGI) is also presented. First plans for pre- and post-processing WFCs are outlined in section 3 . They will be more detailed in D2.1.2 at T0+12. The remaining of this section describes the workflow development and execution environment used in the project.

1.1 Development environment

Workflows are described using the Gwendia language [7] which derives from Scufl [9] and was designed in the Gwendia ANR 2006-2010 project[2]. It is a service-oriented data-driven language where a template directed graph of activities is instantiated on the data available at runtime. Activity invocations are dynamically generated based on the availability of data. Data enters the workflow through its sources and workflow results are eventually produced in sinks. Iteration strategies allow to easily specify how data streams are combined. Data is organized in arrays that can be of arbitrary size and depth. Therefore complex data structures can be described. The language also has a set of control operators among which for and while loops, conditionals and precedence (coordination) constraints.

VIP workflows mainly use the following Gwendia workflow activities:

- Inputs: workflow sources or constants
- Java Beanshells[3], executed by the workflow engine
- Linux executables, wrapped as GASW services [4] and executed on the grid or on the workflow host
- Control constructs: conditionals, loops and array manipulation activities (merge and filter)
- Workflow sinks (outputs)

GASW services are of particular importance since they are used to wrap the command-line tools included in the workflows. Java Beanshells are mainly used to write short pieces of gluing code. Web-Services, DIET SeDs [2] and Virtual (cloud) processors [13] are also available. Gwendia workflows can be described and launched using the MOTEUR2 manager[4].

1.2 Execution environment

Task execution with pilot jobs. The MOTEUR engine can execute GASW activities on various backends. It can run command-lines locally or submit jobs to the DIANE pilot-job management system [8], to the gLite Workload Management System (WMS)[5] and

[2]http://gwendia.polytech.unice.fr
[3]http://beanshell.org/
[4]http://modalis.i3s.unice.fr/moteur2
[5]http://glite.web.cern.ch/glite/documentation

to the ARC middleware[6]. VIP targets a multi-platform workflow execution, supported by the infrastructure of the European Grid Initiative (EGI[7]) and other local clusters. Experience gathered by the Virtual Organizations (VOs) in the EGEE project revealed that it was hardly possible to efficiently exploit such heterogeneous platforms without a pilot-job approach. Similar conclusions have been drawn on the American Open-Science Grid (OSG[8]) where Condor glideIns [12] are extensively used. Additional quantitative information motivating the use of pilot jobs is also available in [1, 11]. For these reasons VIP will exclusively rely on pilot-job execution. The DIANE framework was used for the prototyping reported in this report. In the coming months, the execution framework will be revised in T2.2.a (job management) to provide improved scalability, user management, performance as well as multi-platform capabilities.

Given a set of tasks to execute, pilot jobs are submitted by our pilot factory with a greedy algorithm which keeps on submitting pilots as long as the task queue is not empty. Execution sites are chosen by the gLite WMS. When the task queue is empty (i.e. pilots are ”starving”), the running pilots are killed by DIANE to avoid keeping resources idle. Each user has its own task queue and pool of pilots. When a pilot reaches a worker node, it fetches tasks from the pilot master and start running them. Each task installs the application on the fly on the computing resources. Although it introduces some delay to the execution, this strategy has proven much more convenient than pre-installation, testing and maintenance on the grid sites.

Data transfers. Data storage relies on the system deployed in the EGI. Files can be stored and replicated at various locations and are accessible through a single interface (SRM). A logical file catalog (LFC) provides a uniform view of this distributed storage so that the distribution is actually completely hidden to the applications and users. Basic file permissions can also be defined in the catalog. Additional capabilities will be developed in T2.2.b (data management) to improve the responsiveness, fault-tolerance and management of this system. GASW uses the gLite tools to transfer files. It caches input files on the worker nodes to avoid redundant transfers. Upload tests are done before the job starts to make sure that results could be transferred once the job completes. Results are transferred to the storage element which is closest to the computing node (as configured by the computing site). In case of errors a central storage element is used as a failover. The latency of this kind of data management system is in the order of a few seconds per file. Therefore it is important to reduce the number of transferred files as much as possible for a given amount of transferred data.

Fault-tolerance. Fault tolerance is a major concern in a widely distributed environment. In addition to data transfer checks, GASW resubmits tasks up to a configurable number of times based on the value of their exit code. Moreover, DIANE removes pilots in which tasks failed, which ensures that problematic sites are quickly banned (no “black hole” effect). Pilots also send periodical heartbeats and are discarded in case these heartbeats are not received. Errors still occur in spite of these fault-tolerance mechanisms. The main error causes are (i) data transfers, (ii) communication issues between pilots and the master and (iii) application errors (see detailed statistics in [5]). The error rate is a critical parameter to monitor when workflows are executed on the grid and it significantly

[6]http://www.nordugrid.org/middleware

[7]https://www.egi.eu/

[8]http://www.opensciencegrid.org

VIP_HOME/	models/	ADAM/			
		XCAT/			
	tools/	activities/	model_convertors/		
			simulator_components/	field/	[comp1]/
					[comp2]/
				simri/	
				sindbad/	
				sorteo/	
			other/		
		workflows/	core_simulation/	field/	[wfc1].gwendia
					[wfc2].gwendia
				simri/	
				sindbad/	
				sorteo/	
			pre-processing/		
			post-processing/		
	results/				

Table 1: File catalog structure adopted in the platform.

depends on the workflow (amount of data exchanged, stability of software dependencies, etc.).

Directory structure. Table 1 shows the directory structure adopted in the grid file catalog to store files involved in simulations. This is a basic structure meant to facilitate low-level data management operations (e.g. the handling of file permissions). In the future, file retrieval will rely on the exploitation of semantic data.

2 Core simulation workflow components

During the last 6 months, core simulation workflow components (WFCs) were described and tested for the four simulators of the project, namely Sorteo, Sindbad, FIELD-II and SIMRI. The core simulation WFC of a given simulator is meant to be reused for all the simulations conducted with this simulator in the VIP platform. Such reusability will improve the performance and reliability of grid executions by reducing as much as possible the number of different workflows running on the grid. As a counterpart, these workflows have a quite a low-level interface. They mostly consume a few configuration files that the user has to generate on his/her own. Pre-processing WFCs will provide higher-level interfaces (see first notes in section 3).

A single unique directory is generated to store all the files produced by a core simulation workflow. This is to avoid name conflicts among simulations and to facilitate result retrieval and data management operations (e.g. cleanup, bundling). File bundling (in `zip` or `tgz` archives) is used as much as possible to improve the genericity of the WFCs and to reduce the number of file transfers. In particular, the application dependencies (executable, shared libraries, databases, etc) are bundled in a single package. When possible, input files are also wrapped, e.g., when the organ model consists of several files (see in particular the example of Sindbad in section 2.2). Besides, some simulators produce several output files containing various representations of the result(s) and log files and these outputs often depend on the type of simulation. Since it is not an option to create a specific workflow for each kind of simulation to adjust the number and type of sinks to the produced files, outputs are bundled in single archives as well. Dedicated post-processing workflows could then fish out particular files from these bundles[9].

2.1 PET: Sorteo

Sorteo's core simulation WFC was derived from the parallel version of Sorteo (see manual produced by Simon Marache). Two simulation steps are parallelized: the generation of the singles and the computation of the emissions. As Monte-Carlo computations, these steps may be split in any number of jobs. The implementation of the Sorteo core simulation workflow component from this documentation was straightforward. In/outputs were already well identified and the resulting workflow is supposed to have exactly the same functionality as the original parallel version.

Workflow description. Sorteo's core simulation WFC is diagrammed on figure 1. On this diagram, Beanshell (local) activities are represented by ellipses while GASW (grid) activities are represented by rectangles. Triangles indicate workflow sources and diamonds are sinks. Arrow are data links and circle-terminated lines are coordination (precedence) constraints. Iteration strategies are omitted for legibility reasons. Sorteo mostly consists of two computing steps. First, the "singles" are computed in parallel (activity `sorteo_singles`) and merged (activity `sorteo_single_end`). Then, the "emissions" are computed (activity `sorteo_emission`) and merged (activity `sorteo_emission_end`). Each of these steps correspond to a fork-join workflow pattern. In addition to the simulation directory, only two inputs are required. `text_protocol` contains all the simulation parameters and `fantome_v` is the object model. The workflow produces only a single file (`ccs_file`) that can be converted to a sinogram.

Activity descriptions. Six (6) command-line tools have been wrapped as GASW services. `CompileProtocol` only converts the textual protocol description in a binary format (sink `binary_protocol`). Among other parameters, the protocol specifies how many jobs should be spawned. This number is parsed by activity `parse_text_protocol` and activity `generateJobs` then create process numbers for `sorteo_singles` and `sorteo_emission`. For each process number, `sorteo_emission` produces a modified binary protocol and an "emission" file that are bundled in a `tgz` for performance reasons. Activity `sorteo_single_end` modifies all the protocol files and packs them in a single `tgz`. It also produces a `ccs` file to be updated afterwards. From these files, `sorteo_emission` produces one desintegration file per region in the model and two other files. All these are bundled in a `tgz` and eventually merged by `sorteo_emission_end`.

Performance. A small benchmark was conducted on a PET simulation of a whole-body 128x128x322 object model with 5 active regions. As all the other tests performed in this deliverable, it was conducted in the `biomed` Virtual Organization (VO) of the European Grid Initiative infrastructure (EGI) which gathers some 220 computing centers

[9]In particular, this will be necessary to annotate data at a fine granularity.

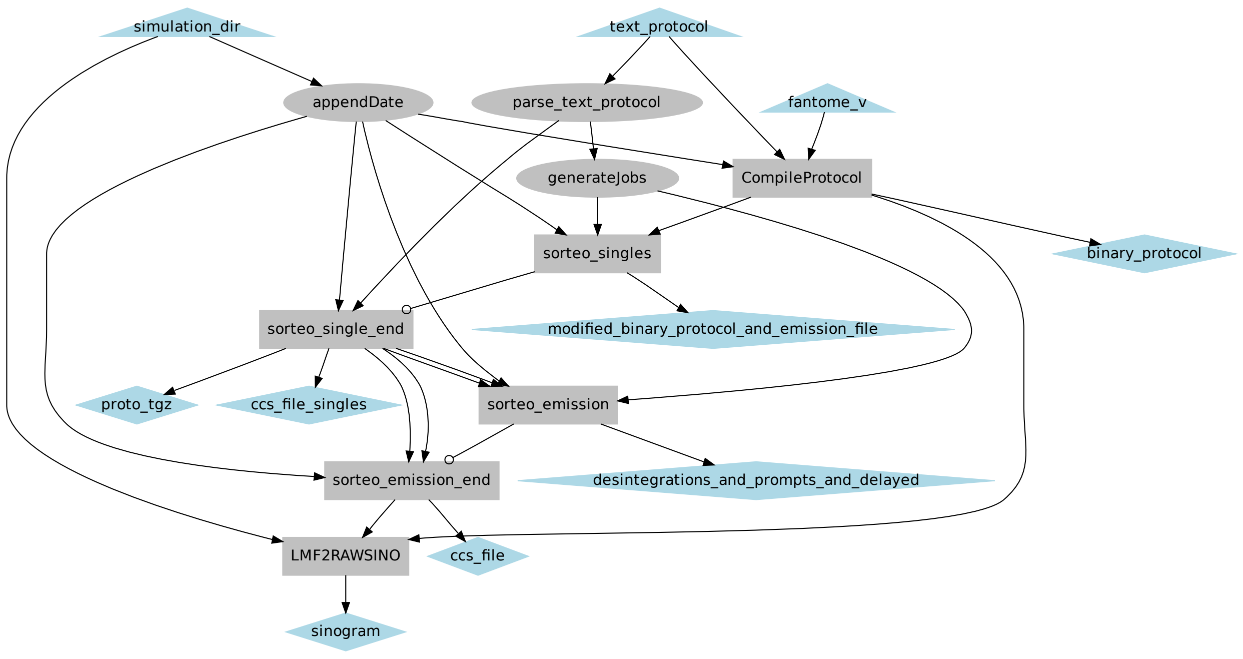

Figure 1: Sorteo core simulation workflow component.

world-wide. The single and emission steps were split into 80 jobs. Table 2 reports the performance achieved from 3 repetitions of this simulation. The elapsed time corresponds to the total duration between the beginning of the workflow and the completion of its last task (a.k.a makespan, i.e., the time perceived by the user). The CPU time and in/output down/upload times are cumulative values on all the simulation tasks. The number of submitted task is the sum of both completed and failed tasks. Failed tasks may have been compensated by resubmission. The target number of tasks is the number of tasks which have to be completed at the end of the simulation (162 per run in this case). Missing tasks are tasks that could not be computed in spite of resubmissions.

The average speed-up obtained in these conditions is 3.5, which is not much. This is partly explained by the high data/CPU ratio (around 40%). Obviously, too many jobs were created for this short simulation. The average duration of a task was 102 seconds, which is quite short on a high-latency infrastructure like the EGI. Moreover, as mentioned above, the Sorteo workflow has a double fork-join, which leads to some performance issues with the current pilot submission mechanism. Many pilots are submitted to support the first fork (`sorteo_singles`). These pilots starve when the first join runs (`sorteo_single_end`) and therefore are not available for the second fork (`sorteo_emission`). A new set of fresh pilots has to be resubmitted, which introduces some delays in the execution. Figure 2 illustrates this issue. Reliability was not an issue for this test. Overall the execution of the Sorteo binaries did not suffer any particular problem on the EGI. They have no external dependencies and are quite portable. The bundling of files in `tgz` archives also contributed to reduce the number of transferred files, therefore errors.

	run 1	run 2	run3
Task execution			
CPU time (s)	16803	17192	17484
Elapsed time (s)	4290	4268	6677
Speed-up	3.9	4.0	2.6
Data transfers			
Input download (s)	4870	4712	4863
Results upload (s)	1934	1735	1747
Data/CPU ratio	0.4	0.38	0.38
Reliability			
Total submitted tasks	164	164	166
Failed tasks	0	0	2
Target	164	164	164
Missing	0	0	0

Table 2: Performance analysis of the Sorteo WFC for a whole body simulation split in 80 jobs, running on the EGI grid.

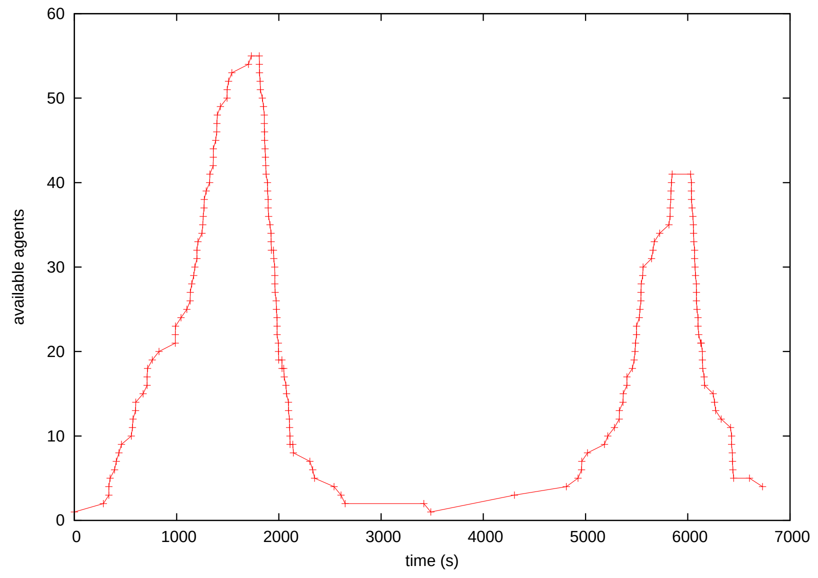

Figure 2: Typical evolution of the number of available resources (pilots) during a Sorteo simulation. Time '0' corresponds to the simulation launch time. The simulation makespan was 6677 seconds. Soon after the beginning of the simulation, 80 `sorteo_single` tasks are submitted which leads the system to start provisioning pilots. When all tasks are running (from t=1700s), pilots start to die from starvation. Eventually, only a single pilot remain to compute the first merging task (`sorteo_single_end`). In this case this task failed twice. From t=3500s, 80 `sorteo_emission` tasks are submitted and the system starts to provision pilots again. From t=5900s, pilots start to starve again and only a few remain in the system while `sorteo_single_end` and `LMF2RAWSINO` execute. The provisioning system proved very inefficient in this case.

2.2 CT: Sindbad

A Sindbad CT simulation consists of a set of projections. Parameters are gathered in a chain file. A geometry file describes the simulation scene and the scanner parameters are gathered in a scan file. For each projection, an analytical step and a Monte-Carlo step are computed. The results of these two steps are then combined to produce the final simulation result.

Three levels of parallelism can be exploited:

1. projections of a CT scan can be computed independently from each other
2. within a projection, the analytical and Monte-Carlo steps can be computed in parallel
3. the Monte-Carlo computation can itself be parallelized

Level (1) was first implemented and validated. First performance tests showed that the Monte-Carlo step could be very long and therefore levels (2) and (3) were also implemented. Overall, the resulting workflow component remains highly customizable and parallelization levels (2) and (3) can be (de-)activated on demand.

In the same spirit, care has been taken to keep the core simulation WFC generic enough to cover various cases of simulation input. In particular, a different phantom is sometimes used for the Monte-Carlo step, which also creates differences in the chain file and in the geometry file. Proper configuration files were setup to describe these changes.

2.2.1 Workflow description

Workflow structure. The Sindbad core simulation workflow component is represented on figure 3. It covers any kind of parallelisation and input configuration mentioned above. The most important activity of this workflow is `sindbad`. This activity simulates only one projection of the complete scan. It is iterated to produce a complete scan. Depending on its parameters it computes either the whole projection, or the analytical step only, or the Monte-Carlo step only, or a sub-part of the Monte-Carlo step. Activities `projection_count`, `get_min_element` and `split_count` define the granularity used by `sindbad`. Activity `merge` merges the sub-parts of the Monte-Carlo simulation into a single result. Moreover, it assembles the analytical simulation with the Monte-Carlo. It follows two conditionals (`test_use_split` and `test_start_merge`) that determine which files have to be merged. Eventually the user retrieves a set of 360 simulated projections. These have to be transferred back to some (local) storage place to enable an efficient transfer to local workstations. Activity `putOnWebServer` implements that. It is common to several core simulation workflows and further described in section 2.5.

Constants. Sindbad WFC has 10 inputs, including the simulation directory. Among these, 3 are constants controlling how the simulation is split into grid jobs (a.k.a the granularity of the simulation). `nb_job_max` is an upper bound on the number of jobs that can be submitted by this workflow. `nb_particles_per_job_max` is the maximal number of particles per Monte-Carlo job. If the simulation chain contains a total number of particles $n > nb\_particles\_per\_job\_max$, then the Monte-Carlo step is evenly split into $\lceil \frac{n}{nb\_particles\_per\_job\_max} \rceil$ jobs[10]. In this case, the analytical simulation is always separated

[10] If the total number of particles to simulate is not a multiple of $\lceil \frac{n}{nb\_particles\_per\_job\_max} \rceil$ then one job simulates the remaining particles.

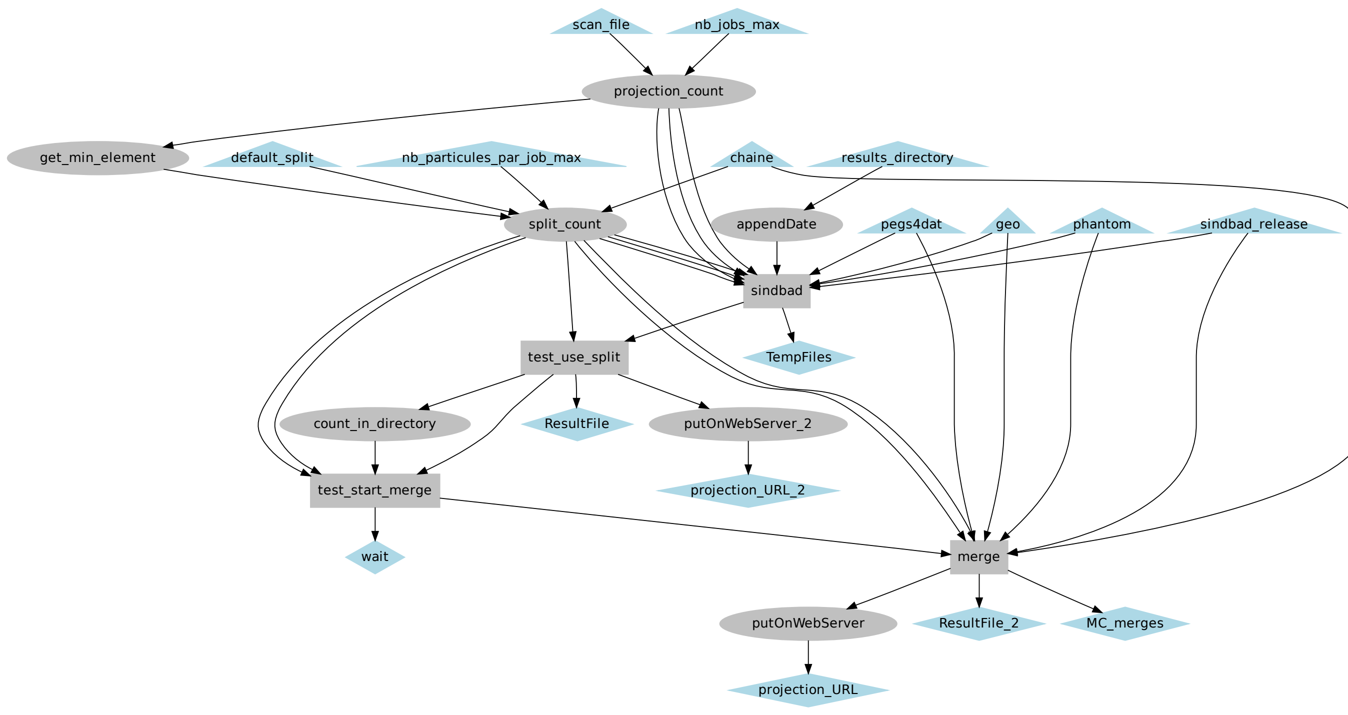

Figure 3: Sindbad core simulation workflow component.

from the Monte-Carlo. Constant `default_split` controls the simulation granularity when $n < nb\_particles\_per\_job\_max$ as follows:

- `default_split` = 0: only one job is created for each projection. This job simulates both the analytical step and the Monte-Carlo.
- `default_split` = 1: for each projection, two jobs are created, separating the Monte-Carlo from the analytical steps.
- `default_split` >1: the analytical step is separated from the Monte-Carlo step *and* the Monte-Carlo step is divided in `default_split` jobs.

Input files. Six (6) workflow sources are input files. `scan_file` defines the initial position of the scanner, the rotation center, the total number of projections, the angular and linear distance between two projections as well as the angular and linear speed. It is strictly identical to the one used in the regular sindbad software. The `chain` file contains all the acquisition parameters, except some geometrical variables. Again, the format is exactly the same `mdf` file as in the regular sindbad software. However, as mentioned above, some simulations require different parameters for the analytical step, the Monte-Carlo step and their combination. In this case, different chains can be bundled in a `tgz` file and a configuration file indicates which `mdf` file to use for each simulation step. This configuration file also defines which steps have to be computed. The same solution is used for the `geo` input which contains the same geometrical (scene) parameters as the `geo` file used in the regular sindbad. The model is specified in the `phantom` source. Depending on the simulation type it can be either a `g` BRLCAD mesh, or a `sdt/sdt` labeled image or both. In case several files are required they are bundled in a `tgz` and a configuration file specifies which should be used at any simulation step. Note that if the phantom is the only parameter to switch between the analytical and the Monte-Carlo simulation then there is no need to use several chain files as the regular sindbad imposes. Input `pegs4` contains the specification of the materials used in the BRLCAD phantom. Finally, `sindbad_release` contains the sindbad executables and all their dependencies.

Detailed activity description. Activity `projection_count` extracts the number of projections from the scan file, compares it with `nb_job_max` and generate indexes for all the projections to simulate. The maximal number of jobs per projection is also passed to activity `split_count` (*via* `get_min_element`) that generates indexes for the Monte-Carlo step. Activity `sindbad` extracts the release, phantom, geometry and chain files. Then it reads the parameters and adjust the scan and chain files according to the projection and Monte-Carlo indexes and to the steps to be computed. The seeds of the random generators are also generated to ensure their uniqueness among jobs[11]. Then the simulation is launched and a three-step check is performed to detect inconsistencies as early as possible. Eventually all the files produced by the simulation are bundled in a `tgz` file. `test_use_split` checks whether a merge operation is required. If not (`default_split` = 0) then the result is directly transferred to the user by `putOnWebServer`. Otherwise, `test_start_merge` checks whether all sub-parts of the projections are ready to be merged and launches `merge` accordingly. The latter sums all the Monte-Carlo results and then combines the analytical and Monte-Carlo results with the appropriate sindbad operation. The final result of each projection is eventually transferred to the user.

2.2.2 Performance study

Two different simulations were considered to evaluate the performance of the workflow on the grid: *Organs* is the example provided in the Sindbad release and *GXL* is the simulation currently considered in task T3.1 of the project.

Both simulations consist of 360 projections with analytical and Monte-Carlo steps. For *Organs*, 1 million of particles were used for the Monte-Carlo step on each projection and a single job was submitted for each projection For *GXL*, the number of particles per projection varied from 0 (pure analytical) to 200 millions. For 0, 50 and 100 millions of particles, only the first level of parallelism was exploited (parallelization on projections only). For 200 millions, two tests were conducted: one without any Monte-Carlo parallelization and one with the Monte-Carlo step split in 5 jobs. Note that the following results do not account for the merging of Monte-Carlo results since this step was still being debugged at the time this report was written. Results are shown on table 3. The final transfer corresponds to the contribution of activity `putOnWebServer` to the makespan.

Job execution. Overall the speed-up obtained with this workflow ranges from 5 to 190. The average speed-up for *Organs* is 44. For *GXL* it reaches 85 when the Monte-Carlo step is not parallelized and it rises up to 190 when it is. Even pure analytical *GXL* simulations receive speed-up, although it is not very significant (in the order of 5). Coherently, the speed-up increases with the total CPU time. Significant variations in the makespan are observed. A factor 3 between the minimum and maximum observed makespan is common.

[11]This is of particular importance to avoid artifacts such as described in [14].

Data transfers. As expected, the impact of data transfers decreases when the CPU time increases. Pure analytical *GXL* simulations are obviously data bound: the total data transfer time (input download + results upload) is comparable to the total CPU time. For *Organs* the data transfer time represents some 11% of the total CPU time, which is usually considered efficient for applications running on the EGI. *GXL* simulations with 50 and 100 millions of particles are definitely CPU bound, with a data transfer ratio below 2.5%. Parallelizing the Monte-Carlo step increases the amount of data transfers. Yet the data transfer ratio remains below 1.5% for the 200-million *GXL* run with Monte-Carlo parallelization. Table 3 does not show the time to transfer results from the grid to the user host at the end of the simulation. This process is a real issue and is currently both unreliable and inefficient. It is discussed in section 2.5.

Reliability. Reliabilty strongly depends on the average task duration. The execution is quite reliable for *Organs* and *GXL* simulations with less than 100 millions particles. For *Organs*, one task hit the maximal number of retries (yet set to 3) and therefore could not be completed. Ten (10) retries were performed for the other tests. *GXL* analytical and 50 millions of particles are quite reliable, with less than 6.5% of failures. *GXL* 100 millions had poor reliability: 23% of the jobs failed and 2 could not be completed after 10 resubmissions. This is mostly due to proxy expirations coming from wrong initializations[12]. These issues were exacerbated for 200 millions particles with no Monte-Carlo parallelization which could never complete due to a 57% failure rate. In this simulation, many tasks run for more than 24 hours, which makes proxy renewal necessary. Renewal issues prevented the simulation from completing. The proxy renewal mechanism will be worked out during the project, in particular through the use of a myproxy server. By decreasing the average duration of a task, the parallelization of the Monte-Carlo step allowed the 200-million particle *GXL* simulation to complete. In this case the average error ratio was reduced to 2.2%. This experiment reveals the gain in parallelizing the Monte-Carlo step of Sindbad for an execution on EGI. However, this parallelization has to be carefully tuned since spawning too many Monte-Carlo jobs would have a dramatic impact on the efficiency (data transfer ratio and reliability) of both the simulation and the merging procedure. Using a dynamic parallelization of the Monte-Carlo step such as described in [1] will be considered.

2.3 Ultrasound: FIELD-II

FIELD-II is a bit particular since it is more a simulation toolbox than an on-the-shelf simulator. FIELD-II provides a Matlab API to setup ultrasonic (US) simulations. Therefore a first step in the integration process was to identify a generic workflow that could be used to parallelize any kind of US simulation. The selected solution was to exploit data parallelism on the simulated radiofrequency (RF) lines. Eventually, all FIELD-II simulations compute a set of RF lines that can be generated independently from each other. The Matlab code generating these RF lines is a real parameter of the simulation. Depending on the type of the simulation, it can take very different forms and it should remain easily parametrized by the user.

Structure of the workflow component. The resulting FIELD-II core simulation workflow component is shown on figure 4. `SimulateRFLine` is the main activity. It is

[12]see `https://savannah.cern.ch/bugs/?70098` about this issue.

		Organs 10^6 part.	GXL 0 part (an. only)	50.10^6 part.	100.10^6 part.	200.10^6 part.	200.10^6 part (MC parallelized)
number of repetitions		12	12	6	6	5	4
Task execution							
CPU time (h:min)	min	135:31	5:19	821:08	1637:52	-	3404:50
	max	155:13	7:08	1024:58	1909:55	-	3610:18
	mean	146:28	5:49	914:14	1743:57	-	3502:56
Elapsed time (h:min)	min	1:52	0:15	7:49	16:31	-	14:01
	max	9:22	2:34	19:09	34:45	-	31:15
	mean	3:20	1:09	10:50	23:50	-	20:16
Speed-up (average)*		43.9	5.1	84.4	73.17	-	172.84
Data transfers							
Input download (h:min)	min	2:39	1:55	9:15	13:59	-	39:43
	max	50:58	7:19	35:03	34:34	-	45:34
	mean	11:23	3:48	19:35	22:33	-	41:35
Results upload (h:min)	min	1:10	1:14	1:47	1:48	-	6:36
	max	25:28	4:17	3:24	6:51	-	9:14
	mean	6:22	2:05	1:01	3:09	-	7:50
Data-CPU ratio (average)*		0.12	1.01	0.02	0.01	-	0.01
Reliability							
Total submitted tasks		4441	4333	2301	2817	2979	7369
Failed tasks		122	13	141	659	1725	169
Target		4320	4320	2160	2160	1800	7200
Missing		1	0	0	2	546	0

Table 3: Performance analysis of the sindbad core simulation workflow component. Except the last GXL-200 millions, all the simulations were parallelized only on the projections (no Monte-Carlo parallelization). * The average speed-up (resp. Data-CPU ratio) was computed as the ratio between the average CPU time (resp. data transfer time) and the average elapsed time (resp. CPU time), which is a bit abusive.

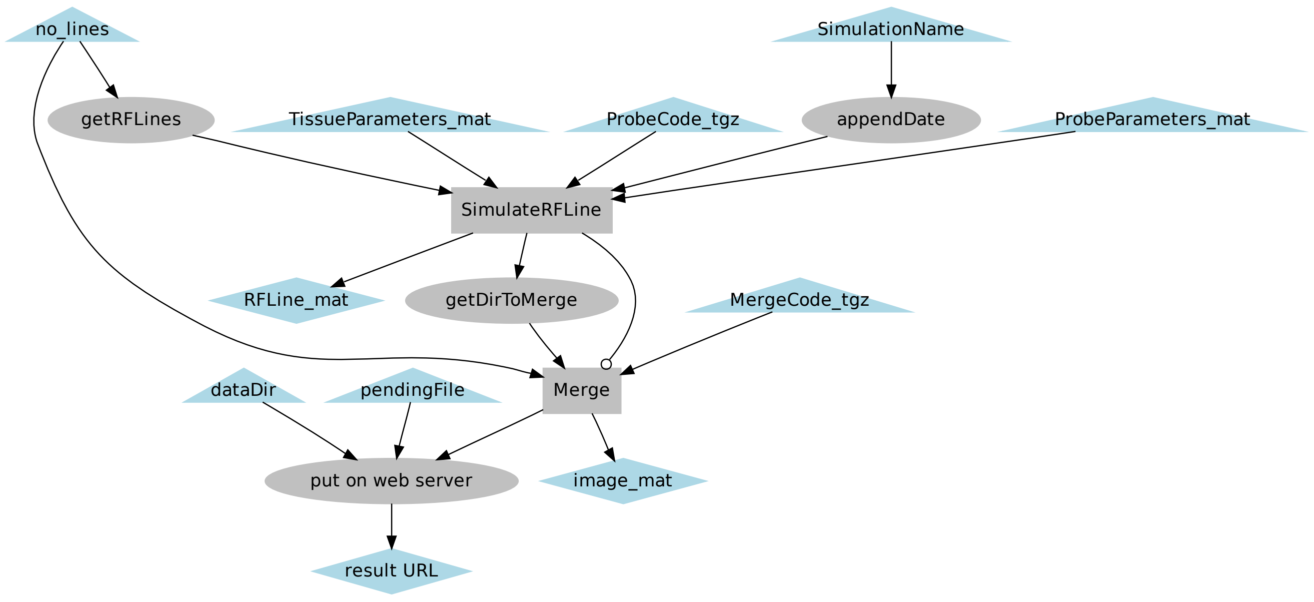

Figure 4: FIELD-II core simulation workflow component.

iterated over the line number array generated by `getRFLines` from input `no_lines`. In general the number of RF lines strongly depends on the probe type. Therefore the degree of parallelization is imposed by the simulation and cannot be adjusted in the workflow. In this respect FIELD-II differs from the other simulators. For the reasons mentioned above, the US probe code is also an input parameter (`ProbeCode_tgz`). The model to simulate is represented by input `TissueParameters_mat` which contains one or several Matlab data files. Each of these consists of a vector of coordinates and a vector of amplitudes defining the scatterers of the model. The probe parameters are also gathered in a Matlab data file represented by input `ProbeParameters_mat`. Once generated, the RF lines are assembled by activity `Merge`. Again, the implementation of this activity is very dependent on the type of simulation. Therefore its code is also an input of the workflow. Finally, activity `putOnWebServer` transfers the merged results back to a local web server.

Activity descriptions. `SimulateRFLine` and `Merge` are both Matlab code. Among various alternates, the solution chosen here is to compile these codes using the Matlab Compiler (MCC) and to execute them with the Matlab Compiler Runtime (MCR) which can be used without license. Due to the Mathworks policy, MCC and MCR are tightly coupled with the target execution architecture and OS[13]. It is not really a problem on the EGI since the targeted population of worker nodes can be restricted to RHEL5-like 64-bit machines without much loss of resources. However this may cause some issues when the VIP platform will be connected to various clusters, as it will be delivered at PM12. Each cluster should then host its own version of MCC/MCR or of the regular Matlab. To prevent from painful installation and maintenance the MCR is installed on-the-fly on EGI

[13]For instance executables generated on a 32-bit platform won't work on a 64-bit.

	run 1	run 2	run3
Task execution			
CPU time (s)	179,171	194,070	181,868
Elapsed time (s)	10,869	10,960	10,639
Speed-up	16.5	17.7	17.1
Data transfers			
Input download (s)	30,749	32,123	13,821
Results upload (s)	5,368	1,825	1,382
Data/CPU ratio	0.2	0.17	0.08
Reliability			
Total submitted tasks	75	73	69
Failed tasks	10	8	4
Target	65	65	65
Missing	0	0	0

Table 4: Performance analysis of the FIELD-II core WFC for a 10^6-scatterer simulation split in 64 jobs, running on the EGI grid.

worker nodes. This strategy certainly increases data transfers but their impact remains limited due to the caching implemented by pilot jobs. `getDirToMerge` just extracts the directory containing all the RF lines from a simulation result.

Performance. A test was conducted on a 10^6-scatterer simulation with a 64-line probe. Table 4 shows the result of this test. Overall, a fair speed-up of 15 to 20 is obtained in spite of the low number of tasks. Data transfers correspond to some 20% of the CPU time, which is significant. In particular, the input download time accounts for the transfer of the Matlab Compiler Runtime environment which is 180 MB. In this case caching input data in pilots is of little use: given the low number of tasks and their long duration, each task executes in a different pilot. The error rate is in the order of 15%. Most of these errors come from configuration issues with the MCR on SELinux-enabled nodes and should be solved with a thorough fine-tuning. Merging the results is a long operation. As shown on figure 5, the merging procedure accounts for 1/4 of the makespan (see last task at the right of the figure). Moreover this merge operation mostly consists of data transfers and its duration only depends on the number of files to merge, i.e., of probe lines. Its impact will therefore be even greater for smaller simulations.

2.4 MRI: SIMRI

Simri[14] is an MRI simulator that has been parallelized using MPI.

Structure of the workflow component. Due to its implicit MPI parallelization, Simri does not need a merging phase like the previous simulators. Therefore, its core WFC is rather simple, with one computing step (corresponding to the simulator itself), multiple inputs and one output. The Simri WFC is diagrammed on figure 6.

The inputs correspond to the Simri release (executable + shared libraries), the object to simulate, two parameter files (one for object related parameters and one for MRI

[14] http://simri.org/

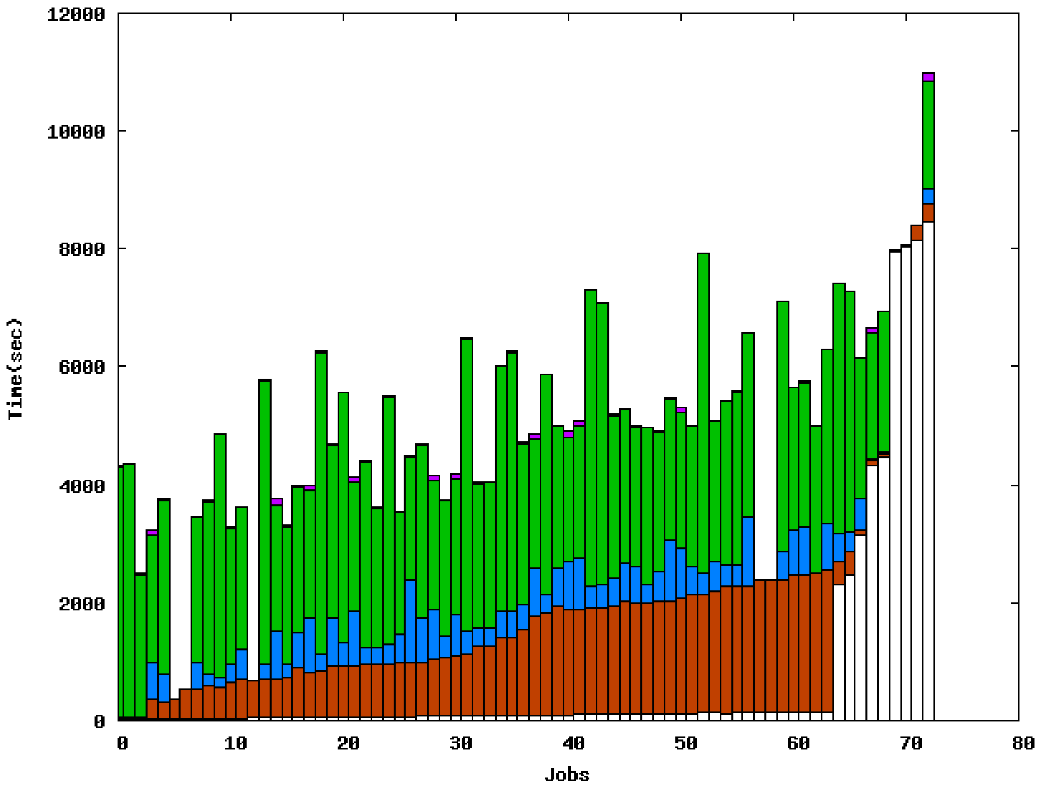

Figure 5: Typical execution diagram of the FIELD-II core simulation workflow. Brown indicates queuing time, blue is input transfer time, green is application running time and purple is output transfer time.

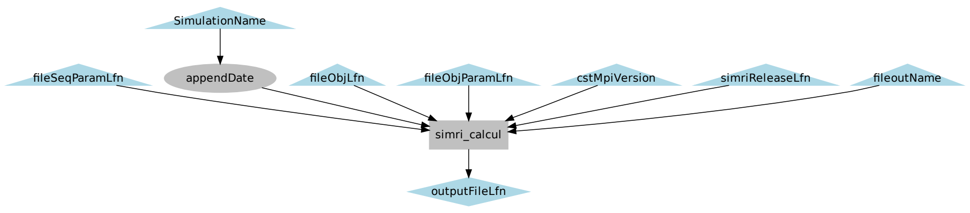

Figure 6: Simri core simulation workflow component.

sequence parameters), the output folder and file name, as well as the name of the MPI implementation. This last parameter is needed in order to properly launch an MPI executable on the grid. The simulator in use at the moment within VIP uses MPICH2.

In order to run an MPI executable, we must specify the number of cores on which it can execute in parallel. The number of cores should therefore be an input of the workflow. Nevertheless, for implementation reasons this number must be given in the activity (GASW) descriptor. Consequently, at present we have multiple Simri workflows corresponding to different activity descriptors, each with its own number of cores (4, 16, 32 and 64).

Activity description. Originally Simri simulated predefined objects. Adding external objects was not difficult but required modifying and recompiling the code. Within the VIP project, Simri was modified in order to simulate new (external) objects without having to touch the Simri release. External objects are described with a pair of mdh/raw files representing a label map which consists of integers (labels) representing the tissue at each pixel/voxel. In addition to this, two more input files are needed. A labelToMatter.txt file associates the labels used with generic matter names, while a matters.xml file associates physical values (rho, t1, t2, Khi and their standard deviation) to each matter. A pre-processing WFC assembles these object related inputs into one archive and prepares the parameter files with the corresponding parameters.

This external object functionality has already been tested within VIP with a heart-thorax 3D model generated within the ADAM project. ADAM provided the model (3D label map and the two label/matter files) and Simri produced the simulated MRI.

Performance. MPI parallelization frees us from the merging burden but introduces higher grid execution constraints. First, there are rather few EGI grid sites (20 to 25) supporting MPI. Second, regular MPI execution is mainly mono-site, making it difficult to achieve a high number of cores in parallel for one MPI job. Last but not least, MPI job execution on distant sites is more prone to failures than no-MPI jobs (among others because the heterogeneous MPI site configuration).

First results obtained when executing the Simri are summarized in table 5. They correspond to a simulation taking about 1400s on an Intel dual core at 3.00GHz (without MPI, using only one core). Values given here should be taken more as qualitative rather than quantitative results. Grid conditions present significant fluctuations, therefore the 4 repetitions presented here are not enough in order to draw final conclusions.

It can be noticed that execution speed-up is almost linear when using 4 and 16 nodes.

Core number	**Avg WFC makespan (s)**	**Avg execution time (s)**	**Success ratio (%)**
4	906	375	75%
16	1813	97	100%
32	1137	54	100%
64	575	38	75%

Table 5: Results obtained when executing the Simri core WFC on 4, 16, 32 and 64 nodes. The same simulation was repeated 4 times for each of the 4 workflows. The makespan represents the total elapsed time from workflow submission till completion, including queuing time. Execution time corresponds to the elapsed time from the moment Simri is executed on the distant node till output is ready.

This is not the case with 32 or 64 nodes probably because of the job duration (rather small for this case) and the cost of MPI communications.

We can see that if the execution time is considerably smaller when using more cores, this is not necessarily the case for the makespan. For 16 and 32 nodes, in 3 out of 4 executions the makespan was significantly smaller (about 500s), but one of the 4 workflows needed a significant amount of time to get running, which increased the average. This one longer workflow corresponds roughly to the 2 unsuccessful jobs (for the 4 respectively 64 core-workflows). These jobs remained blocked (queued or running) for more than 24h and ended up in failure. If we were to count their makespan, it would completely dominate the average.

2.5 General issues

Merging simulation results is a painful operation for Sorteo (see comments on figure 1) and FIELD-II (see figure 5). This issue will also appear when CT images are to be reconstructed from projections simulated with Sindbad. The main issue comes from the transfer of a high number of files (100+) from heterogeneous locations, potentially spread all over the world. It raises both a performance (mostly latency) issue and a reliability problem. This issue is specific to large-scale distributed systems. It is also of outmost importance for other simulators running such systems, for instance GATE. Although it is regularly mentioned when experiments are conducted on large-scale platforms (see, e.g., [10, 3]), it does not seem to be well studied in the literature.

For the same reasons, transferring results back to the user is a common issue, in particular when results consist of many files. The current solution is implemented by activity `put on web server` showing in the Sindbad and FIELD-II core simulation workflows. This activity enqueues a grid logical file name in a FIFO linked to a data transfer daemon. This daemon fetches files from the grid and put them in a web server accessible with good reliability and reduced latency by the user. In case the grid file cannot be accessed, it is added to a list of problematic files. This list is periodically browsed and file transfers are retried until they succeed. This process works as a proof of concept but it has to be implemented in a more robust way and triggered directly by the workflow engine for all result files. This is a development task of the VIP data manager (T2.2.b).

Properly tuning the number of tasks of a simulation is another issue common to all simulators. Determining the number of cores to use for an MPI simulation (for SIMRI) or the number of tasks in which a Monte-Carlo simulation should be split (for Sindbad and Sorteo) is not trivial at all on heterogeneous systems. For FIELD-II (resp. Sindbad), different RF lines (resp. projections) could also be grouped in a single task to reduce the amount of generated jobs, therefore potential errors. In fact deciding on the number of tasks (granularity) is linked to the strategy used to submit pilot jobs (resource provisioning) and on the algorithm used to assign tasks to pilots (scheduling). It depends on the size of the simulation and on the availability of computing resources. When most of the components of the job manager are in place, global strategies to address this scheduling issue will be proposed.

3 Other workflow components

Object preparation WFCs. Object models exist in a variety of formats and their representation often has to be significantly adjusted to fit the simulators' inputs. Object preparation WFCs aim at the following tasks:

1. convert the object model to appropriate geometrical representation(s)
2. convert the object model files to the appropriate format
3. define the simulation scene
4. set the physical parameters of the object

Regarding (1) and (2), a set of converters will be available in the platform. These will be appropriately annotated so that the platform could propose a set of converters when a new model is integrated. To reduce the number of required format converters, an intermediate anatomical model format (IAMF) will be used[15]. So far the IAMF description consists of the following:

- 3D model, in meshed (VTK polydata) representation, one file per model class.
- 3D model, in voxel (VTK mhd/zraw) representation. A lookup table associates labels to class names.

It will probably be extended when models are added to the platform. So far, one volume of the ADAM cardiac model was converted to IAMF and only SIMRI can process it.

The simulation scene is then defined. An arbitrary coordinate system (Rm) is associated to the object. Each scanner involved in the simulation is then positioned in Rm. Depending on the simulator, a scanner may be represented with a single coordinate system (e.g. MR scanner, FIELD-II US probe and PET scanner) or with two (e.g. source and detector in CT). A geometrical transformation of the model to the coordinate system of each simulator involved in the simulation can then be produced.

Physical parameters are then determined. For PET, activity values are provided by the user for each tissue class in the object model. For CT, materials are assembled from tables provided by the International Commission on Radiation Units and Measurements[16]. For US, scatterer distributions (spatial and amplitude) will have to be selected for each tissue class. For MR, the mean and standard deviation of the parameters could either be taken from an existing database or asked to the user.

[15]Converting n model formats to m simulator input formats requires m+n convertors with IAMF while $n \times m$ would be required otherwise.

[16]http://www.icru.org/

Parameter generation WFCs. The goal of parameter generation WFCs is to assemble simulation parameters to generate configuration files to be consumed by core simulation workflows. Simulation parameters can be either specified by the user or fixed as constants. For a given simulator, pre-processing workflows are very much dependent on the type of conducted simulation. As soon as two types of simulation have different parameter sets, a different parameter generation WFC has to be created. For instance, each MR sequence will have its own parameter generation workflow. Different types of X-Ray source or detector will also lead to different parameter generation workflows in CT. Similarly, each US probe will have its own parameter generation WFC. For SIMRI and for Sorteo, parameter generation workflows will consist of very simple activities that just concatenate parameter names and values in text files. No particular difficulty is expected for these simulators. For FIELD-II the technical situation is a bit more complex due to the handling of Matlab files. The activities of the parameter generation workflow will have to be executed on a host where Matlab is available. For Sindbad, the difficulty comes from the number of parameters involved in the parameter generation. It consists of spectrum/source generation, detector creation, core parameter generation, scanner parameter generation and geometrical parameters. An example is shown on figure 7.

Post-processing WFCs. Some kind of post-processing is already included in the core simulation workflows described in section 2. Post-processing can consist of format conversions of the simulated data (see for instance activity `LMF2RAWSINO` for Sorteo) or image reconstruction (activity `Merge` for FIELD-II). The latter is very much dependent on the type of computed simulation (probe type in US, detector shape in CT) and will have to be adjusted to the input parameters.

4 Conclusions and future work

Workflows are first-class objects of the project. They are common discussion objects among the various axes. They will be executed by the applications axis, they are an information sources for the semantic modeling, they are structured based on the semantic analysis and they are the priority test cases for the execution service.

The core simulation workflows described in this deliverable are meant to support the heavy computational parts of the simulations. They can be executed on appropriate computing infrastructures. Performance tests were conducted on the European Grid Initiative infrastructure (EGI). Speed-up values up to 280 were observed for the heaviest tested simulations (3500 CPU hours - Sindbad CT). In all cases, data transfers remain a critical issue. In particular, the merging the simulation results in FIELD-II and Sorteo should be improved to provide better speed-up. The transfer of simulated data to the user is also painful, in particular when it consists of several files. The data manager provided by T2.2.b will greatly improve that. Besides, the simulation granularity tuning (number of grid tasks for a given workload) should be improved in T2.2.a. The performance data reported here could serve as a reference to quantify the gain yielded by the coming improvements.

Some work still has to be done to improve the integration of the simulators. In particular, pre-processing workflow components were only sketched yet. Object preparation and parameter generation WFCs are the next scheduled workflow developments. Post-processing WFCs will follow.

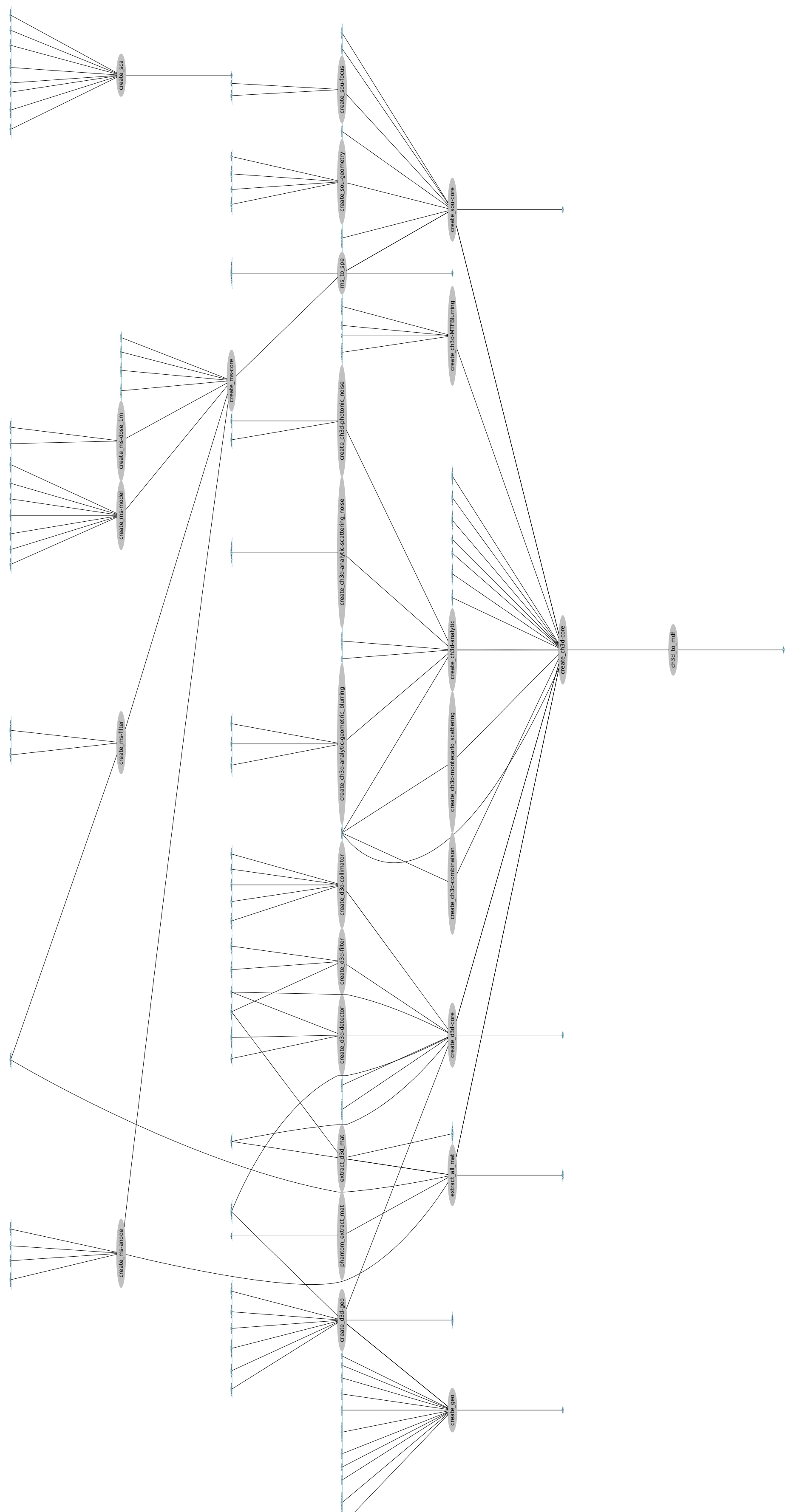

Figure 7: A Sindbad parameter generation workflow. The source is calculated from model parameters and the detector is standard.

References

[1] S. Camarasu-Pop, T. Glatard, J. T. Moscicki, H. Benoit-Cattin, and D. Sarrut. Dynamic partitioning of gate monte-carlo simulations on egee. *Journal of Grid Computing*, 8(2):241–259, 2010.

[2] Eddy Caron and Frédéric Desprez. Diet: A scalable toolbox to build network enabled servers on the grid. *International Journal of High Performance Computing Applications*, 20(3):335–352, 2006.

[3] Gerardo Ganis, Jan Iwaszkiewicz, and Fons Rademakers. Scheduling and load balancing in the parallel root facility (proof). In *XI International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, pages 23–27, Amsterdam, The Netherlands, April 2007.

[4] T. Glatard, Johan Montagnat, David Emsellem, and Diane Lingrand. A service-oriented architecture enabling dynamic services grouping for optimizing distributed workflows execution. *Future Generation Computer Systems*, 24(7):720–730, 2008.

[5] Ting Li, Sorina Camarasu-Pop, Tristan Glatard, Thomas Grenier, and Hugues Benoit-Cattin. Optimization of mean-shift scale parameters on the egee grid. In *HealthGrid'10*, Paris, 2010.

[6] Johan Montagnat, Tristan Glatard, Isabel Campos, Francisco Castejon, Xavier Pennec, Taffoni Giuliano, Vladimir Voznesensky, and Claudio Vuerli. Workflow-based data parallel applications on the EGEE production grid infrastructure. *Journal of Grid Computing (JOGC)*, 6(4):369–383, December 2008.

[7] Johan Montagnat, Benjamin Isnard, Tristan Glatard, Ketan Maheshwari, and Mireille Blay-Fornarino. A data-driven workflow language for grids based on array programming principles. In *Workshop on Workflows in Support of Large-Scale Science(WORKS'09)*, 2009.

[8] Jakub T. Mościcki. Distributed analysis environment for HEP and interdisciplinary applications. *Nuclear Instruments and Methods in Physics Research A*, 502:426429, 2003.

[9] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, and Peter Li. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics journal*, 17(20):3045–3054, 2004.

[10] Ian Stokes-Ress, Francoise Baude, Viet-Dung Doan, and Mireille Bossy. Managing parallel and distributed monte carlo simulations for computational finance in a grid environment. In Simon C. Lin and Eric Yen, editors, *Grid Computing*, pages 183–204. Springer US, 2009.

[11] Wen-Jun Tan, Chuen Teck Mark Ching, Sorina Camarasu-Pop, Pascal Calvat, and Tristan Glatard. Two experiments with application-level quality of service on the egee grid. In *Second Grids Meet Autonomic Computing workshop (GMAC'10)*, Washington DC, USA, 06/2010 2010.

[12] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency and Computation: Practice & Experience (CCPE)*, 17(2–4):323–356, 2005.

[13] Tram Truong Huu and Johan Montagnat. Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure. In *2nd International Symposium on Cloud Computing(Cloud 2010)*, , Melbourne, Australia, May 2010. IEEE Computer Society.

[14] Shu-Ju Tu, Chris C Shaw, and Lingyun Chen. Noise simulation in cone beam ct imaging with parallel computing. *Physics in Medicine and Biology*, 51(5):1283, 2006.