Using Workflows to Control the Experiment Execution in Modeling and Simulation Software

Stefan Rybacki
University of Rostock
Institute for Computer Science
Albert-Einstein-Str. 22
Rostock, Germany
stefan.rybacki@uni-rostock.de

Jan Himmelspach
University of Rostock
Institute for Computer Science
Albert-Einstein-Str. 22
Rostock, Germany
jan.himmelspach@uni-rostock.de

Adelinde M. Uhrmacher
University of Rostock
Institute for Computer Science
Albert-Einstein-Str. 22
Rostock, Germany
adelinde.uhrmacher@uni-rostock.de

ABSTRACT

To control the computation of an experiment means at least to deal with the generation of parameter combinations of interest (at best using experiment design solutions) and to execute, in case of stochastic simulations, the required replications using the hardware available.

Although the process to be supported by modeling and simulation (M&S) software is often constrained to loading the model and instantiating it using a given parameter configuration and executing it by taking into account the simulation parameters, needs might arise to adapt and extend the code controlling the execution, i.e., incorporating feedback loops from validation or analysis steps into the execution scheme during runtime, having user interaction or simply adding further steps to the execution scheme such as further data processing. Additional features, like supporting diverse hardware setups, documentation or security, may imply further changes to the code.

A workflow-based execution abstracts from concrete implementations and hard-coded execution patterns, as it provides a declarative description of this process, which means that anyone can set up own experimentation workflows (by using smaller predefined workflows and non-workflow-based components). Herein we present a workflow driven realization of an experimentation layer, which supports the same features as the hard-coded alternative and we discuss the pros, cons, and performance of the workflow-based approach.

Categories and Subject Descriptors

I.6.7 [**Simulation and Modeling**]: Simulation Support Systems—*Environments*; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Extensibility, Enhancement*

Simutools 2012, March 19-23, Desenzano del Garda, Italy

DOI 10.4108/icst.simutools.2012.247757

General Terms

Software, Performance Comparison

Keywords

workflows, experimentation, documentation

1. INTRODUCTION

Controlling an experiment with a model (a simulation) comprises a multitude of steps. Depending on the model and the question to be answered a large parameter space (comprising thousands of combinations) might need to be examined. The computational requirements of the model and (in case of stochasticity) the number of replications might lead to a high computational effort and also might require to adjust resource scheduling over time. Traditionally, experiment control is hard-coded in modeling and simulation (M&S) software or special batch files are created for each experiment. This might lead to drawbacks as we outline in the following:

Loose of control - any implementation of the experimentation layer has to provide means to cancel the computation; to restart the computation from a certain point (e.g., in case of a power failure); to check the status of the computation; to select computing devices; to pause a computation;

Limited scalability - model development will most often be done on desktop / notebook machines whereby the computations might demand for super-computers;

Limited documentation - any experiment with a model is an experiment and thus requires documentation. As for wet-lab experiments this includes the steps carried out, the means used (software and hardware) and other factors of interest;

Reduced flexibility - changes in the process of experimentation or changes in the batch processing might require changes to the code of the software;

Some of these drawbacks can be alleviated by a careful, plug-in-based design of the experimentation layer [4] to better cope with the challenge of control, scalability, and flexibility. Another possibility is to exploit workflow management

systems for realizing the experimentation layer. Thereby, the process of experimentation and the interplay between the components becomes explicit and thus accessible.

Based on the insights gained by developing the experimentation layer of JAMES II [7] we started to migrate the software towards using a workflow orchestrated experimentation layer. Herein we describe which parts of the layer can be mapped to a workflow and what a workflow execution system must incorporate to conduct experiments. The new development is compared to the existing experimentation layer in regards to flexibility, scalability, documentation, and performance.

2. BACKGROUND

Computational science is facing a credibility problem [12, 16]. Reasons for this can be grouped into two main categories: insufficient knowledge of programmers, e.g., about numerics and software development practice, and insufficient documentation, e.g., of the software and computational experiments.

A solution for the programmer related problem is not easy to achieve and we might need a paradigm shift here first [16]. More extensive reuse can pose a partial solution here as shown in [3]. The less parts have to be implemented the less chances to incorporate bugs and the more frequent reuse the more test cases for the existing code exist and the more reliable it should be. These insights lead to the development of JAMES II, a framework for M&S providing reuse capabilities for anything from the development of new M&S software up to executing simulations on a variety of platforms [8].

However, the second category dealing with more and better documentation is more difficult to handle. Writing good documentation is usually considered to be an error prone and expensive task, something, which especially in combination with deadlines, project ends, and problems in the software development process seems not to get enough attention. Thereby, the need to document scientific processes is well known and the general need to document to achieve quality is not new either [5]. Some institutions already require a more thorough documentation, e.g., shown in [13] for M&S processes. This insight is also reflected by the ISO 9001ff[10] norms, which contain the need to provide well defined processes and corresponding process execution based documentation. Workflows (respectively workflow management systems) provide means to support this electronically: the next step(s) is/are given by a computer and the process can only be finished if the state in the workflow is advanced. Recently some first workflow systems for scientific computations have been developed, e.g., Kepler[14], Trident[1] and Taverna[9]. They allow to define a scientific process and to execute this process as often as required. However, the approaches above focus on the user defining a workflow with its different data sources and sinks, in the following, we emphasize how simulation software can benefit from using workflows internally for handling and controlling experiments.

Process models have a long history in M&S [20]. This does not hold true for user support in M&S software. Further, as already described in [19], workflows can not only be used to guide a user, and to interact with people having different roles, they can also be used to operate the M&S software as such, e.g., to steer the experimentation layer and thereby providing a number of additional features, for "free" (implementation wise). Among those are[14]:

- Data provenance
- Reliability and fault-tolerance
- Different execution schemes
- Monitoring and documentation
- Smart reruns
- Scalability
- Security (data-wise and access-wise)

One or more of them might introduce some performance penalty depending on their actual implementation and the used hardware compared to a system not providing this kind of features.

3. THE EXPERIMENTATION LAYER OF JAMES II

JAMES II is a M&S framework based on the "plug'n simulate" architecture. Therein the M&S software is described as a set of interacting extension points (e.g., the extension point "model" allows to use the framework with any model formalism). For each of these extension points any number of alternative plug-ins can be provided - a feature, which makes the system highly reusable and adaptable to a user's need.

One central part of the framework, comprising a number of extension points, is the experimentation layer. The experimentation layer of JAMES II takes care of executing an experiment with a model. Therefore it needs to provide means for computing the parameter combinations with which the model shall be computed as well as means to control each single computation (e.g., when to end a computation) and for stochastic models how many replications are to be done. The overall features of the experimentation layer of JAMES II are described in [7, 4]. Here we will focus on the subset of those, which have been transferred to workflow logic.

3.1 Parameter configuration/computation

Any computation of a model is based on a concrete set of model and computation algorithm parameters. These parameters might be configured out of a predefined set of parameters, e.g., during a parameter sweep, or the parameters might use generated trajectories fed back into their computation, e.g., during optimization. The latter case might reduce the number of parameter combinations, which can be computed in parallel. The predefined set might thereby be fully written down or it might be based on some generic rules (like explore parameter A with values from 1 to 1000 with an increment of 1).

In JAMES II, parameter configurations are computed using different layers respectively groups of parameters combined

with a modification function such as the example mentioned before. Layers consist of parameters that are changed at the same time. Each layer produces a number of parameter configurations itself, which are then combinatorial combined with the configuration produced by other layers to form a larger parameter configuration.

In detail, for a simple parameter sweep this can look like this: let's say we have a model X with parameter A and B, in which A runs from one to three with an increment of one and B runs from three to one with a decrement of one. A layer allows changing parameter values simultaneously according to a modification functions defined for each parameter. To generate a layer instance those functions are successively called. Assuming ascending and descending order as functions in the above case, the resulting layer consists of the following configurations $\{A : 1; B : 3\}$, $\{A : 2; B : 2\}$ and $\{A : 3; B : 1\}$.

Additionally we also want to compute X with two different simulation algorithms S. These are the second type of parameters, i.e., those referring to the computation of the model. Both types are handled equally by the experimentation layer of JAMES II. Let us assume: in the above example we have simulation algorithms $S1$ and $S2$ for X available and want to execute X with both of them using parameter configurations of A and B. Therefore we put the simulation algorithm parameter S into a second layer whereas S runs from $S1$ to $S2$ hence producing two configurations $\{S : S1\}$ and $\{S : S2\}$.

For the final parameter configurations each configuration of layer one and layer two are combined producing a total of 6 configurations $\{A : 1; B : 3; S : S1\}$, $\{A : 2; B : 2; S : S1\}$, $\{A : 3; B : 1; S : S1\}$, $\{A : 1; B : 3; S : S2\}$, $\{A : 2; B : 2; S : S2\}$ and $\{A : 3; B : 1; S : S2\}$ (see Figure 1).

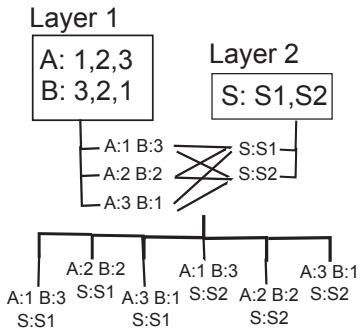

Figure 1: Illustration of the computation of parameter configurations having two layers of parameters. Layer 1: $\{A; B\}$; Layer 2: $\{S\}$. Layer 1 and 2 produce 3 and 2 configurations respectively, which are then combinatorially combined to final parameter configurations.

3.2 Job computation

The computation of a single parameter combination might mean that a number of replications has to be executed, i.e., one parameter of the computation (the random number generator instance used), is the only parameter to be changed in between two replications. All other parameters remain constant. The number of replications to be executed can either be defined by a fixed number, or more realistically, be defined by using a confidence criteria, e.g., the variance of the mean of the trajectory shall be within a certain range. Further on the duration of each single replication might be defined flexibly (fixed simulation end time; wall clock time elapsed; any math based on the values in the trajectory). In JAMES II, parameter combinations are transferred to task runners[7] as "jobs" to be executed. The task runners then have the responsibility to execute the replications as defined.

3.3 Documentation

An important feature of any experimentation layer is the documentation generated by the layer during execution. This documentation needs to contain all information required to repeat an experiment and should especially contain those information not necessarily accessible by end-users, e.g., like the version of software parts used. Currently JAMES II uses a general log file to document parts of the information required. However, the details provided are not as precise as they need to be (at least not sufficient to ensure experiments repeatability with 100% similarity). To further improve the documentation the hard-coded layer would need to be extended at many different places — at all those places where a decision, e.g., for a plug-in to be used, is made.

3.4 Execution control

Usually there exists the need to control the execution of an experiment. Control features required include start, pause, stop, and observation of the progress of the execution. Further some computations might require human interaction, e.g., in case of a face validation process.

In the current experimentation layer, these control capabilities are hard-coded "per loop". Any loop, which takes a considerable amount of computation time, e.g., the loop for the replications, needs to contain code, which allows to stop and pause the computation. This means that code needs to be duplicated and that a general error handling is rather difficult to achieve and currently not realized at all.

3.5 Parallel computation

The experimentation layer of JAMES II provides scalability on a variety of levels. That is, it supports the parallel computation of

- parameter configurations,
- replications,
- single replications,
- stop conditions, and
- data management issues

on a single computer or on a set of interconnected machines [6].

This multitude of possible parallel computations allows exploiting the parallel computing capabilities for different simulation problems on different hardware. For example, if no

replications are needed it might still be possible to compute different parameter configurations or to do the data management in parallel. In regards to the hardware the experimentation layer allows to exploit anything from multi CPU, over multi core up to multiple machines — by this the parallel computing capabilities can be adapted to the hardware available.

4. THE WORKFLOW-BASED EXPERIMENTATION LAYER

The workflow-based experimentation layer is based on the notion of workflows in JAMES II. Particularly the description of workflows herein is done using colored petri nets [11] as they form a subclass of workflow nets [22] used as internal workflow representation of the workflow system in JAMES II [18].

Aspects discussed in [18] are covered and automatically inherited for the experimentation layer. Among those are an improved documentation of the process, the support of different roles, and flexibility.

So far the implementation only covered workflows on a higher level, i.e., the general process of a study was supported but for the experiment execution the existing experimentation layer of JAMES II had been used.

Whereas this experimentation layer could be flexibly configured, the resulting internal working and interplay between the configured components has been hidden behind the interface. Thus, an explicit model of the internal functioning of the experimentation layer and the use of this model for processing experiments promises to help improving and partially automating the documentation of experiments.

4.1 Parameter configuration/computation

Part of the experimentation layer is the plug-in for configuring and computing parameter configurations. To provide compatibility with already defined experiments in JAMES II we use the existing description of those computations to derive a workflow a priori to the execution of the entire experiment. This also allows generating workflows fitted directly to the parameters to generate and therefore allows an efficient execution as well as it provides information of each of the steps taken directly. Figures 2 and 4 show two generated workflows for either two or three layers of parameters, where each layer consists of three parameters.

By representing the computation of parameter configurations this way, by explicitly defining actual parameter computations rather than just a big “black box” outputting a number of configurations, we gain a transparent computation of parameter configurations where each parameter computation is documented, traceable and can be analyzed for instance for performance issues or erroneous output. Another advantage is the flexibility it gives when dealing with input used to compute a specific parameter, which is not known beforehand but depends on the generated trajectories.

Handling feedback of trajectories back into the computation of parameters can be easily achieved by simply connecting

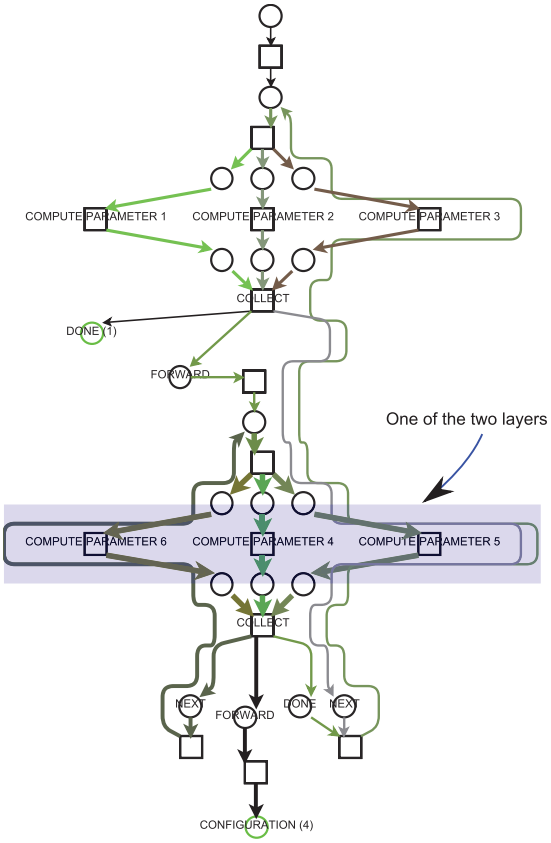

Figure 2: Example workflow representing 2 layers of parameters with 3 parameters per layer. Each layer is represented by a split-join pattern[21] allowing the parallel computation of each parameter (see Figure 3 for a automatically monitored example execution). In this example run this workflow produced 4 parameter configurations, which will be forwarded to the job computation workflow, which is omitted for this figure.

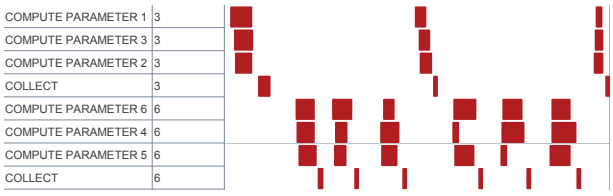

Figure 3: Monitored execution timings for the 2 layered parameter configuration computation workflow seen in Figure 2 (see the parallel execution of parameter computations per layer).

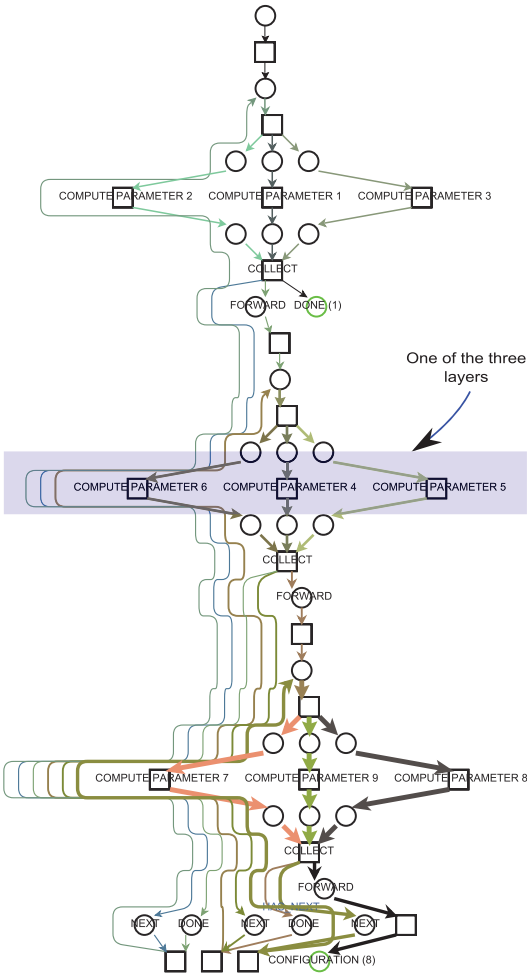

Figure 4: Example workflow representing 3 layers of parameters with 3 parameters per layer. Each layer is represented by a split-join pattern allowing the parallel computation of each parameter (see Figure 5 for a automatically monitored example execution). In this example run this workflow produced 8 parameter configurations, which will be forwarded to the job computation workflow, which is omitted for this figure.

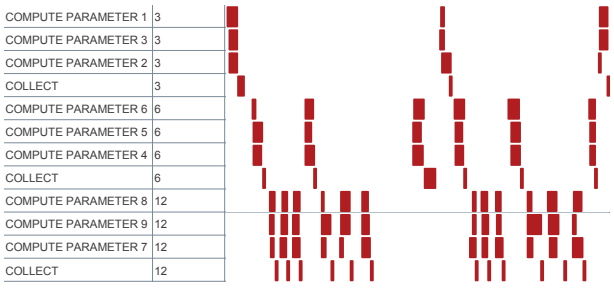

Figure 5: Monitored execution timings for the 3 layered parameter configuration computation workflow seen in Figure 4 (see the parallel execution of parameter computations per layer).

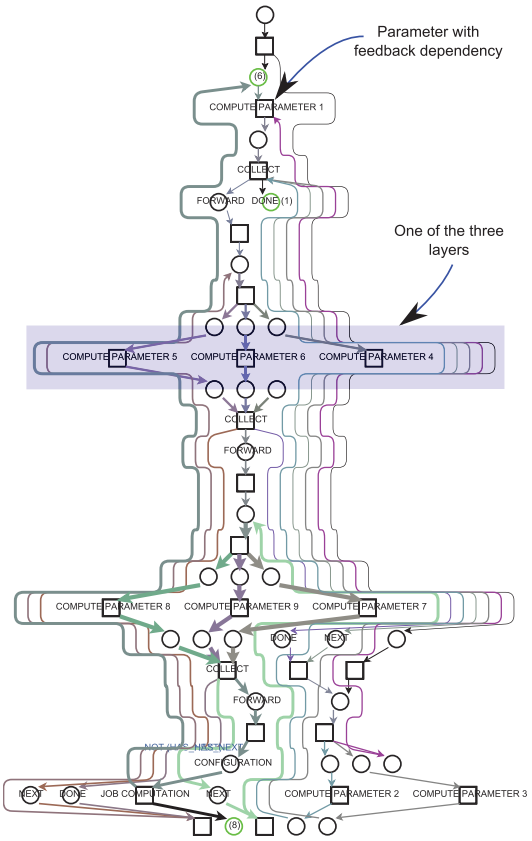

Figure 6: Workflow from Figure 4 extended to have a feedback loop incorporated feeding the computation of parameter 1. Here the computation of job from a generated parameter configuration a workflow itself is abstracted by a single workflow step to illustrate the feedback loop rather than the job computation workflow.

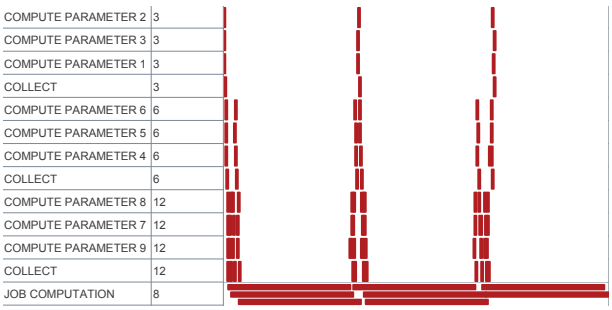

Figure 7: Monitored execution timings for the 3 layered parameter configuration computation workflow with feedback loop seen in Figure 6. Herein the job computation is also listed as the computation of parameter 1 relies on its output, to illustrate how such dependencies affect the execution of the workflow (see that the computation of parameter 1 can only happen if feedback from the job computation is available)

that information to the computation of the dependent parameter.

Figure 6 shows an example of an incorporated feedback loop for the three layered example from Figure 4 providing feedback for parameter 1. As already mentioned feedback loops and interdependencies between parameter calculations have impact on the parallelism that can be employed to compute them, e.g., Figure 7 shows that layer 1 only computes another parameter configuration once at least one job computation was finished hence feedback for parameter 1 is available. If parameters within a layer are independent, they can be computed in parallel.

Since the workflow is fine grained up to the level of individual parameter computations dependencies between parameters their input and output are clearly defined and allow for a more efficient execution of each computation by using, e.g., multiple cores for executing computations in parallel. Figures 3 and 5 show a gantt chart of the execution timings of the three layered parameter configuration computation workflow in Figures 2 and 4 respectively.

Additionally the fine-grained description allows for more fine-grained reruns starting at a specific point in the workflow without the need of executing each step up to that point again, as long as, e.g., a previous workflow execution yields that information.

As there are always two sides of a coin the fine-grained description might produce some overhead in either memory usage or computation performance or both (see Section 5.1).

However, although at first glance the workflow looks complicated and detailed, since it is derived automatically this should not pose a problem with respect to the user.

4.2 Job computation

Once the parameter configuration computation produces a configuration it will be forwarded to the job computation. As already mentioned the computation of a parameter configuration may involve executing replications. The number of replications is typically not known beforehand and depends on the chosen replication criterion, e.g., based on some confidence metric of the generated trajectories. This prevents us from generating or using a workflow, in which each replication is represented by a dedicated path within the workflow (see Figure 8). Instead we use a workflow where each replication is executed by the same workflow with each of the replications represented by a token. Thus replications can be generated and executed on demand (see Figure 10).

This allows for a simple workflow that can handle any number of replications without having a problem with an undefined number of replications or a dynamically changing amount of resources over time.

According to our workflow model, replications are executed as soon as resources are available.

The replication execution itself will be represented as subworkflow in future work as well. This way the documentation can be detailed further.

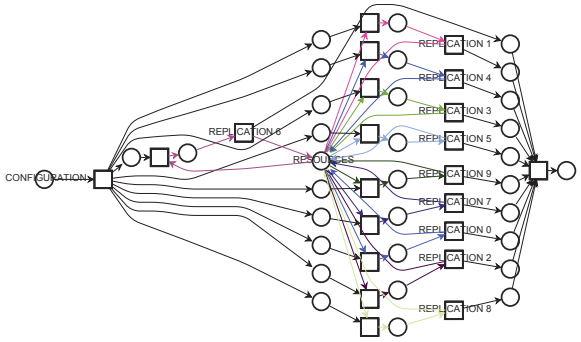

Figure 8: Job computation workflow representing each replication as separate path within the workflow. Parallelism is achieved by using a split-join workflow pattern. An additional resource place is used to assign resources, e.g., core of a multi-core CPU, to one replication.

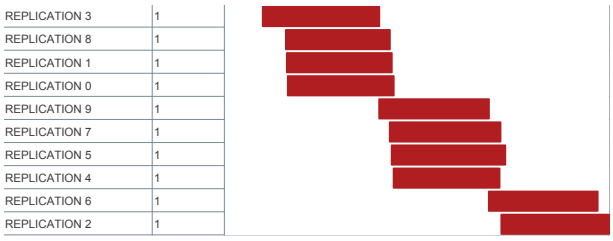

Figure 9: Monitored execution timings of an example run of the workflow depicted in Figure 8. For the example run, 4 resources (each representing one core) were used, which results in timings that show at most 4 parallel executions of replications.

Figure 10: Job computation workflow representing replications as tokens rather than explicit paths in the workflows. Parallel execution of replications is given when resources (cores) are available in the resources place and a replication token is available.

Figure 11: Monitored execution timings of an example run of the workflow depicted in Figure 10. For the example run 4 resources (ech representing one core) where used, which results in timings that show at most 4 parallel executions of replications.

4.3 Documentation

As tasks, subtasks and their order of execution are explicitly defined when specifying a workflow it allows for easy documentation of which task was executed when, what plug-ins were used, how long took the execution and what were its input and output values. Having all this information available allows us to identify bottlenecks in our workflows, performance as well as logic wise. For instance see Figure 3, 5, 9 and 11 for documented workflow step invocation count and time as well as processing time. Furthermore traces through the workflow used for a specific output can be analyzed and for instance visualized.

Figures 2 and 4, encode how often a specific edge was passed during execution in the edge's width, whereas this width is normalized according to the maximum edge width allowed in the visualization.

Additionally input and output parameters for each workflow step taken can be automatically documented if needed, which would lead to a very detailed and thorough documentation of the experiment executed, which in return would allow reproducing that experiment later based on that documentation. Note that the documentation generated does not rely on any log output generated by the experiment, simulation or one of their implementations nor does it imply changes to the existing implementations used in each workflow task as it would be in the hard-coded implementation.

4.4 Execution control

Starting, stopping, pausing workflow-based executions comes with its nature. Especially pausing and un-pausing is provided per step in the workflow - without any extra coding efforts. This can be exploited for a more advanced error management as well: if for example the hard disk is full a workflow could pause the execution; enter a sub-workflow for experimenters / administrators; inform them about the need to interact; and will automatically continue once the problem has been resolved. Managing error cases this way is straightforward and can help to retain already executed jobs and results to avoid unnecessary recalculations due to experiment restarts caused by previous errors. Depending on the workflow system's rights management is automatically provided as well, allowing expert driven experimentation and also preventing unauthorized persons from manipulating the experiment.

5. DISCUSSION

The plug-in-based architecture of JAMES II allows to have both experimentation layer implementations available and to compare these, not only theoretically but also on an experimental base. Herein we focus on the performance issues of the two alternative implementations. We concentrate on the performance as we consider scalability and resource usage to be of major importance for usability and acceptance of software dedicated to computational sciences. A short theoretical discussion is given in Section 5.2 where we discuss the pros and cons.

5.1 Performance issues

There are number of reasons for checking the performance of the two implementations ("hard-coded" experimentation layer and its workflow-based pendant).

- Parallel hardware should be equally used by both approaches.
- The software solutions should scale well with the resources available.
- No alternative should add a relatively huge overhead to the overall computation time.

The workflow-based experimentation layer thereby uses an interpretation mechanism for the workflow definition and thus might impose some overhead. We expect this overhead to be rather low but for some cases, especially for several thousands of replications of very short computations, this overhead might get significant.

To examine this we setup a performance measurement experiment based on a simple species reaction model:

```
[model]
S: S1, S2, S3; //species
R: //reactions
    r1 = 2S1 -> 1S2,
    r2 = 1S2 -> 2S1,
    r3 = 1S2 -> 1S3,
    r4 = 1S1 -> 0S1;

[parameters]
V: 1;
X_0: S1:100000, S2:0, S3:0; //initial state
Rc: r1:0.002, r2:0.5, r3:0.04, r4:1.0; //reaction
     constants
```

The model as given can be directly used in JAMES II. For our experiments it has been computed with two different computation algorithms: an implementation of the tau-leaping algorithm [2] (Simulator 1) and an implementation of the sorting direct algorithm [15] (Simulator 2). The reason for choosing two different computation algorithms is that their runtime behavior differs. The computation time of a single replication (up to simulation time 18) of the model given above using the tau-leaping algorithm is approximately $\frac{1}{10}$ of the computation time using the sorting direct method. We started all experiments using the same seed for the pseudo random number generator to avoid effects caused by the stochastic nature of the computations. The model has only been computed for a single parameter configuration (see the model definition above). Thus the parallelism stems from the need to compute replications only — consequently the experiment given here only examines one facet of the experimentation layer.

We used different test machines to get an impression of the performance characteristics. Using different machines with different parallel computation capabilities helps us to analyze the scalability of the solutions and it reduces the risk to observe architecture related effects instead of scalability issues. Following this argumentation we additionally used different operating systems (Linux and Windows) as well as different JVM architectures (32 bit and 64 bit). We thereby expect the 64 bit virtual machines to perform better as the

Machine setup	Description	Cores	OS	Java VM	Benchmark
A1	Intel Core i7 990X,	6+6	Windows 7 64 bit	JDK 1.6_29 32 bit	Score: 666.61
A2	24 GB RAM,			JDK 1.7_01 64 bit	Score: 1221.15
A3	Hyper-threading		Ubuntu 11.11	JRE 1.6_29 32 bit	Score: 1198.28
A4				JDK 1.7_01 64 bit	Score: 1257.86
B1	Intel Core i7 990X,	6+6	Windows 7 64 bit	JDK 1.6_29 32 bit	Score: 664.04
B2	24 GB RAM,			JDK 1.7_01 64 bit	Score: 1221.15
B3	Hyper-threading		Ubuntu 11.04	JRE 1.6_29 32 bit	Score: 1197.30
B4				JDK 1.7_01 64 bit	Score: 1272.57
C1	2 Intel Xeon 5690,	2 * (6+6)	Windows 7 64 bit	JDK 1.6_29 32 bit	Score: 678.97
C2	48 GB RAM,			JDK 1.7_01 64 bit	Score: 1208.65
C3	Hyper-threading		Ubuntu 11.11	JRE 1.6_29 32 bit	Score: 1180.07
C4				JDK 1.7_01 64 bit	Score: 1224.24
D1	Intel Core i7-2620M, 8GB RAM	4	Windows 7 64 bit	JDK 1.6_29 32 bit	Score: 683.94

Table 1: The machines and their software setups used for executing the performance experiments. The Benchmark column represents the composite score determined using the Scimark2 Java Suite[17] on the specified JVM as well as the $-large$ option.

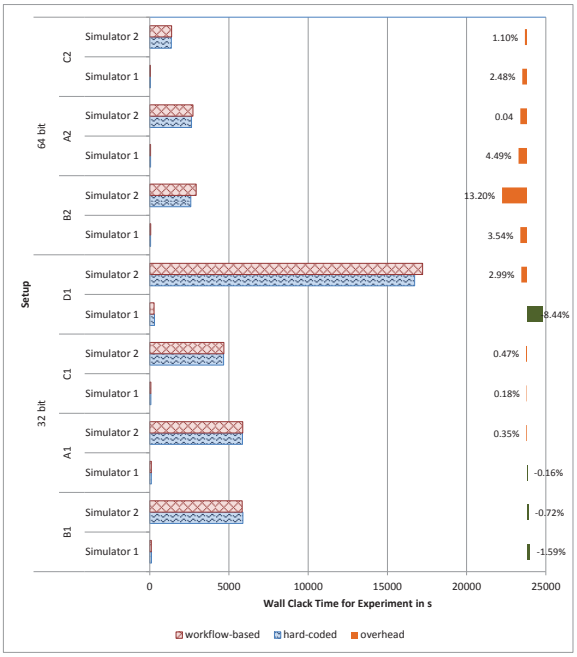

Figure 12: Comparison with parallel hard-coded and workflow-based job computation using Windows 7. Experiments were conducted on different Machines (A,B,C and D) with different JVMs (32 bit and 64 bit) (see Table 1) and different simulation algorithms (Simulator 1: tau-leaping and Simulator 2: sorted direct). All Experiments used the same model executing 100,000 replications. Also shown is the overhead introduced by the workflow-based execution on the right hand side.

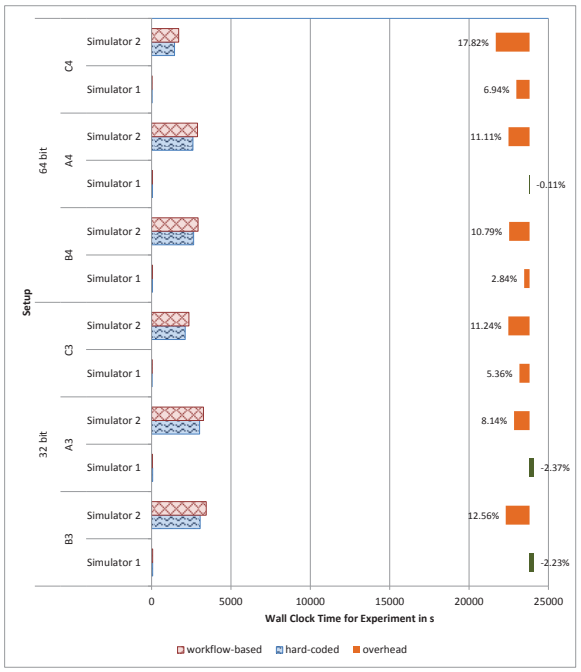

Figure 13: Comparison with parallel hard-coded and workflow-based job computation using Ubuntu Linux. Experiments were conducted on different Machines (A,B,C) with different JVMs (32 bit and 64 bit) (see Table 1) and different simulation algorithms (Simulator 1: tau-leaping and Simulator 2: sorted direct). All Experiments used the same model executing 100,000 replications. Also shown is the overhead introduced by the workflow-based execution on the right hand side.

internal data structures of the algorithms used are based on 64 bit. See Table 1 for more details on the test setups.

On all machines we used $n - 1$ cores (with n the number of cores including the hyper-threading cores) for the computation of the parallel replications. No I/O operations are conducted — the model is read once from the disk and afterwards cached for further reuse.

In Figures 12 and 13 the run time results using the hard-coded versus the workflow-based experimentation layer are given. Both figures show that there is a very small runtime overhead using the current implementation of the workflow-based experimentation layer. In some cases it even performed better than the hard-coded variant. As the average penalty produced for all experiments is low for the very short replication runtimes ($\approx 3.98\%$ per replication) it should not be recognizable for longer replications as the difference imposed remains constant.

Consequently the interpretation of the workflow seems to impose acceptable costs, especially if the expected benefits of a workflow-based computation are considered.

Another interesting observation can be derived from our experiments related to the scalability of both implementations. Considering that, e.g., Machine C has twice as many cores as Machine B with a close benchmark score per core (see Table 1) an execution time reduced by the factor two is expected for Machine C for either workflow-based or hard-coded implementation. As Figures 13 and 12 show this is indeed the case. The same applies for Machine D and Machine B where B has three times more cores and results in an execution time a third of the execution time for C for either implementation.

Further using different operating systems does not affect the 64 bit JVMs, they show very similar execution times on each machine for both operating systems. On the other hand when using a 32 bit JVM the operating system does matter. The 32 bit JVM on Windows needs almost twice as much time to execute an experiment as its 64 bit pendant. Surprisingly this does not apply for the 32 bit JVM on Linux, where it is a little slower than its 64 bit counterpart but the difference is not as significant.

Considering that we used a fairly constructed experiment not using any kind of data storage or computational expensive stopping or replication criterion the results are very promising. A real life experiment with a data storage might annihilate any difference imposed by any of the alternatives to the experimentation routines as delays by them can be used to manage and monitor the workflow without sacrificing processing time (e.g., while results are written to the disk workflow maintaining tasks, documentation and monitoring can be executed).

Further on the used workflow engine is still in development and more improvements in terms of execution speed as well as adaption and improvements of the used workflows itself can lead to further performance gain.

5.2 Pros and Cons

The pros and cons of the workflow-based experimentation layer shall be summarized briefly here. On the pro side (cf. Section 4) we already described in greater detail that we get

- a readable / visualizable execution,
- a declarative process description of experiments, which is easy to change, adapt or extend,
- documentation/monitoring of executing the experiment for free (implementation wise),
- scalability for free (multiple replications in parallel: distributed and local),
- pervasive execution control (the more processes are specified as workflows the more pervasive)
- revising experimentation processes easier (errors easier to find because they are isolatable), and
- the chance to combine workflows.

The main contra issue seems to be that the workflow system provides an additional level (the workflow needs to be interpreted, defined, monitored, and managed) and thus might introduce some slow down and memory overhead due to workflow management, complexity of and errors in workflow descriptions.

But as among the main principles of M&S repeatability of the experiments and their documentation play a central role, these effects annihilate the contra issue we have identified so far.

6. SUMMARY

Whereas workflows are typically used to guide the user, e.g., through the process of experimentation, in this paper we discussed and presented the benefits of using a workflow-based approach for realizing the experimentation layer of M&S software. These benefits lie in the improved scalability, flexibility, and documentation facilities. Our point of comparison has been the plug-in-based (but hard-coded) experimentation layer that has been in use in the modeling and simulation framework JAMES II for quite some time. In the plug-in-based design the interplay between different plug-ins has to be encoded in the plug-in responsible for orchestration. All requirements, all documentation, all changes in processing experiments are reflected in the implementation of this plug-in. Therefore, a series of these orchestrating plug-ins exist in JAMES II providing support for sequential, adaptive, and parallel execution schemes.

A workflow-based approach makes the interplay between components explicit and alleviates the need for having different implementations of execution schemes. The internal working structure of the experimentation layer becomes accessible, providing convenient means for adaptation, extensions, and documentation.

By providing more detailed documentation through the workflow system hence lessening the “black box” effect of the

hard-coded experimentation layer the quality and credibility of simulation results are greatly improved and may help to resolve the crisis of credibility.

The cost might be a computational overhead, which we analyzed in a performance study. Here it was shown that the difference is neglectable for computing parallel replications.

Also the experiments provide experimental evidence that the workflow-based approach is as scalable as the purely plug-in-based approach and the documentation facilities have been improved. To fully experience the flexibility now offered by the workflow-based approach more complex experiments, e.g., validation and optimization experiments, are needed.

7. ACKNOWLEDGMENT

This work is funded by the DFG in the project CoSA.

8. REFERENCES

[1] R. Barga, J. Jackson, N. Araujo, D. Guo, N. Gautam, and Y. Simmhan. The Trident Scientific Workflow Workbench. In *ESCIENCE '08: Proceedings of the 2008 Fourth IEEE International Conference on eScience*, pages 317–318, Washington, DC, USA, 2008. IEEE Computer Society.

[2] Y. Cao, D. T. Gillespie, and L. R. Petzold. Efficient stepsize selection for the tau-leaping simulation. *Journal of Chemical Physics*, 124:044109–144109–11, 2006.

[3] O. Dalle, J. Ribault, and J. Himmelspach. Design considerations for m&s software. In M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, editors, *Proceedings of the 2009 Winter Simulation Conference*. IEEE Computer Society, 2009.

[4] R. Ewald, J. Himmelspach, M. Jeschke, S. Leye, and A. M. Uhrmacher. Flexible experimentation in the modeling and simulation framework JAMES II–implications for computational systems biology. *Briefings in bioinformatics*, 11(3):290–300, Jan. 2010.

[5] D. A. Garvin. What does "product quality" really mean? *Sloane Management Review*, 26(1):25–43, 1984.

[6] J. Himmelspach, R. Ewald, S. Leye, and A. Uhrmacher. Enhancing the scalability of simulations by embracing multiple levels of parallelization. In *Parallel and Distributed Methods in Verification, 2010 Ninth International Workshop on, and High Performance Computational Systems Biology, Second International Workshop on*, pages 57 –66, 30 2010-oct. 1 2010.

[7] J. Himmelspach, R. Ewald, and A. M. Uhrmacher. A flexible and scalable experimentation layer. In *Proceedings of the 40th Conference on Winter Simulation*, WSC '08, pages 827–835. Winter Simulation Conference, 2008.

[8] J. Himmelspach and A. M. Uhrmacher. Plug'n Simulate. In *40th Annual Simulation Symposium (ANSS'07)*, pages 137–143. IEEE, 2007.

[9] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34(Web Server issue):729–732, 2006.

[10] ISO - International Organization for Standardization. ISO 9001:2008 - Quality management systems – Requirements, 2008.

[11] K. Jensen. Coloured Petri Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets*, volume 254 of *Lecture Notes in Computer Science*, pages 248–299. Springer, 1986.

[12] S. Kurkowski. *Credible Mobile Ad Hoc Network Simulation-Based Studies*. Phd thesis, Colorado School of Mines, 2006.

[13] A. Lehmann. Expanding the V-Modell XT for verification and validation of modelling and simulation applications. In *2008 Asia Simulation Conference - 7th International Conference on System Simulation and Scientific Computing*, pages 404–410. Asia Simulation Conference, Ieee, Oct. 2008.

[14] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, 2006.

[15] J. M. McCollum, G. D. Peterson, C. D. Cox, M. L. Simpson, and N. F. Samatova. The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior. *Computational Biology and Chemistry*, 30(1):39–49, 2006.

[16] Z. Merali. ... why scientific programming does not compute. *Nature*, 467:775–777, Oct. 2010.

[17] R. Pozo and B. Miller. SciMark 2.0, Dec. 2002.

[18] S. Rybacki, J. Himmelspach, F. Haack, and A. M. Uhrmacher. WorMS- A Framework to Support Workflows in M&S. In S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, editors, *Proceedings of the 2011 Winter Simulation Conference*, 2011.

[19] S. Rybacki, J. Himmelspach, E. Seib, and A. M. Uhrmacher. Using workflows in M&S software. In *Proceedings of the 2010 Winter Simulation Conference*, pages 535–545. IEEE, Dec. 2010.

[20] R. G. Sargent, R. E. Nance, C. M. Overstreet, S. Robinson, and J. Talbot. The simulation project life-cycle: models and realities. In *WSC '06: Proceedings of the 38th conference on Winter simulation*, pages 863–871. Winter Simulation Conference, 2006.

[21] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14:5–51, July 2003.

[22] W. M. P. van der Aalst and K. M. van Hee. *Workflow Management: Models, Methods, and Systems*, volume 1 of *MIT Press Books*. The MIT Press, 2004.