Flexible experimentation in the modeling and simulation framework JAMES II—implications for computational systems biology

Roland Ewald, Jan Himmelspach, Matthias Jeschke, Stefan Leye and Adelinde M. Uhrmacher

Submitted: 8th September 2009; Received (in revised form): 4th December 2009

Abstract
Dry-lab experimentation is being increasingly used to complement wet-lab experimentation. However, conducting dry-lab experiments is a challenging endeavor that requires the combination of diverse techniques. JAMES II, a plug-in-based open source modeling and simulation framework, facilitates the exploitation and configuration of these techniques. The different aspects that form an experiment are made explicit to facilitate repeatability and reuse. Each of those influences the performance and the quality of the simulation experiment. Common experimentation pitfalls and current challenges are discussed along the way.

Keywords: simulation experiments; experiment design; software support; stochastic simulation; computational biology

INTRODUCTION
The notion of dry-lab experiments in computational biology implies that computers, just as petri-dishes, are a feasible data source for guiding biological model building and hypothesis testing. They have some practical advantages over wet-lab experimentation: their independence of hardly controllable external factors, which may introduce bias to wet-lab data, and the possibility to do experiments that cannot be conducted in wet-labs.

Dry-lab experiments must adhere to the principles known from natural sciences, e.g. reproducibility and validity. Their achievement depends on methodological constraints and the soundness of the involved software components. They are supported by computational efficiency. All of these may lie outside the control of a typical biologist. Here statistical and computer-science expertise are required. Huge computational overhead, insignificant or invalid results, or even the dismissal of viable dry-lab experiments on the grounds of seemingly lacking computational resources might otherwise be the result. This problem is not specific to computational biology, but the abundance of alternative modeling...
and simulation (M&S) methods aggravates it—even more so in combination with the uncertainties and intricacies of applying those methods to complex biological systems that are not fully understood. One way to address this problem is to treat experiments as first-class entities (i.e. to clearly distinguish between models, simulation runs (and algorithms used) and experiments), which requires explicit support for experiment definition, adaptation and reuse.

This article focuses on describing options of the experimental layer of JAMES II to do high performance experiments, what a user can do, what the software can do and why it matters to be that versatile.

**JAMES II**

JAMES II (JAva-based Multipurpose Environment for Simulation II) is developed with an emphasis on flexibility, a feature ensured by a plug-in system [8] that distinguishes between plug-ins and plug-in types: each plug-in belongs to a specific type and each type defines a specific task the software has to fulfill, e.g. writing a model to a data sink, simulating a model or conducting a statistical test. Plug-ins are the level at which contributions from different authors or groups are combined, as all plug-in types prescribe specific interfaces that have to be implemented. Both plug-ins and plug-in types are declared in XML-files and get loaded dynamically into a central registry on start-up. This makes it easy to add new plug-ins and plug-in types. To date, JAMES II offers more than 500 plug-ins belonging to more than 80 plug-in types, although not all of them have been published yet. It is written in Java 1.6 and therefore largely platform independent. The core of JAMES II does not rely on any additional libraries, which eases deployment, reuse and installation.

The classical M&S workflow comprises three major steps: modeling, experimentation and result analysis. Since the focus of this article is on the JAMES II experimentation layer, support for modeling and result analysis is only briefly outlined in the following. In Figure 1 these three steps are outlined from a user interaction point of view.

**Modeling in JAMES II**

JAMES II supports various modeling formalisms for computational biology, e.g. specific extensions of the DEVS formalism, stochastic and spatial variants of the pi-calculus [14], reaction networks and reaction-diffusion networks. Models can be expressed in different languages, e.g. reaction networks can be formulated in our custom language, as depicted in Listing 1, or in SBML [19]. Plug-in types for reading and writing models allow adding support for various other representations. At this point, JAMES II distinguishes between symbolic and executable models: while the user may work with symbolic models (e.g. a textual representation) that are not yet complete and potentially ill-defined, such models need to be transformed into executable models (i.e. Java objects/‘compiled’ symbolic models) that can be easily accessed by simulation algorithms and whose data structures are optimized for simulation. This also makes it possible to transform the same symbolic model into different kinds of executable models.

**Figure 1:** Most user interaction takes place during modeling (step A) and the interpretation of simulation results (C). However, the experiments that are to be conducted on a model have to be specified as well (B), which in case of JAMES II entails the configuration and composition of various sub-components (red box on right). These constitute the so called experimentation layer of JAMES II and may be involved in complex interaction patterns (note the data feedback loop from the execution infrastructure to the experiment design layer).
which in turn can be simulated by different simulators: for example, a symbolic model given in SBML might be transformed into an executable model for either a stochastic simulation algorithm (SSA) (using one of the exact or approximate SSA variants available in JAMES II [7, 20]) or a deterministic simulation by numerical integration. All in all, the distinction between symbolic and executable models allows to exchange (symbolic) models easily and still retains full flexibility when it comes to the internal implementation of (executable) models. Furthermore, each symbolic model can be edited by different kinds of model editors.

Result visualization and analysis in JAMES II
Analogous to the flexible support for various modeling formalisms and languages, both off- and online visualization are supported in principle, i.e. users may plug in customized solutions in case the available components do not suit their needs. This includes statistical methods as well as visual approaches for result analysis. Apart from a pre-packaged component for line charts, which allows to view any kind of numeric variable over time, additional solutions for computational biology including a point-based visualization [21], allowing the simultaneous visualization of hierarchical models up to a million data items, a multiple-view technique for visualizing spatiotemporal data in 3D [22] and the Mosan framework for providing a linked view on model and data from the different phases [23] are under development.

EXPERIMENTS WITH JAMES II
Experimentation is a crucial and challenging feature to implement in modeling and simulation software like JAMES II. It is at this level where many—partly interdependent—techniques interact with each other (compare [25]). Imagine a user who wants to simulate the simple model from Listing 1 stochastically, e.g. to explore different reaction constants $r_1$ and $r_2$. An experiment for this task should rely on efficient mathematical methods to guide exploration, efficient statistical methods to obtain significant results and efficient algorithms to simulate the model. All these responsibilities have been assigned to specific plug-in types, i.e. we followed the common software engineering principle of separating concerns. As illustrated in Figure 1, an experiment is realized by a composition of plug-ins. JAMES II automates the execution of experiments by prescribing meaningful interaction patterns between all involved plug-in types, which eases a user’s responsibility in choosing a suitable plug-in for each type and in orchestrating them. This section outlines the core elements of a JAMES II experiment and the rationale behind them. A corresponding example is given in ‘Example: experiments with species-reaction networks’ section. Further technical details on the experimentation layer are given in [27].

Experiment design provides guidelines for efficiently conducting meaningful experiments, e.g. how to test a given hypothesis or how to find out the most interesting parameter region for a given model. Such techniques have a long history in mathematics and statistics (e.g. [28, 29]), yet results from these fields are often ignored in practical endeavors, such as clinical studies [30]. This only seems natural, since the actual target audiences of such methods are experts in their specific fields, but not necessarily in experiment design as such. Many design techniques are applicable in general and are relevant across various fields of research, so JAMES II aims at including them within its experimentation layer. This should

Listing 1: Textual representation of a simple reaction network model. See also ‘Example: experiments with species-reaction networks’ section for details.

```
1 [model]
2 species: {A}
3 reactions: {r1 = 1A \rightarrow 0A}
4 {r2 = 0A \rightarrow 1A}
5 name: {SimpleModel}
6 volume: {1.0}
7
8 [parameters]
9 state: {A:1}
10 rc: {r1:1.0, r2:2.0}
```
empower non-expert users to design more sophisticated experimental setups by themselves, without having to re-implement these techniques manually. JAMES II supports different families of techniques, as described in the following. Thereby, the specification of the experiment is crucial. We distinguish model and simulation execution parameters. Model parameters define the initial state (including start parameters, composition, etc.) of the model to be executed. Simulation parameters define execution-related aspects as data collection means and simulation run execution techniques. JAMES II provides additional assistance for both parameter sets. The parameters and the manual or automated combinations of techniques influence the performance of experiment execution. Here performance can refer to run time/memory efficiency or to quality of results.

Model parameters
Model parameters are parameters used to parameterize/setup a model. The parameter combinations to be used for one dry-lab experiment can be given using fixed sets or they can be dynamically computed. The effect of model parameters on efficiency is obvious, as useless or faulty parameter combinations can add a bias to the result and they consume unnecessary computation time.

Parameter scanning
A simple parameter scanning experiment, i.e. simulating a fixed set of model parameter setups, is defined in JAMES II by nested lists of experiment variables and their modifiers. Each variable corresponds to a model parameter that shall be changed over the course of the experiment. Its modifier updates the value of the variable during the experiment, e.g. step-wise or following a pre-defined list of values. To allow for all kinds of parameter combinations, an experiment variable can be either added to an already existing list or to a new sub-list, as shown in Figure 2.

Optimization, validation, sensitivity analysis and meta modeling
Another relatively common task for simulation software is optimization, i.e. to find a parameter setup that maximizes an objective function defined on model outputs and parameters. The more parameters a (biological) model has, the larger the search space for optimization techniques.

Several common meta-heuristics [32] have been implemented for optimization in JAMES II, e.g. simulated annealing, genetic algorithms and tabu search. They are broadly applicable since they treat simulation models as black boxes. More powerful methods can be integrated by wrapping them in a JAMES II plug-in.

Figure 3 shows how optimization algorithms are combined with experiment variables to define an optimization experiment. Note that sub-lists are not allowed, as this would mean to let the optimizer examine a single setup that actually represents multiple ones. Additionally, a user needs to define an experiment variable that holds a list of all optimization algorithms to be executed. These will be independently applied to the problem, one after another. This, for example, allows to define multiple optimization strategies for parameter fitting, and compares
their performance for the task at hand. By defining additional layers on top (compare Figure 3), such jobs can be performed on a whole range of different setups.

One of the most crucial aspects for successful experimentation is model validation. Elaborate classifications of validation methodology exist (e.g. [33]), but supporting experimental model validation in practice is still very challenging [34]. Validation makes use of several other experiment design techniques, e.g. sensitivity analysis (SA): if parameter interactions and impact in the real system are known such analyses can invalidate all models which fail to comply. Similar to optimization algorithms, validation methods often rely on external data sources, e.g. results from the wet lab.

A user should always do a model parameter SA at first. SA can be used to identify the important and the valid domains of parameters. Thus, SA allows restricting the search space to a few key parameters and value ranges. JAMES II currently supports SA with full factorial designs, fractional factorials [31] and Plackett-Burman designs [28].

Another domain of experiment design is concerned with the efficient generation of so-called meta-models. A meta-model approximates the behavior of a simulation model. It is usually defined mathematically, e.g. as a polynomial function of the model parameters, for which some coefficients are unknown and have to be fitted to the behavior of the original model. Typical meta-models are relatively easy to compute, so that they can save large amounts of computing time, e.g. when used by optimization algorithms. Moreover, they can be used to steer experiments into those regions of the parameter space that will yield more (statistical) information [35]. JAMES II already supports deterministic kriging (compare [35, 36]). We regard meta-modeling as an interesting future direction, as it allows reducing the number of simulation runs needed significantly. Technically, meta-modeling is implemented similarly to optimization and sensitivity analysis.

**Simulation parameters**

While the model parameter set creation techniques discussed above define what experiments to execute with the model, they do not prescribe how, how long and how many times runs shall be executed. Simulation parameters have a direct impact on performance. Depending on the simulation algorithm execution paradigm, i.e. sequential or parallel, due to different simulation algorithms [20] and combinations with other algorithms, the time needed to execute a simulation run can vary significantly. The accuracy, e.g. due to different numerical integration algorithms or exact or approximate Gillespie variants can be traded for efficiency as well. Last but not least statistical efficiency, which aims at obtaining a result of maximal statistical significance with minimal consumption of computational resources, has a significant impact. With increasing popularity of stochastic simulation approaches in computational biology, the latter mechanisms become more and more important.

**Stopping rules**

Apart from stopping at a pre-defined point in simulation time, JAMES II supports dynamic stopping rules, e.g. for configuring any simulator to stop after a certain number of simulation events, a certain amount of wall-clock time, or after an observed variable reached equilibrium or a given threshold. Additional plug-ins allow to concatenate arbitrary stopping rules using AND and OR operators. There is a large body of statistical work on stopping rules, which are of particular importance for steady-state simulations, e.g. [37]. This knowledge should be leveraged for experiments on biological models as well, since a suitable stopping rule avoids unnecessary computation and may therefore reduce execution time considerably.

**Replications**

Outcomes from simulation runs that contain stochastic elements have to be regarded as random variables. These variables need to be sampled repeatedly in order to gain significant results, which again require statistical methods. In JAMES II, replication is controlled by a list of replication criteria. They determine how many times each parameter setup of an experiment has to be replicated. The replication of a parameter setup is not stopped until all criteria are satisfied. A typical application of replication criteria is to ensure certain confidence intervals for some simulation output.

As too many replications waste computation time and too few replications yield unreliable results, JAMES II allows using replication criteria to end up with the minimal number of required runs.

Furthermore, a proper handling of random numbers is essential for any tool that offers stochastic simulation. The (pseudo)-random numbers are
generated by specialized random number generators (RNGs) and have to be as uncorrelated, i.e. random, as possible. If this aspect is neglected, which is often the case in experimental simulation studies [5], this can lead to strongly biased and invalid observations [38–40]. The initial state of an RNG, also called its seed, predetermines its output and has to be defined within the experiment. Otherwise, the results obtained lack reproducibility. Since it is extremely hard to determine the best RNG for a given problem a priori, stochastic experiments that strongly rely on random numbers should be executed with different kinds of RNGs which produce unequally correlated output [39]—if results still match, chances are good that the obtained data is valid.

JAMES II provides a framework to integrate existing RNGs, along with custom implementations of the most popular algorithms and various established statistical tests to check the randomness of new RNG plug-ins [41]. Since most RNGs only generate numbers that are uniformly distributed in [0, 1), JAMES II offers over 20 commonplace probability distributions that transform the output of RNGs appropriately.

Recording simulation results
After deciding which parameter setups are evaluated (Model Parameters Section) in what manner (Stopping Rules and Replications Sections), a JAMES II user has to select which results of a simulation run need to be recorded, i.e. the data that shall be observed from each run. This is an important part of the experimentation process, as selecting insufficient data will hamper later result analysis. For example, infrequent snapshots of the model state may miss potentially interesting phenomena that occurred in-between. On the other hand, even relatively simple experiments with chemical reaction networks may generate overwhelming amounts of observations, as discussed in [20]. In JAMES II, simulation trajectories are recorded using a plug-in type which hides the actual storage technology. This allows to alternatively use data base-, file system- or memory-based storage approaches.

Instrumentation
Instrumenters in JAMES II are responsible for selecting the properties of simulation algorithms and models that shall be recorded. This procedure is straightforward in most cases, but can be challenging when only specific parts of a large model are of interest. The trade-off between experiment duration and storage space on one side, and the amount of required measurements on the other, ultimately depends on the objectives of the experiment. This is decided by the JAMES II user, who configures the chosen instrumenter accordingly. The instrumenter and its configuration are also part of the experiment definition, since the reproducibility of the obtained measurements cannot be guaranteed otherwise. It is invoked once, after model and simulation run instantiation. Models with dynamic structures, such as some of our DEVS extensions [10, 42], pose future challenges for instrumentation in JAMES II, because new model entities might be added at runtime and thus need to be instrumented on the fly.

Observation
JAMES II relies on the well-known observer pattern [43] to take measurements during simulation. An observer is registered by an instrumenter at one or more entities that may belong to model or simulation algorithm. It gets notified when the state of the observed entity has changed. Different interfaces mark observers that require a data storage to write to or that may feed an online visualization. So far, observation code for JAMES II has to be programmed manually, but it can often be reused across similar experiments, e.g. when they rely on the same modeling formalism.

Being able to integrate custom observation code is important because a naive solution might be computationally expensive and would then slow down the overall experiment. Figure 4 illustrates the potential impact of observation on simulation performance.

Simulation Run Execution
Finally, a JAMES II user has to specify the way in which the model setups are to be simulated, i.e. with which algorithm and on which resources. Defining the algorithms to be employed is crucial for reproducibility and comparability of simulation results. Simulators are non-monolithic entities in JAMES II, as they often rely on other plug-ins, such as event queues or numerical integrators. All plug-ins can be exchanged by alternative implementations of the same plug-in type. This leads to a combinatorial explosion that allows JAMES II to offer thousands of distinct simulation algorithm setups in some cases.

Algorithm selection
Due to the flexibility of JAMES II users can be overwhelmed by the large amount of options, and it is often unclear which setup will perform best for a given model. Assisting the user in
making this decision relates to the so-called algorithm selection problem [45]. In JAMES II, it has been tackled in different ways, e.g. by using machine learning for an automated selection before execution [46] or by using sequential experiment designs to identify the most suitable setup for replication at runtime [47]. The latter can save up to 60% execution time in experiments with our SSAs and stochastic-pi simulators.

Distributed execution schemes Further reduction of execution time can be achieved by distributing the load over multiple resources. Such parallelization makes sense on several levels: experiment design, replications or even for speeding up a single run. JAMES II provides parallelization mechanisms for each of them. On the level of experiment design, it allows to create an arbitrary number of parameter setups at once, so that these can be simulated in parallel. This suffices for many sensitivity analysis techniques (see Model Parameters section), which generate all parameter setups at the beginning and require simulation results only for identifying parameter interactions at the end. The situation is different for optimization and meta-modeling. Here, feedback in form of observed simulation data is required to steer experimentation itself, so the generation of parameter setups needs to be throttled. Techniques for parallel optimization may be employed to alleviate this effect.

On the level of replications, each criterion returns the minimum number of additional replications it requires, rather than a yes/no answer (see Replications Section). This enables parallelization for a single parameter setup, since all replications that are already known to be required can be executed in parallel.

These two levels of parallelization are managed by the simulation runner plug-in type [48], which may use local resources or connect to a master server for job scheduling. The master server manages simulation servers, which act as workers and execute the simulation runs distributed to them. The master server also manages additional services that may be necessary for distributed execution, e.g. databases to store simulation results.

Finally, very large or very complex models may require a fine-grained parallel and distributed simulation, i.e. parallelization on the level of a single run that is distributed over multiple resources [49]. While this powerful technique is very promising in the advent of multi-core CPUs in standard desktops, it also poses some considerable implementation challenges.

Two major problems are the partitioning of the model and the synchronization between the resources. The partitioning has to be chosen in such a way that neither computational load nor communication costs hamper performance dramatically. Finding a good balance between computation on the resources and communication between them is crucial for the performance of fine-grained simulations. In order to realize fine-grained synchronization schemes, specialized (conservative or optimistic) simulation engines are required. If such simulators are available for a modeling formalism, as it is the case for some DEVS and pi-calculus variants in JAMES II, the experiment can be configured with a resource allocator, a component that selects the desired number of servers for executing a single run. Several formalism-independent partitioning schemes can be used to assign model entities to available resources. The list of eligible resources is provided by the master server. Both master and simulation servers can be started and administered via the user interface. Figure 5 summarizes the possibilities for parallel and distributed simulation in JAMES II.

Experiment Storage and Retrieval

All the above parameters constitute the specification of the dry-lab experiment. A custom XML-based file format is used for storing and retrieving experimental setups. Reading and writing of experiment
descriptions is based on plug-ins and can therefore be extended easily.

EXAMPLE
A small example shall briefly demonstrate how a user can set up a simple experiment by combining some of the JAMES II components introduced in the previous sections. The task for the user should be to create an experiment that estimates the kinetic reaction constants for two reactions. Our toy model (compare Listing 1) represents a very small reaction network with only one species and two reactions; one reaction increases the number of particles for the species, while the other decreases it. Obviously, the system will converge toward a steady state.

The reaction constants \( r_1 \) and \( r_2 \) shall now be estimated, based on a known amount of species A in the steady state. Listing 2 shows the corresponding experiment definition. Here, the user chooses to perform a stochastic simulation of the model with a specific SSA, (lines 43–45). The core of the experiment, i.e. finding the values for the missing parameters, is defined in lines 9–16. A single simulation consists of 10 independent replications (line 25) that are executed in parallel (line 47) and utilizes the Mersenne Twister for generating random numbers (lines 26–28). Snapshots of the model state are taken at three different time points and are stored in a file (lines 31–39).

RELATED WORK
Not only is the number of M&S methods being increased steadily, but also the number of M&S software packages for computational biology. Thus, the following discussion of related work remains necessarily incomplete. JAMES II is of course not the only M&S software that supports the experimentation techniques discussed above.

In systems biology, many tools support different simulation algorithms, e.g. Copasi [51], E-Cell [52] and Dizzy [53]. Those tools typically focus on one family of formalisms though, e.g. reaction networks (Copasi and Dizzy). Copasi (V 4.5 (Build 30)) provides support for a number of additional techniques (e.g. optimization, parameter scans, etc.), but to our understanding they cannot be combined easily.

The Bio-PEPA Workbench [54] offers many options when it comes to simulating and analyzing a model defined in Bio-PEPA. It does this by using other tools such as Dizzy for simulation or PRISM for probabilistic model checking. Dizzy itself offers a variety of simulation algorithms, but it does not seem to support more complex experiment setups. The need to accommodate other formalisms is addressed in Bio-PEPA by exchange formats, which allow models defined in other formalisms to be translated into Bio-PEPA (as long as they do not have a different expressiveness). In the CoSBi Lab [55], a collection of tools for visualization, analysis and simulation is built around the language BlenX. Both the Bio-PEPA Workbench and the CoSBi Lab adopt the idea of the Systems Biology Workbench [56], i.e. supporting the integration of different tools. Cell Illustrator 4.0 [57] provides a variety of different model descriptions, which are then mapped to a single internal model representation. It supports graphical model editing and runtime visualization via the main model editor. In contrast to JAMES II, the experiment setup is restricted to parameter scans, external servers and thus coarse-grained parallel simulations can only be configured manually, per simulation run. Many other products from simulation research address specific sub-topics that have been raised, e.g. random numbers [58] or instrumentation [26].

Finally, we observe a trend toward further automation for wet-lab experiments [59] and dry-lab experiments [60] alike. All this documents the need for flexible tools that support different approaches to model, simulate and analyze. However, the referenced approaches differ from JAMES II as their support of alternatives, i.e. their flexibility, typically ends at the level of simulation algorithms or entire tools. This is unsatisfying with respect to the quality of...

---

**Figure 5:** Different abstraction levels for parallelization with JAMES II. Note that setups and their replications can be executed independently of each other, while the parallelization of a single run implies the communication of intermediate results (e.g. via timestamped messages) among the participating entities (blue arrows) and hence requires synchronization.
results and performance, as details like data structures and sub-algorithms can have a significant impact [20]. Configuring single simulation runs in detail (algorithms, sub-algorithms, model data structures, etc.) is a unique feature of JAMES II. The thorough application of JAMES II’s plug’n simulate concept also provides the basis to configure and steer experiments automatically, e.g. with the help of a performance data base or machine learning methods.

CONCLUSION
Many advantages of using JAMES II for computational biology research have already been discussed [7], albeit from a more technical and less experiment-centric and comprehensive viewpoint.

Here we have illustrated the usage of JAMES II as a general tool for high performance dry-lab experimentation. We have discussed several important issues any experimenter has to take care of: general experiment design, statistical efficiency, measurement data processing and execution specifics. The fundamental requirements of reproducibility, software component soundness and efficient execution pervade all of the above aspects. They are not just orthogonal, but also determined by their weakest link: one under-defined mechanism and an experiment may not be reproducible; one malfunctioning algorithm and its outcomes may be invalid; one slow implementation and the computation may take ages.

While the diversity of techniques makes dry-lab experimentation a daunting challenge, it also hints at
the huge potential in combining different methods, e.g. to reduce the number of required parameter setups, the number of replications per parameter setup, the execution time per replication and so on. These synergies can be leveraged by using a common platform, which allows to automate the overall process and to integrate research efforts from all relevant areas. We hope that JAMES II provides a solid base for such developments. Another important direction of future work aims at facilitating the definition of JAMES II experiments, e.g. by developing comprehensive experiment editors, which should make it easier for non-computer scientists to exploit the flexibility and to come up with performant and valuable experiments. A version of JAMES II is available at http://jamesii.org.

**Key Points**

- We illustrate the usage of the modeling and simulation framework JAMES II as a tool for dry-lab experimentation in systems biology.
- Important issues in dry-lab experimentation are general experiment design, statistical efficiency, data processing, and execution specifics. Thereby, fundamental requirements to be addressed are reproducibility, software component soundness, and efficient execution.
- The diversity of methods for efficient dry-lab experimentation in systems biology is challenging, but also provides huge potential, e.g. when combining those to reduce the number of required parameter setups, the number of replications per parameter setup, and the execution time per replication.
- This potential can be exploited effectively by a plug-in-based modeling and simulation framework like James II, which allows to integrate research efforts from diverse areas and to automate parts of the experimentation process.

**Acknowledgements**

The authors would like to thank the anonymous reviewers for their constructive comments.

**FUNDING**

Deutsche Forschungsgemeinschaft (DFG; German Research Foundation).

**References**

22. Unger A, Gutzeit E, Jeschke M, Schumann H. VioNeS - visual support for the analysis of the next sub-volume


