ABSTRACT
JAMES II is a modeling and simulation framework designed to ease the creation of specialized M&S applications, to experiment with models, and to ease the experimentation with alternative data structures and computation algorithms. The flexibility of the framework is based on the plug’n simulate architecture which allows to have any number of alternatives coexisting, from modeling means over computation algorithms to experiment control and analysis, within a single software. Herein we show how alternatives are added to JAMES II, how JAMES II can be used to efficiently execute simulations, what can be reused to built specialized M&S software, and how this can be used to do a more fair comparison of the alternatives.

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

General Terms
Software, Performance Comparison

Keywords
framework, plug-ins, experimentation, reuse

1. INTRODUCTION
The plug’n simulate architecture [8] has been developed to overcome diverse problems of software for the computational sciences as described in [16, 11, 18, 15]. It allows to share code over modeling means and application fields and supports, due to the strict separation of concerns, experiments with data structures and algorithms.

JAMES II (Java Based Multipurpose Environment for Modeling and Simulation) has been created based on this architecture and is being under development from 2003 on.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Simutools 2012, March 19-23, Desenzano del Garda, Italy
Copyright © 2012 ICST 978-1-936968-47-3
DOI 10.4108/icst.simutools.2012.247768

More than 40 people have contributed, almost half a million lines of code have been produced, and JAMES II has been used in such diverse application fields as systems biology [22, 14], demography [25, 24], e-learning [17] and network simulation [19]. Therefore modeling means ranging from variants of DEVS [23], reaction networks [9], process algebra [10], multi level rule based modeling [13], multi agents [5], cellular automata [21] up to component based modeling [5] have been added to the framework and used for the applications. Experiments with algorithms have been made so far for DEVS [6, 7], reaction networks [9] and cellular automata [20]. The architecture is the base for further advanced research on M&S software, like using workflows for controlling the execution of experiments [21], the automatic selection of algorithms to be used to solve the job at hand with a good performance [2, 1] and work on parallelization of computations [3].

Working on all these diverse challenges was thereby eased by the fact that each single challenge has been resolved by using developments made to solve the others. For example, experimentation control has been widely reused by all the applications.

Herein we’ll show first how JAMES II can be extended by using a computation algorithm for a reaction network as an example (cf Section 2). A simulation is defined and executed in the second part (cf Section 3). In Section 4 a specialized application is shown which uses JAMES II in its backend. Section 5 gives a short introduction on how JAMES II can be used for experimental algorithmics. In summary we will demonstrate the flexibility and potential of the architecture, and which benefits arise if a software built on such an architecture is consequently used for almost a decade.

2. EXTENDING JAMES II
The plug’n simulate architecture and thus JAMES II provides extendability on two levels

- plug-in types and
- plug-ins.

The first category defines types of plug-ins (an interface any plug-in has to implement which shall be available through this extension point) and the second category are the alternative implementations for the plug-in type (e.g., different random number generators). The plug-in schema has
been realized using the abstract factory and the factory patterns. Thereby the abstract factory has to filter the plug-ins available according to user selections and compatibility issues (e.g., selecting a computation algorithm for cellular automata if such a model shall be computed and not one for DEVS etc.) [8]. Each plug-in type description comprises an xml file defining the type, an AbstractFactory, and a base factory for creating the instances the plug-in factories have to inherit from, and the interface definition which has to be implemented by the plug-ins. If no complicated filtering is required the main effort here is in the definition of the interface. Each plug-in comprises the class implementing the interface, a factory class inheriting from the base factory to create the instance of the plug-in and an xml file describing the plug-in. Plug-in types and plug-ins are automatically loaded by JAMES II and thus it is possible to extend JAMES II without any need to modify existing code.

3. SIMULATION WITH JAMES II
JAMES II provides a flexible and scalable experimentation layer which supports users on doing experiments with their models. The experimentation layer of JAMES II [4] allows to setup any M&S experiment from brute force parameter scans up to optimization and validation experiments. Thereby the control means required for computing single replications (e.g., number of replications, stop conditions, etc...) are realized as plug-in types and required alternatives can be added by everyone. A special emphasis is laid on how JAMES II tries to exploit the hardware available (multi core/multi cpu/multiple machines) [12, 3]. JAMES II provides a class named BaseExperiment which allows to setup all the parameters required. The most simple setup looks like

```
BaseExperiment experiment = new BaseExperiment();
experiment.setModel (path to your model);
```

This will setup a simulation with the model using default parameters. It can be started by calling the execute method of the experiment instance).

4. BUILDING A SPECIALIZED M&S APPLICATION
JAMES II has been used in the project MicMac, funded by the European union, to build a specialized M&S application to be used throughout statistical offices all over Europe [25]. For that purpose first a novel modeling formalism for describing of a demographic micro model based on empirical data has been added to JAMES II. Together with a newly created computation algorithm for the modeling means, which has been built based on additional plug-in types in the framework, e.g., event sets and random numbers, no further extensions nor any changes had to be done. The application has an own graphical user interface which allows to load a model and to start a simulation. This interface has been developed without using the user interface framework shipping with JAMES II and it just calls the experimentation control elements of JAMES II, parameterized with the model to be computed, and then “waits” for the framework to finish the computation. Using JAMES II has helped a lot on developing the application. Using plug-ins instead of hard-coded functionality experiments on the impact of implementation alternatives have been achievable and thus it was possible to figure out an efficient setup to compute the models on desktop computers in less than a minute. Currently the application does not make use of a number of additional features provided by the experimentation layer: neither the internal mechanisms for parameter scanning, optimization nor the support for parallel computations are used. But if any of those things are required in the future it will not require a lot of implementation work (if any) to make use of these. Further on, bug fixes, performance improvements etc... applied to the code reused are automatically available in the next build of the specialized application - which reduces maintenance effort and costs.

5. EXPERIMENTS WITH ALGORITHMS
Good experiments with algorithms are hard to achieve. According to literature about experimental algorithmics a number of preconditions need to be fulfilled if such experiments shall really reveal insights. Among those fair comparisons of the competing algorithms (something which may be hard to achieve in case that all alternatives are implemented by the one who wants to show that his algorithm performs better), a fair testing environment (only the algorithm under test should be evaluated and not an unknown combination of data structures and additional algorithms in the test setup), and fair problem instances [11]. The plug’n simulate concept allows to provide any number of alternative algorithms for an extension point, and thus allows any number of, e.g., computation algorithms for a modeling means. Any of those can be parameterized with additional plug-in types, e.g., event sets. If an algorithm is added it can be compared to those already available for an extension point (reducing the “my code shall perform better” bias). Additionally, if someone doubts the results of the comparison he can add his own alternative implementations of the competitive algorithms, even if implementations of the algorithms already exist, and repeat the experiments. The separation of concerns helps to avoid or at least reduce side effects. Further on JAMES II provides some tools, developed for research on the automatic selection of algorithms, which support experiments with algorithms [1].

6. SUMMARY
This demo shows how JAMES II can be used for diverse tasks in research and application in the computational sciences. According to our experience using JAMES II consequently for the research questions in our group had a number of important advantages: implementations of core parts (e.g., plug-in management, experimentation control) have been widely reused and thus tested, implementations of formalisms and computations algorithms have been used for applications and for experimental algorithmics, and we have been able to exploit the implementations done for further research projects like the automatic selection of algorithms. The software created based on the plug’n simulate architecture is available as open source (http://www.jamesii.org), containing examples for most of the aforementioned uses.

7. ACKNOWLEDGMENT
This work has been mostly funded by the DFG in the projects CoSA, dimersoi, and the research training group dIEM oSiriS.\(^1\) accessed, January 2012
The work done described here has been done and supported by more than 40 colleagues and students for what I’m very thankful.

8. REFERENCES


