**Jacek STARZYŃSKI, Robert SZMURŁO, Bartosz CHABER, Stanisław WINCENCIAK**

Warsaw University of Technology

# Flexible Scenarios Based System for Scientific Computing

***Abstract.*** *Cloud computation technologies open a new perspective for scientific computing. Sophisticated software can now be made available as on-demand service, reducing costs and broadening accessibility for end-users. The paper presents an implementation of a system serving open source electromagnetic field simulation software as a web based service. The primary users of the system are medical staff and electromagnetic safety engineers which are usually not familiar with field simulation methods. The user friendliness of a system combined with its flexibility, scalability and extendibility is realized with help of carefully designed concept of configurable usage-scenarios. Our aim is to present this scenarios concept in detail.*

***Streszczenie.*** *Technologia obliczeń w chmurze otwiera nowe perspektywy w zakresie symulacji naukowych. Zaawansowane systemy symulacyjne mogą być z jej pomocą udostępniane w postaci dostępnych na żądanie usług, co zmniejsza koszty i zwiększa możliwość stosowania przez użytkowników końcowych. W artykule przedstawiono implementację systemu udostępniającego narzędzia do numeryczenj symulacji pola elektromagnetycznego lekarzom i inżynierom zajmującym się problematyką bezpieczeństwa i higieny pracy, którzy nie są ekspertami w dziedzinie symulacji komputerowych. Przyjazność systemu dla użytkownika została połączona z elestycznością konfiguracji, skalowalnością i rozszerzalnością przez zastosowanie konfigurowalnych scenariuszy użycia. Celem autorów jest szczegółowa prezentacja tej koncepcji.* ***(System obliczeń naukowych oparty o elastyczne scenariusze użycia)***

**Keywords:** cloud computing, computer system architecture, electromagnetic fields
**Słowa kluczowe:** obliczenia w chmurze, architektura systemów komputerowych, pole elektromagnetyczne

## Introduction

Great development of hardware and software dedicated for medical application creates new opportunities for medical diagnostics. A doctor can now "look inside" the virtual patient and prepare more effective and less invasive treatment. In this paper we focus on computer simulation in bioelectromagnetics. Electric and magnetic stimulation of human body are nowadays commonly used in wounds healing, rehabilitation and psychiatry. Use of computer models would allow to improve these treatments, especially if simulation could be carried with individualized models of patient body-parts. The current state of art in bioelectromagnetic modeling allows such simulations, but the application of it is limited by high demands for users. Modern scientific computing requires many sophisticated software tools combined in heterogeneous systems. Commercial scientific computation environments are expensive and their capabilities can often be to narrow for a researcher. Open source systems allow more freedom but at cost of complicated and time consuming configuration and maintenance.

Several systems can be used for numerical field simulation [2]. From an end-user point of view all of them have one in common: they are complicated and require a lot of time for mastering. Any system requires specialized knowledge in the subject of simulation and at least some general orientation in the principle of software internal algorithms and methods. A first time user of any such system faces a high barrier of knowledge which must be acquired before the first useful results of simulations are obtained. Moreover a single system if rarely sufficient. To prepare, carry through and evaluate a scientific computation process one usually needs several "processors" which need to be stitched together, what makes the mentioned barrier even higher.

Recently a cloud computation technologies open a new perspective for scientific computing. Sophisticated software can now be made available as on-demand service, reducing costs and broadening accessibility for users. The goal of the authors', described in detail in [3], is to develop a system which will not build any barrier even for the first time user. In fact it is nothing new in the Internet: most of it's users face every day a lot of systems which can be used without any preparation. Starting from web browsers, through e-mail hosts, e-shopping up to the grid computing more and more advanced services are offered.

However, scientific software is much more challenging. It's subject is not so obvious, it's results are more difficult to evaluate and the way from the input to the result is much longer and more complicated comparing to an e-mail client or Internet shopping.

For example the typical electromagnetic simulation environment involves specific engineering skills. Thus, such system can not be explicitly used by medical staff who desires different interface, focused to their needs. The wide gap between those two should be effectively shortened. From the designer point of view the system should be flexible and extensible and for the end user it must be easy.

The authors idea to achieve this goals is to use configurable scenarios. This idea is not new, it was probably invented for engineering by an architect—Christopher Alexander [1] and it is commonly used in nowadays software engineering and "software patterns". In the presented work the pattern-oriented approach allows effectively combine easiness of the system seen from the users' point of view and its internal complicated structure which is expected to evolve continuously to meet new needs and use new possibilities.

The typical scenario contains a series of steps which are usually performed by the end user to obtain the desired result, for example compute the power absorbed by the human body during the electromagnetic field exposition. The scenarios are configured by specialized technicians, but execution of scenarios is available to domain staff which is not required to have the knowledge about simulation methods. The user interface, which major goal is to be as easy to use as possible, can be a web application and has been presented by the authors in [3].

The presented system's architecture is divided into three layers. The client interface layer which is responsible for graphical presentation of results and this can be any application which is capable of communication using web services. The second layer is a central system which is called the Coordinating system. The purpose of this layer is to provide a common way to serve the computational resources to the web clients. This part of the system contains a scenario execution environment (Scenario handler) and a set of specialized *plugins* which can be invoked from scenarios, each for a specific computational system or a program. Plugins cooperate with programs and simulation systems (called here *processors*) which build the third layer.

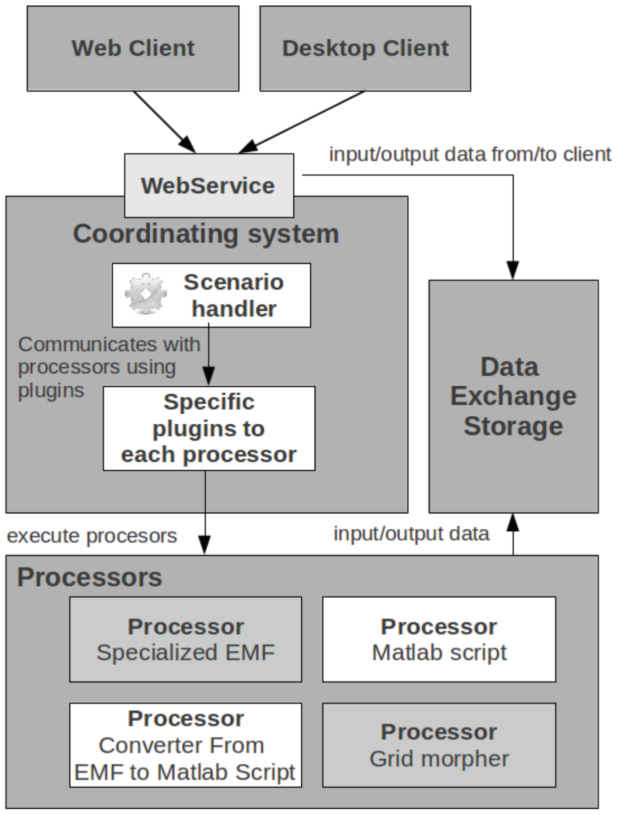


Fig. 1. System architecture concept.

## Concept of flexible scenarios

Considering development of scenario-based system, one have to face the challenge of data and control flow implementation. In this work, the authors explored a solution based on data dependencies between steps which allowed them to develop very flexible scenarios. Using proposed approach a user is able to choose from available steps. As well he or she can take steps proposed by the system and follow the main scenario path.

The main idea behind described scenario-based system implementation is to remove need of explicit transitions between steps. In fact, steps does not depend on each other, but they depend on *data* produced and required by them.

To simplify flow of data between steps in scenario the need of data centralization has emerged. Every running scenario (called *scenario instance*) has its own result's repository which is accessible by every step.

To better understand proposed method, few definition have to be established:

**Processor** Independent data processing unit, used to be run at an operating-system level. It can be implemented in any available technology. It has access to hardware.

**Plugin** Java class which is accessible for system described in this paper. This class is used as a connector between **processor** and Java—written system.

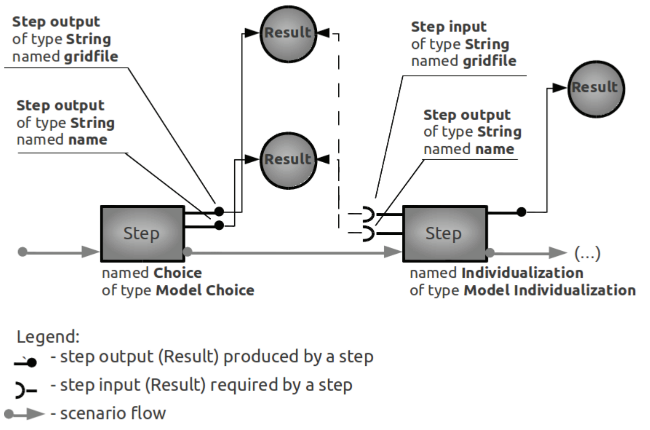
**Step** Base element of scenario. It has it's own name, and associated **plugin** class. Each step has any number of inputs and outputs, of defined type.

**Result** Product of processing a scenario **step**. It has it's own specified type.

**Scenario** Set of **steps** and **results**, where there exists connection between **steps** and **results**. Scenario has also defined one, or many suggested paths.

**Panel** Abstract object representing message between a user and the system used for interaction between them. Panel can be presented as a form, to be filled in by the user or in other case it can be used for visualization of results.

In this paper the authors focus on the last four definitions, which are the foundation of described flexible scenarios module. An example of defined scenario is presented in the Fig. 2. It must be stated that this example is very simple, and even not complete, but serves for presentation purpose well.



Step output of type String named gridfile
Step output of type String named name
Result
Result
Step input of type String named gridfile
Result
Step output of type String named name
Step
named Choice of type Model Choice
Step
named Individualization of type Model Individualization
(...)
Legend:
- step output (Result) produced by a step
- step input (Result) required by a step
- scenario flow

Fig. 2. Sample scenario, consisting of two steps (presented as rectangles), connected to three results (presented as circles).

All inputs, outputs and results have specific type definitions, so the scenario module could perform type check and prevent accidental mis-connections between results and steps.

Very important part of executing scenarios is the user's input. While processing a scenario, the system needs to ask the user about information necessary to run processors. This information is collected by panel with the fields defined by a processor plugin. To regulate step processing three phases of a step were specified:

**Setup** Phase in which a step is prepared to build panel for a user, or execute processor. In this phase only step inputs are available.

**Panel preparation** Phase in which panel presented to the user is being built. This phase defines which fields have to be filled in.

**Execution** Phase in which the main processor execution is performed. Execution phase has access to both step's input and the user input.

Step is processed in following order: at first, step plugin is found and instantiated. According to step inputs names, corresponding Java object fields are initialized. This is called **step inputs population**, because after this stage, all step inputs in plugin are set based on connected result values. Later, after all step inputs are set, step setup phase is executed. After this, possible step outputs generated in this phase are being stored and step plugin panel is being built and sent. In this moment, control is passed to the user, until next step is chosen. When the panel comes back to the system, once again plugin is instantiated and it's input is filled with result data. What is more, user input is collected from the panel and also passed to the plugin. Then, final phase is executed. Example step with presented phases is depicted in the Fig. 3.

After step full life cycle, a new step is found and processed as described until there is no step to process. It turned out, that three phase step execution is a very flexible solution, because it can describe many types of steps.

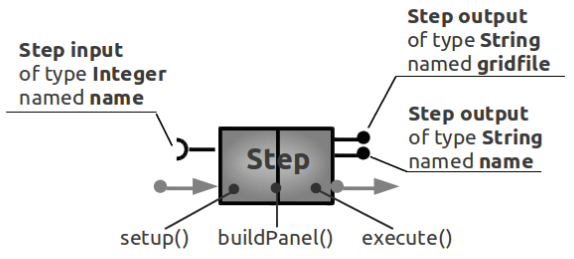


Fig. 3. Example of step, taking as an input mesh, and producing simplified version of it.

For now, let we consider three types of steps illustrated with examples:

**Full processing step** Step using both setup and execution phase. *Example*: Choosing model for FEM simulation. In setup phase, available models are fetched from database and packed into panel with drop-down list of those models. Execution phase, performed after collecting user's input, retrieves selected model and stores it in user result's repository.

**Visualization step** Step using only setup phase. *Example*: Visualization of 3D vector field. Setup phases reads vector field values and generates field view (represented as a 2D slice). Step's panel contains visualization component, and no components expecting user's input. Execution phase is unused in this case.

**Stateless processing step** Step using only execution phase. *Example*: Mesh simplification. In this case, there is no need for any setup, because all required data is provided. Panel built by the step may contain some tweaking parameters (for example the lowest acceptable number of elements). Main processing is held in execution phase invoking mesh processor. Stateless means in this case that result produced by this step depends only on input data, state stored in database is irrelevant.

Beside described three types of steps, there exists two more possible types which does not utilize panel generation phase. They can be extremely useful in automatic scenario execution, but in current system usage are not such significant.

## Implementation of flexible scenarios

One of the main goals of this module was to develop such solution which could provide easy way of adding new steps, plugins and processors. Adding new processor into proposed system is fairly easy, because it involves only installing processor and enabling it for execution by a operating system. Next step is to add new plugin exposing new processor to the system. Adding new plugin is done by creating new Java class, extending base plugin's class. This base class has three abstract methods: `void setup()`, `void buildPanel()` and `void execute()`. There is also helper function `void run(String command)` used for processor running.

Because input's and output's names are determined during runtime, there appeared need for safe mechanism of accessing data for step. Using String identifier is the simplest and also the most error prone solution, because misspelled name in one place leads to unrecoverable plugin's error. This the reason why the proposed module uses Java's mechanism of reflections. Each plugin's class has defined theirs fields of type `PluginInput` or `PluginOutput`. Using name of input/output defined in database, an expected field name is found. Rules on converting input/output names into field names are simply, and encourages using Java naming conventions. When plugin's data are populated, underneath the system finds appropriate setter, checks destination type and sets value. Any error is caught before running a plugin. In the future, this type of check can be performed before starting scenario.

It turned out, that using this approach many scenarios can be easily defined. Following designed conception alternative scenario paths were also implemented. In fact, using result–step relation alternative paths emerges automatically. It is the result of possibility of providing one result by few, different steps. In such case, only one is mandatory, and others are optional steps. Described situation illustrates Fig. 4.

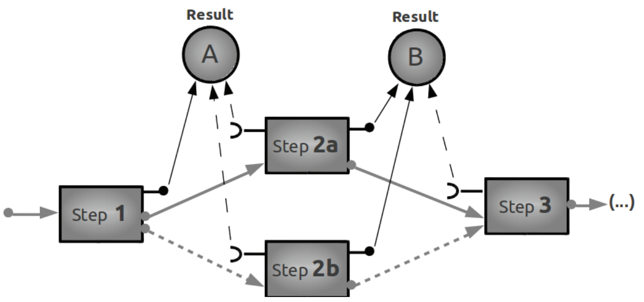


Fig. 4. Scenario with alternative paths. Both steps (2a and 2b) provide result B consumed by Step 3. Any of these steps can be performed to enable Step 3.

Implementation of this part of the system had been done using several, widely used and well—written tools as Java programming language, Hibernate persistence library, and CXF webservice framework. Final module takes advantages of all of this tools.

## Conclusion

The system is based on a predefined, precise scenarios helping a domain user not familiar with numerical modeling methods to solve realistic simulations using state-of-the-art simulation environments. The system is highly modular and can be easily extended with new functionalities provided by independent modules using a dedicated hardware platform.

**Acknowledgment:** this work was partially supported by Polish Ministry of Science and Higher Education (research grant no. N N510 148838).

BIBLIOGRAPHY

[1] Alexander C. et. al: A Pattern Language, Oxford University Press, 1977.
[2] Open Directory Project: Science: Technology: Software for Engineering `http://www.dmoz.org/Science/Math/Software/`, retrieved Oct. 10th, 2011
[3] Sawicki B., Chaber B., Starzyński J., Szmurło R., Internet application concept to trivialize EMF biomedical computing EHE 2011, 4th International Conference on Electromagnetic Fields, Health and Environment, 2011, Electronic Edition.
[4] Oliver Niehörster et. al. Providing Scientific Software as a Service in Consideration of Service Level Agreements Proceedings of the Cracow Grid Workshop (CGW), pp. 55-63, 2009.

***Authors:*** Ph.D. Jacek Starzyński, Ph.D. Robert Szmurło, M.Sc. Bartosz Chaber, Prof. Stanisław Wincenciak, Institute of Theory of Electrical Engineering, Measurement and Information Systems, Faculty of Electrical Engineering, Warsaw University of Technology, ul. Koszykowa 75, 00-662 Warszawa, Poland, email: jstar@iem.pw.edu.pl