Computing (2013) 95 (Suppl 1):S61–S74
DOI 10.1007/s00607-012-0278-9

Bioelectromagnetic simulation for everyone

Jacek Starzyński · Robert Szmurło · Bartosz Chaber

Received: 26 October 2012 / Accepted: 20 December 2012 / Published online: 8 January 2013

Abstract Cloud computation technologies open a new perspective for scientific computing. Sophisticated software can now be made available as an on-demand service, reducing costs of maintenance and improving availability of the software to end-users. The paper presents a web based service system for electromagnetic computations. The back-end of the system is built of open-source scientific code. The service is targeted on medical staff and electromagnetic safety engineers which usually are not familiar with electromagnetic field simulation methods but its design allows it to be extended for more demanding, scientific-oriented users. The primary goal of the system is user friendliness for medical and engineering staff combined with flexibility, scalability and extendibility for scientists developing simulation software. These goals were achieved with help of carefully designed concept of configurable usage-scenarios (workflows). The purpose of this paper is to present in detail the concept and implementation of scientific scenarios.

Keywords Computational electromagnetics · Software engineering · Web service · System design

Mathematics Subject Classification 68M14 · 68U35

1 Introduction

Scientific computations can generate huge costs related to both human and hardware resources. Maintenance of the hardware, updates to new versions of simulators require employment of technical staff, take time and generate financial costs usually related

J. Starzyński · R. Szmurło · B. Chaber (✉)
Institute of Theory of Electrical Engineering, Measurements and Information Systems,
Faculty of Electrical Engineering, Warsaw University of Technology,
ul. Koszykowa 75, 00-662 Warsaw, Poland
e-mail: chaberb@iem.pw.edu.pl

to purchasing of expensive upgrades. Very often scientific computations require a lot of effort from the end users and additionally a deep understanding of numerical methods and simulation system software. Scientists must choose the most efficient numerical simulation software from plethora of available packages—both closed and open source.

Commercial codes tend to lower the barriers to use simulation methods. They usually offer a quite complete and universal simulation environment for typical problems, but plugging new methods or solving nonstandard tasks in such systems can be very difficult if possible at all.

Open source codes do not offer such complete environments. Some of them meet the user's requirements for numerical computation, while other have implemented better visualization techniques. The user of open source simulators usually have to convert and migrate data. To find satisfying solution of the problem one have to struggle with converting data from one format to another between numerous software bundles. Not only it requires user to get to know various software solutions but also prevents him from sharing his simulators, because there is no easy way to combine different applications with each other. An additional work is necessary to make a simulator useful. This is effort for creation of special tools, programs, converters which must be bundled together with the simulator.

Modern technology enables us to seek a solution for these challenges. Using existing software it is possible to build a platform which will both, allow users without experience in the field of numerical methods to solve their simulation problems and provide a space for collaboration between scientists allowing them to share their components with each other and publish their simulators to nontechnical staff. The main purpose of this paper is to present the concept, the architecture and the implementation of such a platform.

Current state of the Internet technology allows anyone with an Internet connection to use computational resources of the external computing system. This is obtained with the web services technology, web front-end applications or a desktop application connecting to the remote interfaces.

The idea of the remote system for electromagnetic simulations emerged when the authors were developing a software tool for computer-aided design of electromagnetic flow measurement systems [7]. This tool was created in cooperation with a technical team, implementing the real, hardware flow-meter. The software design process required repeated delivery of new versions of the application. The authors have observed that a lot of effort has been consumed by creating installation packages and dealing with hardware incompatibility problems on technical team computers. This experience proves that the system presented in this paper would be highly beneficial.

Similar solutions were already presented [1–6], but we propose a new and unique solution which can be tuned to meet needs of wide audience: from inexperienced users to experts of numerical field simulation. What we would like to present and prove in this paper is that the system can be entirely implemented using Internet software without the need of installation of any dedicated applications on end-user computer.

This paper focuses on the concept of practical realization of scenario-like approach in the context of high usability of the system for the technical and non-technical users as is organized as follows. It starts with the detailed presentation of the target audience of the system and the expected functionality. The next section deals with the problem of natural guided user interface and high accessibility of the system to the users. It was achieved by creating a web based user interface and a flexible Scenario engine. The next section of the paper presents the architecture of the system, after which comes the most important part: description of the implementation.

2 Fundamental problems

The web-service based systems for numerical (especially biomedical) simulation of electromagnetic fields must solve some fundamental problems: transferring possibly huge input/output data through the Internet, privacy and security issues and finally an intuitive and extendable user interface. While bearing in mind two first problems, in this paper we focus on the last one.

The problem of data transfer was already addressed in other works of authors [8–10]. The amount of input data used to built the simulation model is kept minimal by methodology of mesh morphing. The concept of mesh morphing needs a separate paper to be presented in detail and it was described in [8]. Summarizing, a user needs to send to the server only a few measurements of the individual patient (see the left inset in Fig. 2). They are used on the server side to morph the basic, realistic model of human body with fine discretization and precise anatomical details into the individualized model. Thus the complex models and grids are stored and processed on the server side. The quality of approximation of the measured object with the morphed grid increases when more input data are available. Currently we are focused on avoiding the data transfer on the cost of the model quality, but the trend in the Internet connection speed assures us that this barrier will lower with time passing.

The network connection latency seems to influence seriously the interaction between a user and a post-processor running on a remote server. To solve this problem the data compression algorithm aimed at efficient mesh sending over the net was designed [9]. This method allows the server to transfer quickly a reduced surface grid to the client side. This reduced mesh can be conveniently manipulated even on a smartphone browser [10]. Thus we use client-side rendering of a reduced model to control the post-processor running on the server side (see Fig. 2). The high quality images of fields may be rendered on the server and send to the client side on user's demand.

The privacy and security issues seem to be solved with standard TSL/SSL protocols. They provide secure communication over the Internet for web browsing, electronic mail, Internet faxing, instant messaging and voice-over-IP. The user data privacy may be easily assured by one authentication over the secure connection. Thus the privacy and security issues are out of scope of this paper.

One of the main concerns regarding the presented system was the form of guiding the user through the whole process to finish scientific simulation. As a form of presentation of this work flow the authors have decided to use the scenario-like approach. The Scenario is divided into smaller autonomous tasks called Steps. Each Step can collect input from the user using automatically generated user interface. The input is persisted by the system during the process of simulation, and provided if necessary when the Steps are executed. By execution of the Step we understand running of a program or a script on the server side. The numerical tools (programs and scripts), created by

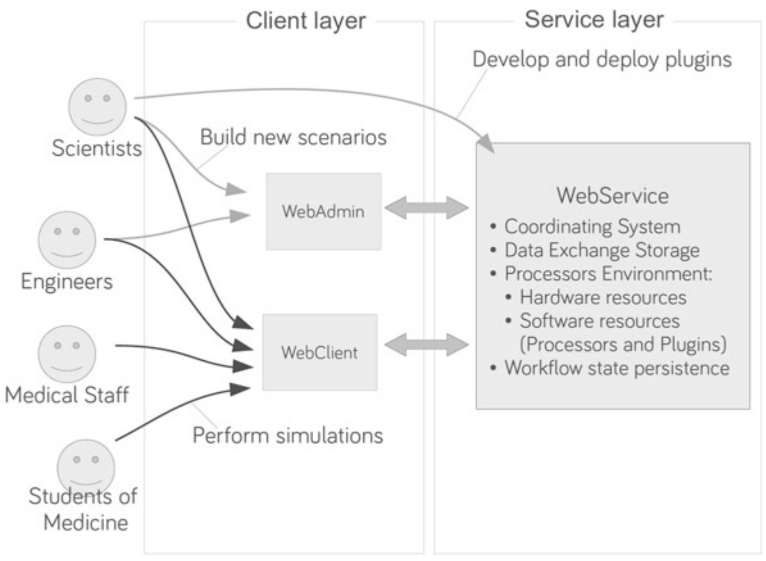


Fig. 1 User roles and dedicated functional system modules

scientists, are connected to the Steps using a uniform interface called Plugin. Plugins are deployed in the system and are specific to various numerical tools. They provide a unified way to execute numerical tools and exchange the results of the simulations between the system data storage and tools. The part of the system which is responsible for executing Scenarios is called Processors Environment.

The second aspiration of the authors was the ability to easily extend the system. We can distinguish two kinds of extendibility: development of new, general numerical tools and creation of new Scenarios fitted to specific problems. The new numerical tools can be deployed to the Processors Environment as a software package and must only provide a Plugin, which will be used by the system to communicate with a tool. The new Scenarios can be configured in the system using dedicated Scenario editing tools which are implemented in the WebAdmin application of the system (see Fig. 1).

3 Users and concept of functionality

The targeted users may be divided according to their expectations and level of expertise in numerical methods. We have distinguished three groups of users. They are presented in the Fig. 1 (Medical Staff and Students of Medicine are combined into a single group in the following explanation).

The primary group are nontechnical users who want to analyze the results of simulations fitted exactly to their needs without the necessity to configure anything. We assume that those users can be guided by a wizard/creator requiring the possible minimal amount of the input data and presenting easy to interpret results. The system is aimed primarily on bioelectromagnetic simulations and thus this group of users can be related to medical staff. They may use the system for planning individualized therapies, designing better general therapeutic methods and developing deeper understanding of the electromagnetic treatment of human organism.

The second group of users is the technical staff (engineers) who have the knowledge of the domain problem but are not necessarily familiar with numerical methods of field simulation. They may need more sophisticated interface with more parameters and more complicated output showing many different field quantities, but they do not need to modify existing, predefined Scenarios of simulation. The engineers can use the system to design new or improve existing bioelectromagnetic devices. From their point of view our system behaves similarly to a sophisticated multi-physics simulator, but does not need to be installed and maintained on their hardware. It can be accessed on-demand.

The last and the most advanced group of users are scientists who develop simulators, numerical tools, test them and share them to the other groups of users. The scientist can both contribute to the system development by uploading new simulators or can use existing ones to simplify their work and improve productivity. The systems offers them a unified, but agile environment in which their components may be tested.

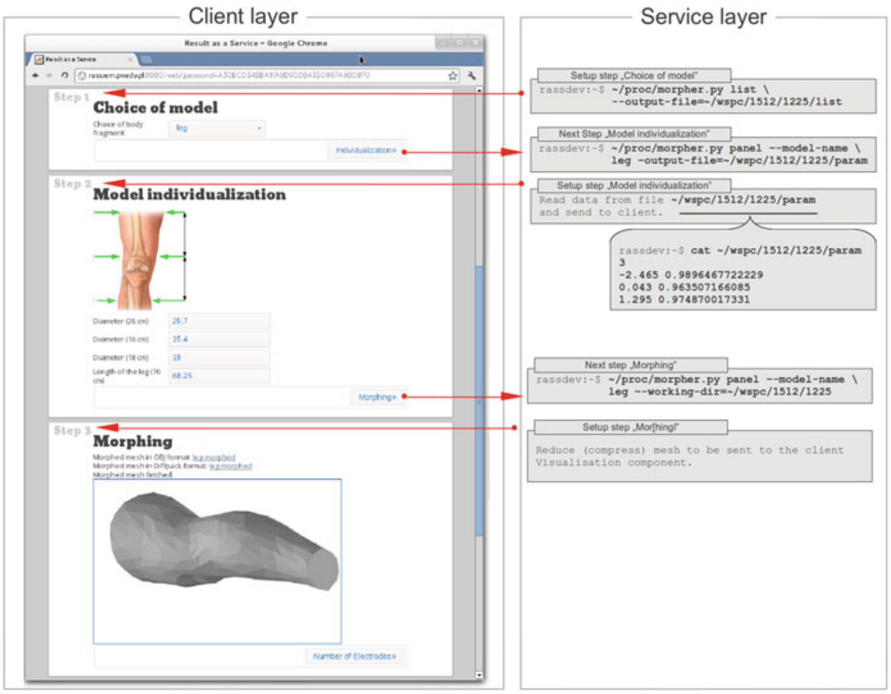
By using a dedicated framework, the creator of a new numerical tool can concentrate on the module itself, and for example reuse exiting set of pre- and post-processors. To share their code the scientists need just to plug it into the system, again reusing the existing Scenario engine and the user interface generator. Once plugged, the new module becomes an autonomous and fully functional step, which can be used in any Scenario.

Reviewing the previous paragraphs it becomes clear, that a compromise between the essentially different expectations of the above mentioned groups of users needs various interfaces to the system. However, we did not want to end with a variety of specialized interfaces. We would like to limit their number to the required minimum to avoid endless and tedious process of GUI creation.

Distinction between the first two and the last group of users suggested, that the system should have a two-level interface: the lower level should allow a specialist to do more, while the high level should allow only a limited guided access, suitable for less advanced users. Secondly, we have decided to create a mechanism for automatic generation of user interfaces based on specification provided by Plugins. These Plugin-specifications are defined at a high level of abstraction what allows us to adjust the user interface to the technology used by a client or to the user demands.

The connection between Steps and Plugins can be understand in the following way: a Step is an abstract form of an action performed by a user. On the other hand, a Plugin is an actual action (program/script execution or file creation). This is illustrated in Fig. 2. As it can be seen, every Step (Choice of model, Model individualization, Morphing) is used to gather some data from a user. Then this data can be translated into a command (for example `morpher.py list` lists available models for morphing and saves the output to the `output-file`) or the parameter file (`param_morph` is the file filled with data computed basing on user's input). The important part is that with each running Step only one Plugin is associated. However, this single Plugin can perform several actions before and after the user provides an input.

In the next sections we present the ideas and tools used to realize the following three main concepts of the system:



```
rassdev:~$ ~/proc/morpher.py list \
           --output-file=~/wspc/1512/1225/list
```

```
rassdev:~$ ~/proc/morpher.py panel --model-name \
           leg -output-file=~/wspc/1512/1225/param
```

```
Read data from file ~/wspc/1512/1225/param
and send to client.
```

```
rassdev:~$ cat ~/wspc/1512/1225/param
3
-2.465 0.9896467722229
0.043 0.963507166085
1.295 0.974870017331
```

```
rassdev:~$ ~/proc/morpher.py panel --model-name \
           leg --working-dir=~/wspc/1512/1225
```

```
Reduce (compress) mesh to be sent to the client
Visualisation component.
```

Fig. 2 Connection between Steps and Plugins: a user provides the bare minimum input data, which is translated by the system into configuration and parameter files, as well as into operating system commands. The base model of a human knee is transformed into an individualized model on the basis of four anatomically based measurements. The reduced surface mesh will be sent to the client's browser to allow verification of the result of morphing

– Internet web service / web client technologies,
– wizard based, automatically generated user interface,
– flexible Scenarios (workflows).

3.1 Two-layer web system

Recently a very strong trend in the software development can be observed. More and more services are translated to be accessible as a web application or a web service. Both technologies were used in the presented system.

Many existing applications used for scientific simulations or visualization can be extended to use web services. It is possible to build an application which performs CPU-heavy operations of electromagnetic field simulation using an infrastructure hidden behind the web service. The hardware resources will be delivered to this application transparently with the help of a web service. The application can fetch data stream directly from a web service and present it to the user. In our work we have distinguished two functional layers of the web system: the Client layer (web client) and the Service layer (web service—see Fig. 1). The most advanced group of users may communicate with both layers: the Client layer to use the system for numerical computations and with the Service layer to modify the behavior and functionality of the system. The nontechnical users communicate only with the Client which offers a simple and intuitive interface suitable even for an incidental visitor.

3.2 Intuitive and natural guided user interface

To provide a user an elastic and easy to use interface of the web client we have decided to build it similarly to wizard creators. This type of interaction resembles dialog between a specialist and a nontechnical user. Specialist asks the user a series of questions or presents him possible options. The user is guided through the whole process, giving the answers and deciding about the most important from his point of view parameters of simulation. Because by default only the necessary answers are gathered from a user, he is not confused with variety of parameters which are often meaningless to him. The suitable values of the low level parameters are chosen by the Scenario designer. A user is able to change most of them, but only on the explicit request. After each Step the user is presented a set of possible next actions. The user is suggested one or two possible Steps from the main Scenario path. The user usually has many valid Steps to continue Scenario. In addition he can find a list of all other Steps valid for execution which are less emphasized than those suggested ones. What allows to execute any action (Step) is the availability of necessary input data.

3.3 Flexible Scenario engine

In order to allow Scenario designers to easily reconfigure their Scenarios or assemble them from available Steps, a flexible Scenario engine had to be implemented.

One of the main aspects considered while designing this part of our system was connection between Steps in such a way, that they were possibly independent from each other. On the other hand we needed a mechanism of validating if the Scenario, built of independent Steps, will allow one to obtain the desired simulation result.

To solve this issue we decided to extract data processed by Steps as individual entities. This way, Steps do not directly rely on each other, but rather on data they process. The data is related to the Steps by its inputs and outputs. In this model it is possible to replace a Step in a Scenario by another Step withe the same type of input and output The difference between an approach based on Step dependencies and the one based on data dependencies is clearly visible in Fig. 3.

Our idea of Scenario data flow is based on Steps acting as functional blocks. Every Step can be considered as a function with a set of arguments (inputs), possibly returning multiple results (outputs).

In general the Scenario engine is divided into 3 layers (which are visible in Fig. 4). The bottom layer is designed to store unconnected, independent definitions of base entities in our system: Steps and Result Types. At this level a Step has only its base properties like the name, the associated Plugin Java class and the inputs/outputs. A Result Type is the list of all possible data types of inputs or outputs of Step

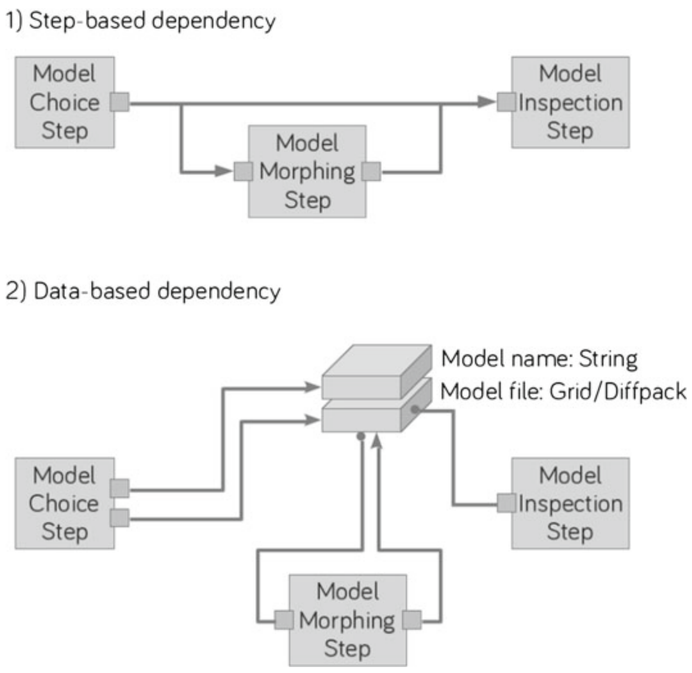


Fig. 3 Data dependent Scenario flow—how we detached Steps from each other

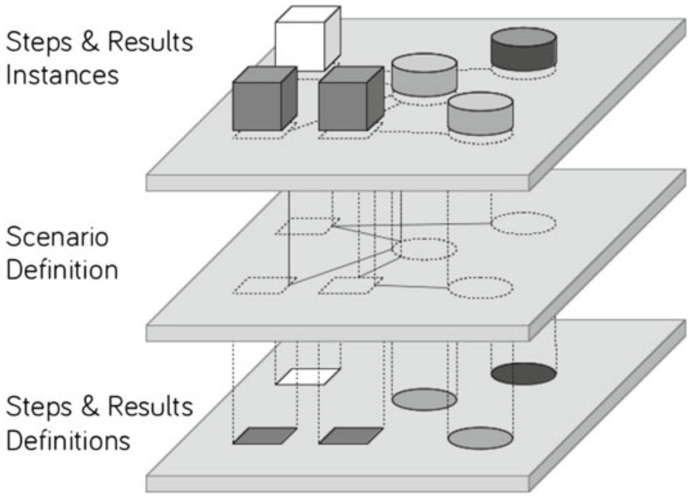


Fig. 4 Concept of mutlilayered Scenario representation

(e.g. 3DSurfaceMesh, 2DVectorField, List/Integer). This layer can be seen as a set of templates of Scenario Steps and Scenario Results.

The middle layer works on totally different level of abstraction. It operates in context of a Scenario. Using elements (referred earlier as templates) a designer with access to WebAdmin application is able to compose definition of Scenario. It is worth noticing that in this layer we are using references to entities from the definition layer. What is more, in the middle layer we can have many references (Scenario Steps) to a single Step definition (we can use each Step definition more than once, and connect its inputs and outputs with different results). In the middle layer the most important are connections between Steps and data (Results). It is still only a **definition** of Scenario.

The last layer is the instance layer. Whenever a user starts a Scenario, the system instantiates it according to it's definition from the second (middle) layer. In fact it is the exact copy of this definition but all entities are extended by unique identifier of the user. This way we can separate the results generated during two simultaneous sessions of the same Scenario. The top layer is similar to a process running in an operating system. The process is also copied every time it is executed (the definition of the process is the program stored on disk). The main difference between running processes in operating systems and our approach of the Scenario instances is the structured way of handling data of finished instances.

In operating system, after the process is finished, all memory which was assigned to it is released and the operating system does not care about any results produced by the process. Usually it even does not have any knowledge of it. In our system data results produced by the particular Scenario instance are stored in a dedicated Data Exchange Storage and our system keeps references to it in the database.

4 Architecture

In order to build software which will be very flexible and ready for extensions a layered architecture has been chosen. The diagram of this structure is presented in Fig. 5.

The first is the Client Interface Layer responsible for the interaction with a user and presentation of the results. This layer serves as an interface between the user and the computational engine. In fact, this engine can be located very far away from the user, because it is exposed using web service technology. The main reason for this

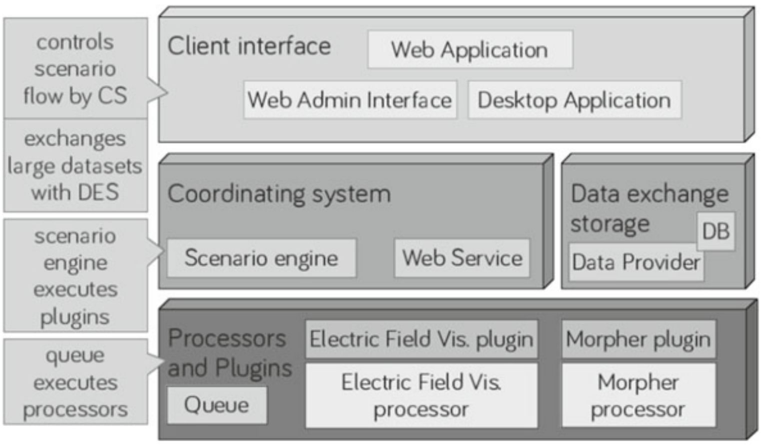
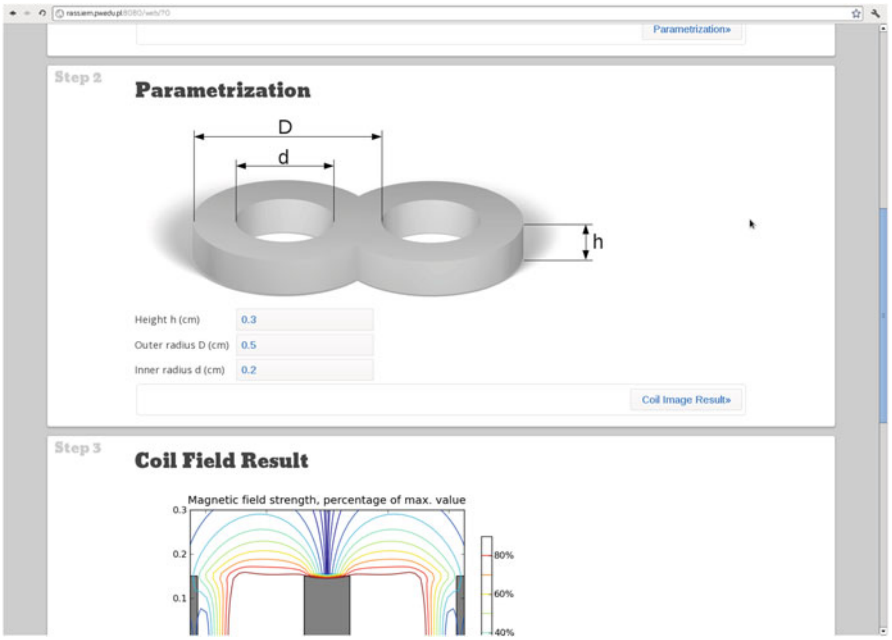


Fig. 5 The main concept of system architecture



Parametrization»

Step 2

Parametrization

Height h (cm) 0.3

Outer radius D (cm) 0.5

Inner radius d (cm) 0.2

Coil Image Result»

Step 3

Coil Field Result

Fig. 6 Coil design Scenario in action in a PC's web browser

is that the same web service can be connected to a desktop application residing on user's personal computer or a web application deployed on a server connected with the web service by very fast and reliable connection. The second layer is the engine of the service. It is called Coordinating System, and it's main purpose is to provide a common way to serve the computational resources to other clients. This part of the system contains a Scenario execution environment (Scenario Handler) and a set of specialized Plugins invoked by Scenario Handler.

These Plugins cooperate with programs and scripts or even sophisticated simulation systems (generally called Processors) which form the third layer of the system. It is worth stating, that the Plugins are wrappers around Processors. They provide universal way of passing information between different and often incompatible Processors, so they tie together the specific script or program with the abstract Step entity.

4.1 The user interface

Decision of choosing web user interface for our execution engine was motivated by intention of making the software available to the people without computer, numerical or scientific skills. This approach has various advantages over desktop applications. One great advantage of an application available in a web browser is lack of need of installing any software by the user. Software is ready to be used just after opening web browser. Using our Wizard-like web interface the user is able to access his simulations and results not only using PC, notebook but even smartphone having Internet connection (see Fig. 6). This way a user can perform their simulations even without access to a stationary computer if he has a modern smartphone or an Internet tablet.

One of the biggest challenges was to overcome big differences in performance between possible types of devices. Example of such challenge was presented in [10].

4.2 Database storage

Database storage is a very important element of the system. Its main purpose is keeping Scenario definitions, Scenario Steps, results and persisting state of executed Scenario instances. The database serves also as a communication media between the Scenario engine and a queue runner (scheduling system) which is responsible for managing hardware resources. Additionally the database is used as a shared memory for components of the system.

In our architecture we allow only the Web Service, the Data Provider and the Queue to connect with the database. It simplifies communication between applications and improves security, because we don't expose all our database structure to clients.

4.3 Processors and plugins

One of challenges that we have faced during designing our system was developing a way of invoking external processes by the Coordinating System. We wanted to wrap this process call in object oriented way, so it would be consistent across our system. The idea of Plugins emerged. Plugins are plain objects in Object Oriented sense, which are aware of Processors existence. They know how the particular Processor can be invoked. Plugins translate the parameters provided by a user into the command parameters for a Processor. Another crucial element of our system is the queue component. It is tightly connected to the concept of Plugins and Processors. The Queue is the connection point between our system and an operating system. When a Step is executed, the Coordinating System contacts with the associated Plugin and provides it with all required input data. Based on the input data the Plugin determines what specific Processor must be started and which data it should process. After the final command is prepared it is passed to the Queue.

The main purpose of the Queue is to provide an interface to operating system. This way, we provide a single way of scheduling new tasks for different numerical and operating system environments. We can also choose a different Queue implementation for the local system installation and different for remote installations. We can also provide a special Queue implementation capable of submitting tasks directly to computational cluster's job scheduling system or to a computational cloud.

The main overview of connections between database (DB), Queue and WebService is depicted in Fig. 7.

5 Implementation

Currently our system uses Apache CXF Web Service framework, which enables SOAP based Java implemented web services to be easily exposed. We have defined general API and conversation formats. In our system we distinguish between two different

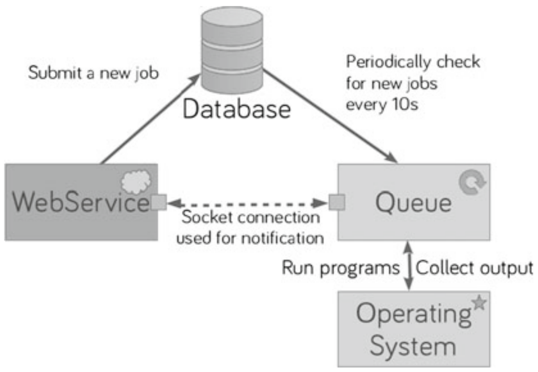


Fig. 7 Using database as a connection between WebService and Queue

types of messages: CreateScenario message and TakeNextStep message. The first one is used only for initialization of the new Scenario instance. The second one is the main part of the conversation between the web service and a client. When the client is willing to take the next Step in the current Scenario the web service sends message in XML format. This message describes what parameters are required for the next Step, how they should be collected (by specifying the type of the parameter) or what are the constrains for the parameters. Additionally, this message can hold also definitions of components and the data which should be visualized by the given component (e.g. a message can contain information about the 3D surface mesh which should be presented as a 3D interactive model or the still image projection). By using general types of parameters any client using our WebService is able to create the User Interface component best suited to gather desired input from user or to present data in the most convenient way. When some parameter's type is unrecognized, the client application can notify the user about unsupported content.

Because WebService is already working in the Java EE Container, the web based user interface and the web admin management system were also implemented using this technology. The Apache Wicket framework was chosen to simplify generating highly dynamic web pages. Thanks to the Component concept of this framework, it was very easy to achieve modularity of this part of the system. In particular, every component returned by the WebService has its counterpart in the web user interface application and this user interface component is easily exchangeable.

Scenario engine can access data stored by the Steps in database, but what is more important it can also submit a new job request to the Queue. There must be specification of the Processor's command to run, the submitter's name, and such a new job must have appropriate status (in this case NEW). After adding to the database, the unique ID is generated. Normally, Queue is waiting in an infinite loop, checking for new jobs every 10 seconds. When the new job is available it is automatically executed. However, to improve the user interface we combined this standard approach with additional notification system. When the client is using the web server application and he requests some action to be taken by the system, the Scenario engine adds a new job and memorizes the generated ID. After that, it can connect to the Queue's socket and send there a message to force checking the database for new jobs, and also to subscribe for the notification when the job with particular ID is done. Queue manages

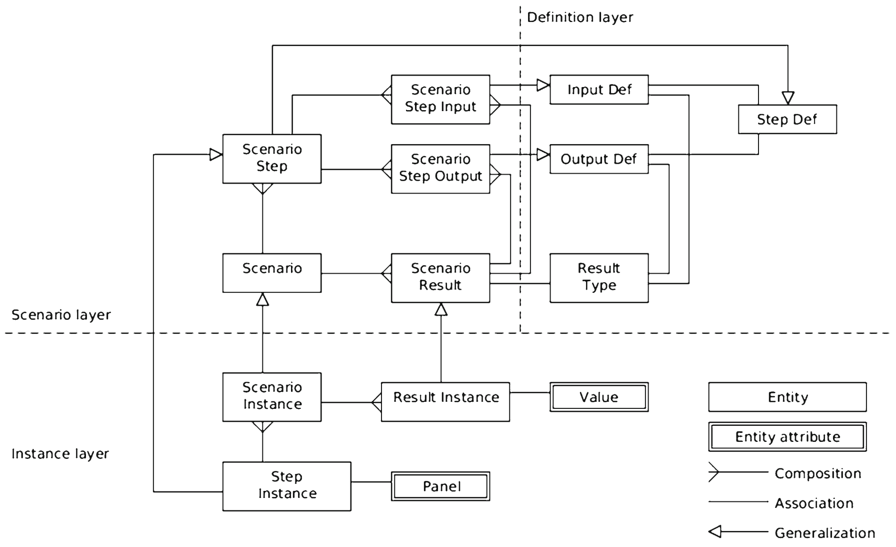


Fig. 8 Structure of the database entities used for representation of the Scenarios

the list of subscribers for different job's IDs and when the job is finished it uses the established socket connection to reply with ID of the finished task. Then the Scenario engine is able to retrieve the status code and to process the output from the database.

The Queue was implemented using Python 2.7, because of it's convenient socket handling library and support for various database engines. Also, Python's sub-process module was used to allow invocation of the ex ternal Processors.

In Fig. 8 we present part of the database, which defines the structure of Results, Steps and Scenarios. We enabled our database for holding all byproducts of Scenario Steps in a way that they are accessible from Scenario engine. It was achieved using a separate layer of Scenario instances mentioned in the Sect. 3.3. Because each instance is a copy of the Scenario definition, all result's definitions are also copied.

Every time the Step is taken, it accesses a Scenario's **instance's** result and fills its with data. In case of simple data like lists, strings or numbers it uses a string representation. After that, using the result's type we can always determine the original data type and parse it's string representation. In case of file results, only the file path is stored. Together with a shared file system solution like NFS, we are able to store data effectively while still being able to access them from our system's back-end components.

In our database scheme we have separated data related to define Scenarios from jobs or application configuration. Using a relational database we were able to implement our idea of the flexible Scenario engine described in Sect. 3.3.

6 Conclusions

The innovation presented in this paper is the configurable and extendable simulation environment for computational electromagnetics. It can combine many different programs into one system and its configuration is determined by a database contents.

Thus it is extremely easy to add a new component, even to the working system. Additionally the system is available in the Internet and it can be accessed through a standard web browser.

The presented system was successfully implemented and installed. It's usefulness has been proven in a common research work on software development. As the researchers we are fully contented of comfortable environment in which a new electromagnetic code can be installed and tested. Today we use it not only for bioelectromagnetic software, but for medical image processing and even for antenna design.

The concept of a general framework of web interface for scientific computing seems to be and ideal way to improve and to share methods and tools for numerical field simulation.

In our opinion this idea fits ideally into the nowadays trend of cloud-based services accessed on demand by even untrained incidental users, who want to benefit internet technology at possibly low costs.

Acknowledgments This work was partially supported by Polish Ministry of Science and Higher Education (Research Grant No. N N510 148838).

References

1. Taylor IJ, Deelman E, Gannon DB, Shields M (2006) Workflows for e-Science: scientific workflows for grids. Springer, Berlin
2. Curcin V, Ghanem M (2008) "Scientific workflow systems–can one size fit all?". In: Cairo international biomedical engineering conference, 2008. CIBEC 2008, vol 1–9, pp 18–20
3. Altintas I, Berkley C, Jaeger E, Jones M, Ludascher B, Mock S (2004) Kepler: an extensible system for design and execution of scientific workflows. In: Proceedings of the 16th international conference on scientific and statistical database management
4. Eker J, Janneck JW, Lee EA, Liu J, Liu X, Ludvig J, Neuendorffer S, Sachs S, Xiong Y (2003) Taming heterogeneity-the Ptolemy approach. Proc IEEE 91(1):127–144
5. SCIRun: A Scientific Computing Problem Solving Environment, Scientific Computing and Imaging Institute (SCI), http://www.scirun.org
6. Sascha MS (2012) NCLab-public computing laboratory. Computing, doi:10.1007/s00607-012-025-3
7. Starzyński J, Szmurło R, Michalski A (2009) Computer aided design tool for electromagnetic sensors. IEEE Instrum Meas Magazine 12:28–33
8. Szmurło R, Starzyński J (2009) Specimen-specific finite element models of human head obtained with mesh morphing. Elec Rev LXXXV(4):47–49
9. Bartosz C (2011) Progressive linear triangle surface mesh implementation. In: 20th international meshing roundtable pitfalls and challenges. http://www.imr.sandia.gov/20imr
10. Bartosz S, Bartosz C (2012) 3D Mesh Viewer Using HTML5 Technology, Przeglad Elektrotechniczny (Electrical Review), ISSN 0033–2097, R. 88 NR 5a/2012, pp 155–158