2017 21st International Conference on Process Control (PC)
June 6–9, 2017, Štrbské Pleso, Slovakia

A Web-based Tool for Design of Simulink Models

Ľuboš Čirka, Martin Kalúz
Institute of Information Engineering, Automation and Mathematics
Slovak University of Technology in Bratislava
E-mail: lubos.cirka@stuba.sk, martin.kaluz@stuba.sk

***Abstract*—This paper deals with the development of an educational web-based application that allows users to create simple block schemes in the Internet browser. The structure of created schemes is based on XML, and it is entirely compatible with the Simulink environment. The application allows the user to perform an evaluation of created schemes by executing them in the remote instance of MATLAB/Simulink via an HTTP MATLAB Web Server. The result of the simulation is returned either in the form of a graph or a data file for further processing. The application can be used in education for a teaching of systems' modeling and control, and also as a support tool for development of virtual and remote laboratories.**

I. Introduction

Many universities, research centers, and companies use the MATLAB for various computing-related tasks, such as data processing, statistics, modeling, simulations, control design, and many others. In the past, the MathWorks company has developed a toolbox called MATLAB Web Server [1] that allows performing the computational tasks via the Internet. However, this toolbox has been discontinued since the version 2006b, and new distributions no longer support it. This toolbox has been replaced by independent open-source project `Web Server` by a community member Dirk-Jan Kroon [2].

The principle of the server is based on execution of native MATLAB code over the Web. The server side service is emulated in by Java package classes java.net.* which are designed for I/O network communication handling. The service in set up to process common HTTP requests for GET and POST methods sent from client side Web pages. The source code of server consists of M-file executable scripts and functions containing base methods for handling HTTP request/response model and methods for local execution of server-side processing scripts.

Both mentioned toolboxes are also used in the education process [3], [4]. Based on them, several other educational tools have been developed in different universities such as virtual and remote labs for monitoring and control of dynamical systems (CyberVirtLab [5], Recolab [6], WebPIDDESIGN [7], [8]).

Both toolboxes exhibit several disadvantages compared to pure MATLAB/Simulink execution of models/scripts. These are:

1) the absence of mechanism for online change of parameters during the execution of simulation - limited interactivity;
2) inability to continuous gathering of simulation data or their visualization during the run of task - only batch mechanism;
3) the absence of feature to create user defined models (even the change of parameters of existing ones is available). The schemes must be created in advance and then uploaded to Web server. This procedure requires a Simulink to be available for end used.

The solution to first two disadvantages has been already provided in [9].

Many various software projects deal with the creation and editing of block schemes. These are e.q. JointJS[1], Mxgraph[2], GoJS[3], Draw2D[4], etc. The main disadvantage of these tools is that they do not account for all the specifications compatible with the Simulink (for example, they allow to define multiple connections between block or does not allow to define their parameters). The tool that allows the creation and editing of a Simulink scheme have been developed at FEI STU in Bratislava [10]–[12].

The main goal of this paper is to describe a new tool that allows not only a definition of block models in visual form, but also their remote execution, data acquisition, and visualization of results.

The paper is organized as follows. Section 2 describes the main structure of current Simulink model format, defined in the form of an XML file, along with the main elements required for definition and visualization of particular blocks. Section 3 deals with the structure and operation principles of created Web-based application. This section also describes the technologies and languages used in the application, procedure of block library extension. Additionally, the third section describes the process of scheme creation, including block insertion, the configuration of parameters, the interconnection of blocks, and deletion of scheme objects. An illustrative example of model definition/editing is shown in section 4.

II. Simulink Models

The current versions of Simulink stores the model information in MDL files or compressed SLX format (by default). The developed application works with the more recent version of models (SLX) that is XML compatible. The main internal file structure of SLX is:

[1]http://www.jointjs.com/
[2]https://www.jgraph.com/
[3]http://gojs.net/latest/index.html
[4]http://www.draw2d.org/

978-1-5386-4011-1/17/$31.00

```
_rels
    .rels
metadata
    thumbnail.png
    mwcoreProperties.xml
    coreProperties.xml
simulink
    blockdiagram.xml
[Content_Types].xml
```

All the information about a particular scheme is stored in `blockdiagram.xml` file located in the sub-directory `simulink`. The root element of the model is called `ModelInformation`. This element contains additional sub-root element `Model` that carries all the information about schema and its operation. Most important parameters to mention are settings of the editor, configuration of simulation (solver, time base, sampling, etc.), default and user-defined parameters of simulation and blocks, and visual representation.

Each block contains a set of predefined parameters that are used as default parameters when a block is firstly created. A user can later redefine these parameters.

Mechanism of block generation is as follows. The action of new block insertion triggers the copying of predefined parameters (`<Block>` elements) into `<BlockParameterDefaults>` in the model. If the user changes a parameter of a block, this change is reflected by copying the `<Block>` element into `<System>` element, while actual parameters are stored in `<P>` elements. Each `<P>` contains an attribute with the name of the parameter, while the value is enclosed in the element's tag.

III. Web Application

The application is developed on the top of several technologies and languages, and it split into three operational layers. The first layer is a user interface (served by a Web browser), written in HTML5 and driven by JavaScript. For the simplification of development and addition of interactive features, libraries *jQuery* and *jQuery UI* were used. The mechanism of block manipulation in schemes is provided by another JavaScript library *Fabric.js*[5].

The second layer of the application is written in PHP, and it is located on a server side. This processing part contains a set of scripts that are used as a program interface to the bottom layer, where MATLAB HTTP Web Server is located. The PHP layer performs the translation of user-defined schemes (initially handled as JavaScript objects and send in JSON) into an XML representation, used by Simulink. Additionally, the PHP part is used for reading and writing SLX files, and for constructing and sending the requests to MATLAB to run simulations and acquire results.

The third layer contains an instance of MATLAB with listening HTTP Web server. This server allows invoking the execution of predefined M-files to run the Simulink model via standard commands.

[5]http://fabricjs.com/

The operation of the application uses the following principles. The Web application is provided to a user on demand via entering its URL into a browser. The application loads the newest set of predefined blocks from the server as a JSON. This set is located in the block library, and it is continually updated over time. The whole work-flow of scheme design is performed in a browser, and actual visual and logic part of the scheme is kept in a JavaScript object. Any change of block scheme in editing window is immediately projected into the object representation. Finally, the created model can be saved on a server and downloaded back to a user. By executing this task, an object representation of a model is sent to PHP layer that loads the template SLX file, writes the whole model into it and provide a user with reference to file. A user also has the option to run the file directly on a server in Simulink. In such a case, the simulation is executed with the parameters defined in the scheme and numerical and graphical results of output blocks are returned to a user.

All the communication, except the acquisition of simulation results, is performed in an asynchronous mode using AJAX. The architecture of the application is shown in Fig. 1.

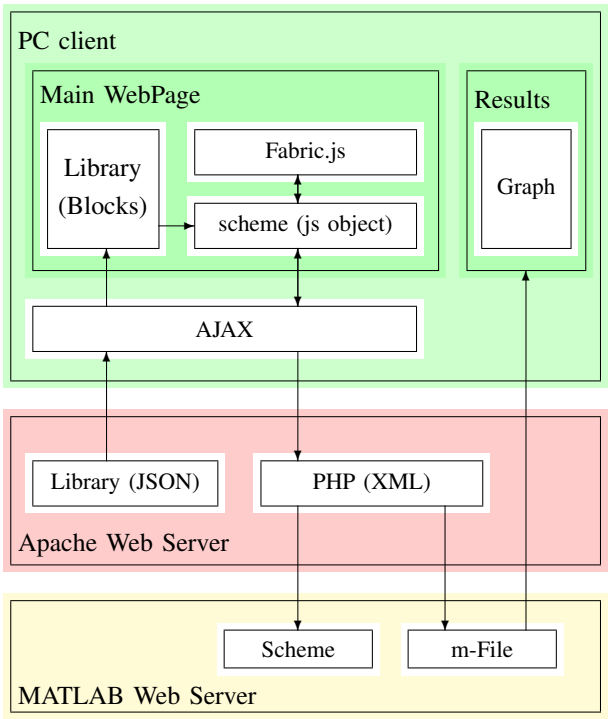

Fig. 1: Operational Architecture

A. Modularity of Application

The application is designed in such a way that it allows simple extensions of block library without changes in program core.

The definitions of all available blocks are located in the directory called `library`. Each definition is located in a separate sub-directory, and it contains four separate files:

- `display.json` – defines an appearance of block;
- `param.json` – stores default parameters of block;
- `default.xml` – data structure for `BlockParameterDefaults` element;
- `system.xml` – data structure for `System` element.

The addition of new block into the library is quite simple, and it is explained on the following `Digital Clock` example.

In the `library` directory, a new sub-directory is created using the name of new block (in this case `digitalclock`). The content of files `default.xml` and `system.xml` is taken from a model that was previously created in Simulink. It is necessary to create a scheme with the desired block and save it in SLX format. Then, after the internal files are subtracted, the element `<Block>` (representing a digital clock) is copied from the `<BlockParameterDefaults>` of `blockdiagram.xml` into `default.xml`. The definition of this block is as follows.

```
<Block BlockType="DigitalClock">
 <P Name="SampleTime">1</P>
</Block>
```

The implementer repeats the procedure for another `<Block>` element (digital clock) located in `<System>`.

```
<Block BlockType="DigitalClock"
  Name="Digital␣Clock" SID="1">
 <P Name="Position">[110,68,175,92]</P>
 <P Name="ZOrder">1</P>
 <P Name="SampleTime">1</P>
</Block>
```

This structure is inserted into a file `system.xml`. An information about user-defined block's parameters are stored in `param.json` file. The first part contains general info about block properties and appearance:

- `io` – connection type (`in` - source block, `out` - sink block, `both` - transient block),
- `NumberOfInputs` – number of input ports,
- `NumberOfOutputs` – number of output ports,
- `BlockType` – type of the block,
- `title` – title used in modal window

The following parameters define the modal input form and its contents.

- `type` – type of a form element (`text`, `checkbox`, `selectbox`),
- `data` – attributes of form (`name`, `id`, `size`, `value`.

```
{
 "DigitalClock": [{
   "io": "out",
   "NumberOfInputs": 0,
   "NumberOfOutputs": 1,
   "BlockType" : "DigitalClock",
   "title": "Source ...: Digital Clock"
 },{
   "type": "text",
   "data": {
      "title": "Sample time",
      "Name": "SampleTime",
      "size": "10",
      "id": "SampleTime",
      "value": "1"
   }
 }
]}
```

Information about the appearance of block is stored in `display.json`. These are split into:

- `type` – shape of block (e.g. `rect` - rectangle),
- `Text` – text that appears inside the block,
- `Name` – title located next to the block,
- additional data such as dimensions, position, line type, etc.

```
{
 "DigitalClock": [{
   "type": "rect",
   "data": {
     "width": 60,
     "height": 30,
     "fill": "white",
     "left": 0,
     "top": 0,
     "stroke": "black"
   }
 },{
   "type": "text",
   "Text": "12:34",
   "data": {
     "fontFamily": "Arial",
     "left": 14,
     "top": 10,
     "fontSize": 10,
     "fill": "black"
   }
 },{
   "type": "name",
   "Text": "Digital Clock",
   "data": {
     "fontFamily": "Arial",
     "left": 1,
     "top": 32,
     "fontSize": 8,
     "fill": "black"
   }
 }
]}
```

B. Application Usage

The main application is shown in Fig. 2. The user interface contains the library of blocks (left-hand side) and canvas area (right-hand side) where user created and arranges a scheme.

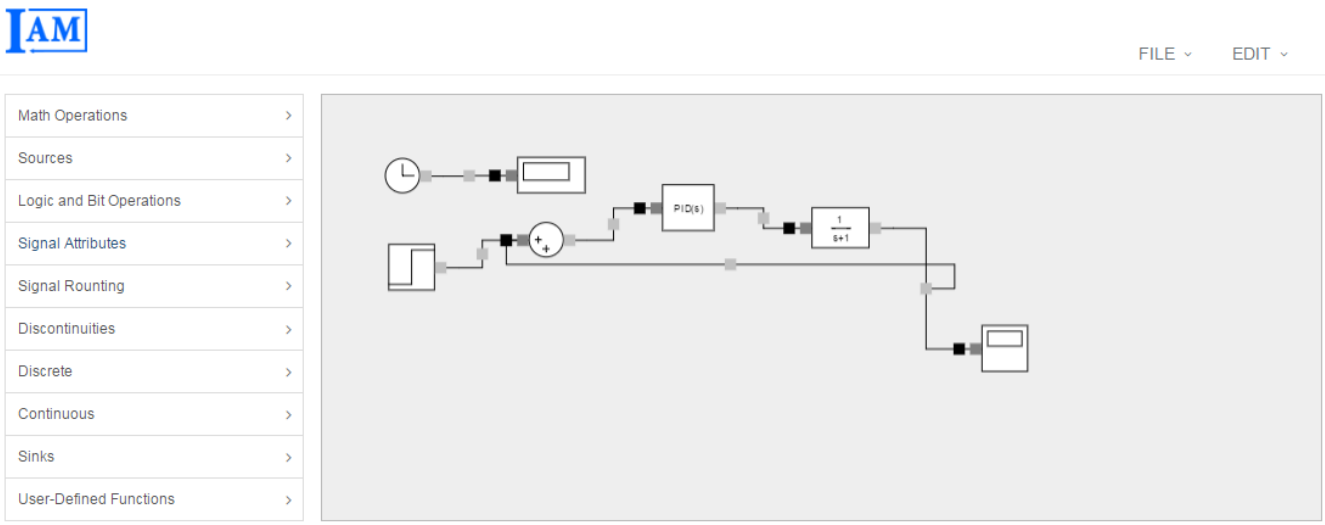

Fig. 2: Main page of the application

The library is loaded dynamically from server. For each type of defined block a `display.json` is loaded. Based on this information, the blocks are rendered and slip into categories.

The copying of block from the library into the canvas is performed via double-click action. Another double-click on a newly created block triggers its modal window, where parameters can be set (Fig. 3). This window contains form that is generated using the `param.json` information. This dynamic action is performed asynchronously via AJAX calls.

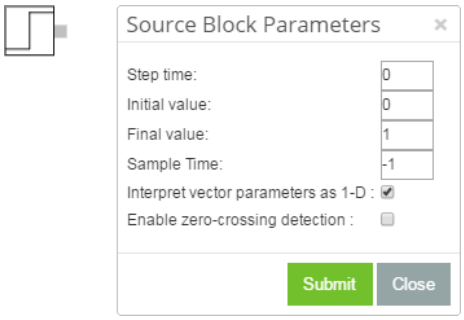

Fig. 3: Change of block's parameters

The interconnection of two unconnected blocks (Fig. 4a) is created by dragging of line from the output port of first block (silver square - Fig. 4b) on the input port of second block (gray square - Fig. 4c). The connection is automatically created on a mouse release action (Fig. 4d). Every connection line is defined by four points, and they are symmetric relatively to input and output port. A line contains square marks that allow a user to interact with it. Silver mark defines a port, where a new branch of the line can be specified. The black mark can be used for deleting the line.

Each block can be moved in the canvas using a drag and drop mechanism. All the lines are automatically adjusted if a block is relocated to another position. A separate movement of

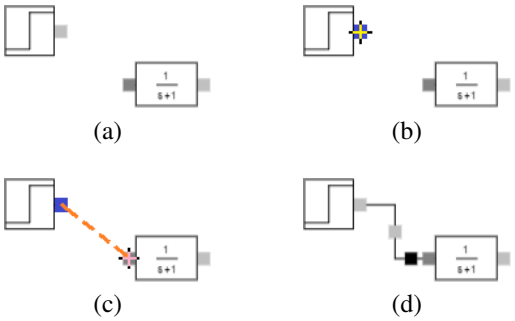

Fig. 4: Definition of interconnection between two points

lines is not available in the current version of the application, but it is considered for the future extensions.

A block can be deleted by selecting it and pressing either delete button on a keyboard or by clicking `Edit/Delete Object` in the top menu of the application. If the deleted block has any defined connections, these are deleted as well.

In the current version of the application, each block has only one input and one output port. If the user needs to connect multiple input/output signals, their number can be defined in the modal window of block's properties (Fig. 5), and only unoccupied ports are offered to the user.

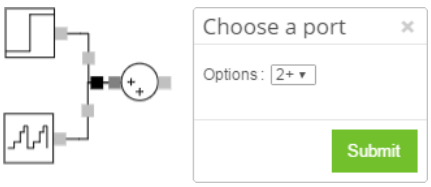

Fig. 5: Multiple-port connection

Created model can be saved in an SLX file format. In the menu of application user selects `Save Model` option. By

clicking on the save button, a new modal window appears with a link to generated model file. If the generated model contains at least one block from a set of `Sinks` (`Scope` or `To Workspace`), user is provided with the possibility to run the model simulation remotely on a server (by clicking the `Run` button). By doing so, the application stores a copy of SLX file in work directory of remote MATLAB and executes it by another M-file generated by the application. If model contains `Scope` block, user is provided with the resulted graph, generated using a JavaScript library Flot[6]. The application returns the data for graph generation in the form of a JavaScript array.

```
var data = [[t0, y0], [t1, y1], ...,
            ..., [tn, yn]];
```

The results are returned to a user only if the corresponding output block has selected the option `Save data to workspace`. The `Save name` option must have a unique name in the model. The structure of save data can be either of available formats (`Stucture with time`, `Stucture`, and `Array`). If the model contains at least one `To Workspace` block, then returned data are provided in the from of space-separated values.

```
t0  y0
t1  y1
 ...
tn  yn
```

These results are provided in the form element `textarea`.

IV. Illustrative Example

The example of application usage is shown on step responses of two systems defined in the form of transfer functions

$$G_1(s) = \frac{1}{s+1}$$
$$G_2(s) = \frac{2}{s^2+2s+1}.$$

A user creates a scheme in Fig. 6, based on the procedure provided in the previous section. The scheme contains two `Transfer Fcn` blocks, two `Scope` blocks and one `Step`.

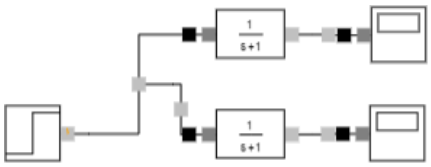

Fig. 6: Example model for step response generation

User defines the following parameters of blocks:

- Block `Step`:

```
Step time: 2
Initial value: 0
Final value: 1
```

- Block `Transfer Fcn 1`:

```
Numerator coefficcients: [1]
Denominator coefficcients: [1 1]
```

- Block `Transfer Fcn 2`:

```
Numerator coefficcients: [2]
Denominator coefficcients: [1 2 1]
```

- Block `Scope 1`:

```
Save data to workspace: checked
Save name: out1
Format: Stucture with time
```

- Block `Scope 2`:

```
Save data to workspace: checked
Save name: out2
Format: Array
```

The application generates the scheme that can be downloaded and opened in Simulink (Fig. 7). The scheme provided by the application is principally identical with the scheme in Fig. 6. The only difference is in the appearance of signal lines, however, the scheme provides the same functionality.

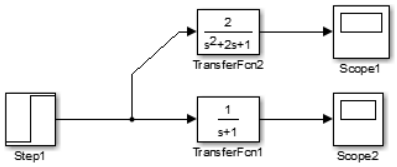

Fig. 7: Generated scheme opened in Simulink

Since the block `Scope` is present in the scheme, the user can run the simulation and get the results. The step responses of systems' dynamics is shown in a new window of Web browser (Fig. 8).

V. Conclusions and Future Work

This paper presents a tool for editing of Simulink schemes in the form of Web-based application that runs in an Internet browser. It allows users not only to design and create their own models but also to execute them remotely on a server and acquire results. The application can be used in the educational process, specifically in modeling/evaluation of dynamical systems, control design, and as additional tools for MATLAB-based virtual and remote laboratories. The application currently contains a library of 35 blocks.

The future work will be focused on the extension of block library, graphical enhancement, and addition of various interactive features (rotate & flip, zoom, rearrangement of connection lines, etc.).

The full application (including the server-side execution of Simulink models) is available only to students of the faculty

[6]http://www.flotcharts.org/

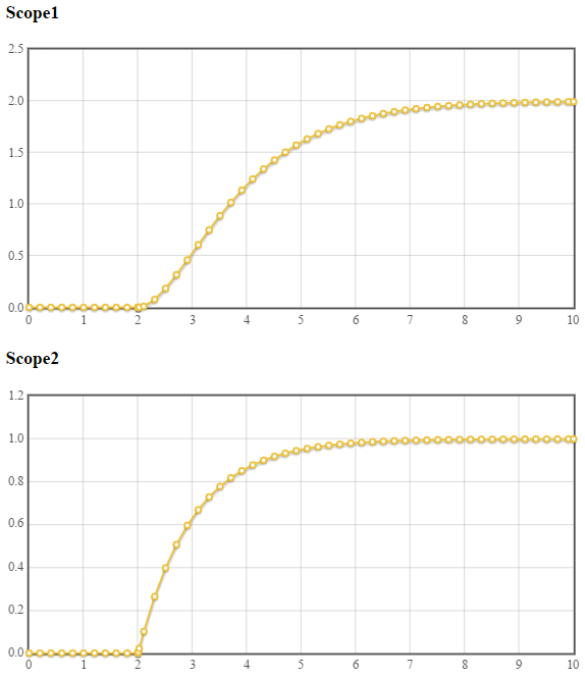

Fig. 8: Graphical result of simulation

for handling their study tasks (according to MATLAB license agreement). The part of the application that allows to create and edit Simulink models is publicly available at http://www.kirp.chtf.stuba.sk/~cirka/websimulink.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the contribution of the Scientific Grant Agency of the Slovak Republic under the grants 1/0004/17.

REFERENCES

[1] The Mathworks, Inc., Natick, Massachusetts, *MATLAB Web Server*, 2001.

[2] D. J. Kroon, "Web server." https://www.mathworks.com/matlabcentral/fileexchange/29027-web-server/, 2011. [Online; accessed 20-February-2017].

[3] M. Bakošová, M. Fikar, and Ľ. Čirka, "New approaches to control engineering education," *AT&P Journal*, pp. 17–19, 2003.

[4] M. Kvasnica and Ľ. Čirka, "Usage of milab in education of automatic control," in *Proceedings of the 13th International Conference Process Control '01*, Slovak University of Technology Bratislava, Slovak, 2001.

[5] I. Zolotová, J. Liguš, and A. Jadlovská, "Remote and virtual lab–cybervirtlab," in *Proceedings of the 17-th EAEEIE Conference on Innovation in Education Engineering, Craiova*, pp. 339–342, 2006.

[6] R. Puerto, L. Jiménez, and O. Reinoso, "Remote control laboratory via internet using matlab and simulink," *Computer Applications in Engineering Education*, vol. 18, no. 4, pp. 694–702, 2010.

[7] J. Oravec, M. Kalúz, Ľ. Čirka, M. Bakošová, and M. Fikar, "Webpiddesign for robust pid controller design," in *Proceedings of the 20th International Conference on Process Control* (M. Fikar and M. Kvasnica, eds.), (Štrbské Pleso, Slovakia), pp. 393–399, Slovak University of Technology in Bratislava, June 9-12, 2015 2015.

[8] J. Oravec, M. Kalúz, Ľ. Čirka, M. Fikar, and M. Bakošová, "Webpid-design - software for pid controller design management," in *European Control Conference 2015*, (Linz, Austria), pp. 3020–3025, 2015.

[9] Ľ. Čirka, M. Kalúz, and M. Fikar, "On-line remote control of matlab simulations based on asynchronous communication model," in *21th Annual Conference Proceedings: Technical Computing Prague 2013* (P. Byron, ed.), pp. 1–6, Institute of Chemical Technology, Prague, ICT Prague Press, 2013.

[10] Z. Janík, "Web-based modification of block schemes in matlab/simulink," Master's thesis, FEEIT STU in Bratislava, 2010.

[11] Z. Janík and K. Žáková, "Online design of matlab/simulink block schemes," *iJET*, vol. 6, no. VU 2009, pp. 11–13, 2011.

[12] Z. Janík and K. Žáková, "Online design of matlab/simulink and scilab/x-cos block schemes," in *Proceedings of the 14th International Conference on Interactive Collaborative Learning* (M. H. Michael E. Auer, ed.), (Piešťany, Slovakia), pp. 241–246, International Association of Online Engineering, Kirchengasse 10/200, A-1070, Wien, Austria, September 21 - 23 2011.