Slovak University of Technology in Bratislava
Institute of Information Engineering, Automation, and Mathematics

PROCEEDINGS

17th International Conference on Process Control 2009

Hotel Baník, Štrbské Pleso, Slovakia, June 9 – 12, 2009

ISBN 978-80-227-3081-5

http://www.kirp.chtf.stuba.sk/pc09

Editors: M. Fikar and M. Kvasnica

Čech, M., Balda, P.: A New Technique for Automatic Generation of Java Applets for Web-based Control Education, Editors: Fikar, M., Kvasnica, M., In *Proceedings of the 17th International Conference on Process Control '09*, Štrbské Pleso, Slovakia, 491–497, 2009.

Full paper online: http://www.kirp.chtf.stuba.sk/pc09/data/abstracts/040.html

A NEW TECHNIQUE FOR AUTOMATIC GENERATION OF JAVA APPLETS FOR WEB-BASED CONTROL EDUCATION

M. Čech * and **P. Balda** **

* *University of West Bohemia in Pilsen, Department of Cybernetics, fax : +420 377632502 e-mail: mcech@kky.zcu.cz*
** *e-mail : pbalda@kky.zcu.cz*

Abstract: The paper presents a new technique for automatic generation of Java source code for complex control algorithms simulation. The algorithms can be designed in graphical form either in Simulink model editor or in RexDraw, the graphical editor of the REX control system. The designed models are used for generation of Java source code files, which can be easily accomplished by graphical user interfaces and further used in both standalone Java applications and Java applets embedded into web pages. The proposed methodology can speed up the development of interactive control education tools because it interconnects Java with simulation and real-time domain. It is demonstrated on a closed loop simulation example using Pulse-step predictive controller and compared to earlier manual procedure.

Keywords: Web-based control education, interactive tools, Java applets, REX control system, Matlab-Simulink.

1. INTRODUCTION

Interactive education/demonstration tools based on Java applets embedded into web pages became very popular in all technical branches including automatic control (Schlegel and Čech (2004); Dormido (2002)) and physical modelling (Esquembre (2005)). The popularity grows up for many reasons: Java applets are self-contained, thus one need not to install additional development or runtime tools [1]. Moreover, Java is a multi-platform language. Therefore, the same applet can be started in any operating system (Windows, Linux, MacOS) directly from web page shown in the browser. Java applications are also portable to pocket PC's and mobile phones through Java ME (Micro Edition). It could be approved that these tools attract more web traffic and help to make your ideas familiar (Sirovich and Darie (2007)). Unfortunately, there is still a lack of such web simulation tools that have closer relation to real-time applications. It is caused by the fact that the *simulation* environment and *real-time* platform usually does not support straightforward migration to Java.

In the past, the connection [2] between simulation and real-time platform was solved for example by the REX control system (Balda et al. (2005)) or by Matlab Real-time Workshop. They both allow the user to generate real-time code directly from Simulink which is nowadays a standard tool for system modelling and simulation. In authors' previous paper (Balda and Čech (2006)) the client Java interface to REX was presented. The Java application or applet could contain a TCP/IP client and read data or parameters from any

[1] We assume, that the common JRE (Java Running Environment) is installed.

[2] It does not mean that more platforms are running simultaneously, although it is under some limitations possible, especially when Java is a remote visualization tool for real-time target.

REX target device. This simple interface was used together with *Easy Java Simulations* (EJS) to create web pages with remote access to several models in process control laboratory (Balda and Princ (2007)).

In this paper, the previous results are extended in order to support creation of self-contained Java applets for complex control algorithm simulation. Therefore, the following components were transformed into Java

- Simple simulation core
- Main part of RexLib
- Server side of REX interface

where the particular instance of simulation core (task) is generated automatically from Simulink or RexDraw model and the RexLib blocks Java classes are generated from their C-code originals. By this new set of components, Java is naturally joined up with simulation (Simulink) and real-time platform.

The paper is organized as follows: Section 2 introduces the overall structure of Java packages collectively called JavaREX and refers to its most important classes. Section 3 focuses on the automatic creation of Java function block library. The generation of the particular simulation task from Simulink model is outlined in Section 4. Illustrative example of Pulse-step predictive controller (further PSMPC) in the closed loop is given is Section 5 where it is compared to older hand procedure used for PIDlab applets development. Section 6 contains concluding remarks and ideas for future work.

2. JavaREX STRUCTURE

At present, we denote JavaREX the group of Java packages that help us to bridge the gap between interactive virtual tools and real-time applications, in particular based on REX. JavaREX key parts and classes are described in the following subsections. Overall JavaREX structure is also sketched in Fig. 1.

2.1 *Global part - package xGlobal*

The package `xGlobal` contains classes which are common for other modules. Roughly speaking, basic data types, structures, constants and error codes are defined here.

2.2 *Simple simulation core - package xCore*

REX contains a powerful hard real-time core with scheduler which allows to run number of tasks at different priority levels and sampling periods. This core was reduced for Java and web purposes. Java simulation core enables the user to create only a single task with arbitrary number of function blocks and their connections. Consequently, the task inserted into executive can be run as a Java thread with given period (soft real-time). Note, that for the demonstration purposes the thread period may differ from task sampling period used for discretization of each block. Although a specific task class can be coded manually, for more complex control schemes it is necessary to generate a task source automatically from Simulink or RexDraw editor as described in Section 4. Then a consistence of block names, connections and variable [3] values is guaranteed between Simulink, real-time and Java platform. Moreover, it is possible to simulate and debug the control algorithm before Java code generation.

The most important Java classes are [4]

`Block` – superior abstract class for all function blocks and task as well. Contains `init()`, `main()` and `exit()` methods which must be implemented in all inherited classes. Each block has also a given name, sampling period, task owner reference and fixed number of inputs, outputs, states and arrays.

`Task` – extends `Block` class. In addition it contains a sequence of blocks representing a control algorithm. The `init()`, `main()` and `exit()` methods are just calling appropriate methods of each block in the sequence. In inherited class generated from the model, the block instances are created in `init()` using `declareBlock()`, `addBlock()` and `connectBlock()` methods. Fortunately, the usage of those methods is transparent when the task is generated automatically (see Section 4).

`XExecutive` – extends Java `Thread` class. The `Task` instance must be inserted into the constructor. After calling `initExecutive()` the thread can be started calling `start()` method inherited from `Thread`. `XExecutive` performs periodical execution of task `main()` method until `stopExecutive()` or `pauseExecutive()` is called. The first method leads to re-initialization of all task blocks and internal variables while the paused executive can be run again from the halted state by `continueExecutive()`.

[3] By variable we understand any block input, output, state, parameter or array/buffer.
[4] This is just a brief description of the main features, not a full documentation

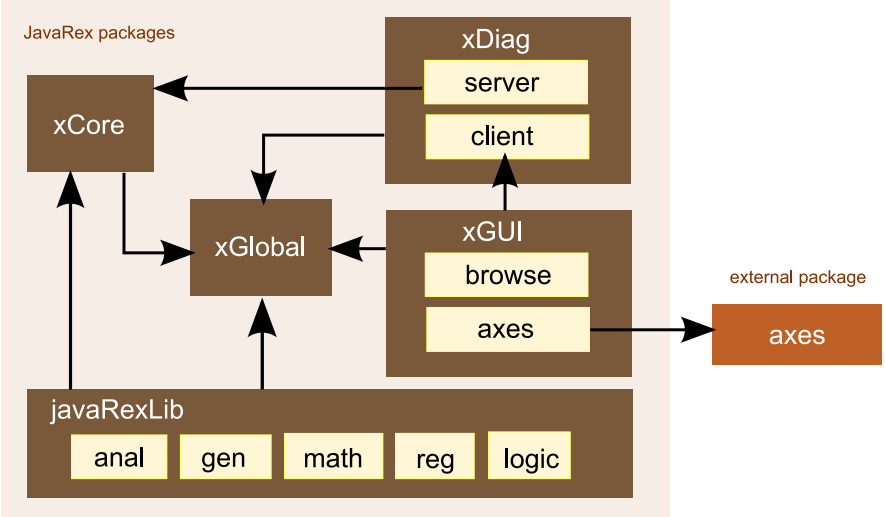

Fig. 1. Internal structure and dependencies of JavaREX packages.

2.3 *Function blocks library - package JavaRexLib*

The REX control system contains a large library (RexLib) of function blocks suitable for process control and mechatronics. Significant part of mentioned library was transformed form C++ into Java language using special technique described in Section 3. Thanks to this automatic generation, the consistency between simulation, real-time and Java platform is ensured including all block internal algorithm, variables names, types, initial values, configuration flags, etc. On the contrary to task generation, the library (further called JavaRexLib) is generated and compiled only once. Hence, the user need not to take care about a source code of individual function blocks. JavaRexLib together with automatic task generation guarantee that exactly the same control algorithms are executed on all platforms mentioned above.

The block library is divided into several sub-packages taking into account an original RexLib structure. Each block is inherited from the abstract `Block` class which is accomplished by definition of inputs, outputs, states and arrays (buffers). Each variable contains configuration flags defining especially LO/HI limits and solving automatic data type conversion in input-output block connections. Naturally, also a `main()` method is implemented. For commented example of block source code see Fig. 2.

2.4 *Communication and diagnostic interface - package xDiag*

The client side of commutation interface was already described in authors' previous work Balda and Čech (2006). Now, the communication package called `xDiag` was completed by a server side. Such step is necessary to connect a client easily to the running `XExecutive` instance. By implementing server side of diagnostic interface, one can simply read or write values of any block variables in the task. As the communication is based on a standard TCP/IP socket the connection between Java and real-time platform can be established also remotely via Internet.

Let us turn the attention to the self-contained Java applets. In this case both parts of the interface are packed inside the application. The client part asks the server to read/write block variables while the server executes those commands and informs client part about the result. A separated thread created by user is then responsible for periodical generation of commands and updating of the application GUI.

2.5 *Graphical user interface - package xGUI*

Several graphical components suited directly to REX needs can be found in package `xGUI`. At present a programmer can use simple trend and tree component which allows to browse REX target structure.

3. JAVA VERSION OF RexLib

RexLib, which has been introduced in Schlegel et al. (2001), is a function block library of the REX control system (Balda et al. (2005)). At present, RexLib contains more than 130 various function blocks. The complete reference of these blocks is given in Controls (2008). Almost all blocks contained in sub-libraries `anal`, `gen`, `math`, `reg` and `logic` (altogether almost 90 function blocks) has been converted to Java.

Algorithms of function blocks in RexLib are written in ANSI C intensively using macros. This fact allows us to have only one source code for each target platform. C language MEX files are created for Matlab-Simulink, C++ class is created for each block in all ports of the REX control system. We insisted on having only one source code of the each particular block, but we cannot use preprocessor in Java (like in C and C++ languages) because Java has no one. Thus we had to develop a technique of automatic conversion of C language block source files to Java block classes. This technique is based on two ideas:

- C language source file preprocessing. Fortunately, Java has almost the same syntax of many statements as C/C++, e.g. expressions, assignements, `if` statement, `for` statement, etc. That is why we needed to preprocess (expand) mainly access macros to block inputs, outputs, states and working arrays. For performing of this task, we used the RexComp (REX compiler) program for each function block source code with the special command line parameter `-p` (i.e. *Preprocess*).
- Generation of special Java syntax by a specially written application. The files obtained (in the previous step) were not Java class compatible source files yet, so we had to modify them and add more information from the REX C++ implementation files. This task was provided by the program which uses the regular expression technique to find necessary information and to add it to the preprocessed files. This program has been developed in C# using .NET Framework regular expression classes.

The result of the ADD block (addition of two inputs) conversion is shown in Fig. 2. For such simple blocks it would be simpler to write the code manually. But it is not a case of advanced controllers which code is longer than thousand lines per block. The biggest advantage of this method is the possibility of code regeneration after any block change.

4. GENERATION OF SIMULATION TASK

Simulation task is automatically generated from function block diagram configured in Matlab-Simulink graphical editor or RexDraw, the drawing tool of REX. Code is generated by the RexComp compiler in a similar way as in Balda (2007) and further developed in Balda and Schlegel (2008).

The Java source code automatic generation is a two step process. In the first step, a meta code consisting of the macro sequence is generated using the RexComp command line option `-s`. The second step is preprocessing of the first step result file again by RexComp, this time with the `-p` command line option. The difference between the preprocessing in Balda (2007) and here is the usage of alternative macro definition file. The automatically generated code in Fig. 3 is described in the next section.

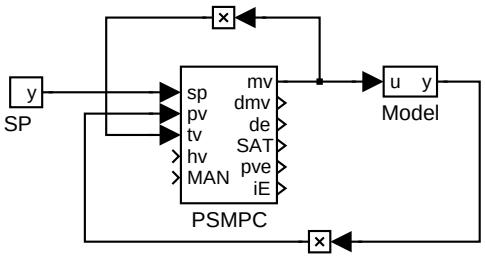

Fig. 5. Example of PSMPC demo drawn using RexLib blocks in Simulink editor.

5. EXAMPLES

5.1 *Example 1: manual way*

In the past, the process of block and task implementation into Java language had to be done manually. Although the implementation of `main()` block method is almost the same in both C++ and Java languages, the manual way is very time consuming and error-prone. Despite it, manual way was used to develop PIDlab interactive applets where the RexLib PIDU block is implemented (Fig. 4, Controls (2008)). The popularity of PIDlab applets was the motivation to create the automatic procedure which allows the programmer to focus on the the interactive GUI.

5.2 *Example 2: automatic way*

Let us show the process of creating Java education demo of pulse step predictive controller PSMPC (Controls (2008)):

- Assume that JavaRexLib was already automatically created using a procedure described in Section 3.
- Firstly, the simple model was drawn in Simulink editor (see Fig. 5). Note, that one can use full power of Simulink to debug and simulate the loop.
- After that, the model task Java code was generated as shown in Fig. 3.
- Finally, the generated task was compiled and used together with a prepared trend (package `xGUI.axes`) to created simple Java demo that can be embedded into a web page. It can be reviewed in Fig. 6.

```
public class ADD extends Block{
    public static XInInitVar[] xInInitVars = new XInInitVar[]{
        new XInInitVar("u1", -1,
                       new XInCfg((short)0, (short)XAV.XATM_XANY_VAR, XAV.MIN_DOUBLE, XAV.MAX_DOUBLE,
                       new XInVar(XAV.XATC_XDOUBLE))),
        new XInInitVar("u2", -1,
                       new XInCfg((short)0, (short)XAV.XATM_XANY_VAR, XAV.MIN_DOUBLE, XAV.MAX_DOUBLE,
                       new XInVar(XAV.XATC_XDOUBLE, (double) 0))), };

    public static XOutInitVar[] xOutInitVars = new XOutInitVar[]{
        new XOutInitVar("y", -1,
                       new XOutCfg((short)0, XAV.MIN_DOUBLE, XAV.MAX_DOUBLE,
                       new XOutVar(XAV.XATC_XDOUBLE))};

    public static XStatInitVar[] xStateInitVars = new XStatInitVar[]{};
    public static XArrInitVar[] xArrInitVars = new XArrInitVar[]{};

    public ADD(String name) {
        super(name);
        nInCount = (short) xInInitVars.length;
        nOutCount = (short) xOutInitVars.length;
        nStateCount = (short) xStateInitVars.length;
        nArrCount = (short) xArrInitVars.length;
        nExtInCount = 2;
    }

    public void init() throws JRexLibException {
    }

    public void main() throws JRexLibException, RexServerException {
        super.main();
        getOutAt(0).xDouble = getInAt(0).xDouble + getInAt(1).xDouble;
    }

    public void exit() throws JRexLibException {
    }
}
```

Fig. 2. Example of Java source code for simple block ADD generated automatically from C++ original.

6. CONCLUSIONS

Interactive education/demonstration tools based on Java applets embedded into web pages became very popular in all technical branches including process control. The paper describes a new JavaREX platform for automatic generation of Java source code for complex control algorithms simulation. The simulation models can be build from RexLib function blocks and simulated in standard Matlab/Simulink environment. Thanks to the automatic generation of Java simulation core and REX system features one can guarantee the control algorithm consistence at all simulation, presentation (Java) and real-time platforms. The described methodology was demonstrated on a closed loop simulation example using Pulse-step predictive controller and compared to older manual procedure. The authors believe that JavaREX platform will significantly speed up the development of interactive web-based education tools and support the technology transfer. The future work involves creation of basic components that can be easily connected to the chosen variable and which will support automatic creation of communication groups.

ACKNOWLEDGMENTS

The work has been supported by the Czech Ministry of Industry and Trade, project No. FI-IM5/030. This support is very gratefully acknowledged.

References

P. Balda. Automatic conversion of advanced control algorithms of the REX control system to various embedded platforms of microcontrollers. In *Automatizace, regulace a procesy (in Czech)*, pages 47–54, Prague, 2007. Dimart.

P. Balda and M. Princ. Remote laboratory experiments based on easy java simulations and rex. In *Process Control 2007 Summaries Volume*, Strbske Pleso, Slovak Republic, June 2007 2007.

P. Balda and M. Schlegel. BRRexLib – Function block library for B&R Automation PLCs. In *Process Control 2008*, pages 1–7, Kouty nad Desnou, 2008. University of Pardubice.

P. Balda and M. Čech. Java interface to REX control system. In *Process Control 2006 Summaries Volume*, Kouty na Desnou, Czech Republic, June 2006.

```
public class TaskExample extends Task{

    public TaskExample(String name) { // class constructor
        super(name);
    }

    public void init() throws JRexLibException, RexServerException{

        // Blocks declaration, allocation. By declareBlock() the block 'tels' the task number of its variables
        CNR SP      = new CNR("SP");      declareBlock(SP);
        PSMPC PSMPC = new PSMPC("PSMPC"); declareBlock(PSMPC);
        MDL Model   = new MDL("Model");   declareBlock(Model);

        // the task allocates arrays of inputs, outputs, states, and buffers for all declared blocks
        allocateMemory();

        // in addBlock(), the block inputs, outputs, states and buffers are created and inited
        addBlock(SP);
        SP.setParamDouble(0, 5.0);
        SP.setPeriod(0.01);
        SP.init();
        addBlock(PSMPC);
        PSMPC.setParamDouble(0, 1);
        .... // for easy reading, the remainig  PSMPC params were removed from code listing
        PSMPC.setPeriod(0.01);
        PSMPC.init();
        addBlock(PSMPC);
        Model.setParamDouble(0, 10);
        .... // for easy reading, the remainig  MDL params were removed from code listing
        Model.setPeriod(0.01);
        Model.init();

        connectBlock(0,0,1,0);
        connectBlock(1,0,2,0);
        connectBlock(2,0,1,1);
        connectBlock(1,0,1,2);

        this.setPeriod(0.01);
    }
}
```

Fig. 3. Example of automatically generated task for PSMPC demo shown in Fig. 5.

P. Balda, M. Schlegel, and M. Štětina. Advanced control algorithms + Simulink compatibility + real-time OS = REX. In *Proceedings of IFAC 2005*, Praque, Czech Republic, July 2005.

REX Controls. *Function blocks of the REX system (Czech)*. http://www.rexcontrols.cz, 2008.

S. B. Dormido. Control learning: Present and future. In *IFAC 15th Triennial World Congress*, Barcelona, 2002.

F. Esquembre. *Easy Java Simulations - The Manual*. 2005. Available at www.um.es/fem/Ejs.

M. Schlegel and M. Čech. Internet PID controller design: www.PIDlab.com. In *Proceedings of IBCE 04*, pages 1–6, Grenoble, France, September 2004.

M. Schlegel, P. Balda, and M. Štětina. C mex blockset for industrial automation. In *Proceedings of the 9th Matlab Conference (in Czech)*, pages 361–369, Prague, 2001. Vydavatelství VŠCHT.

J. Sirovich and C. Darie. *Search Engine Optimization for PHP*. Wiley Publishing, Inc., 2007.

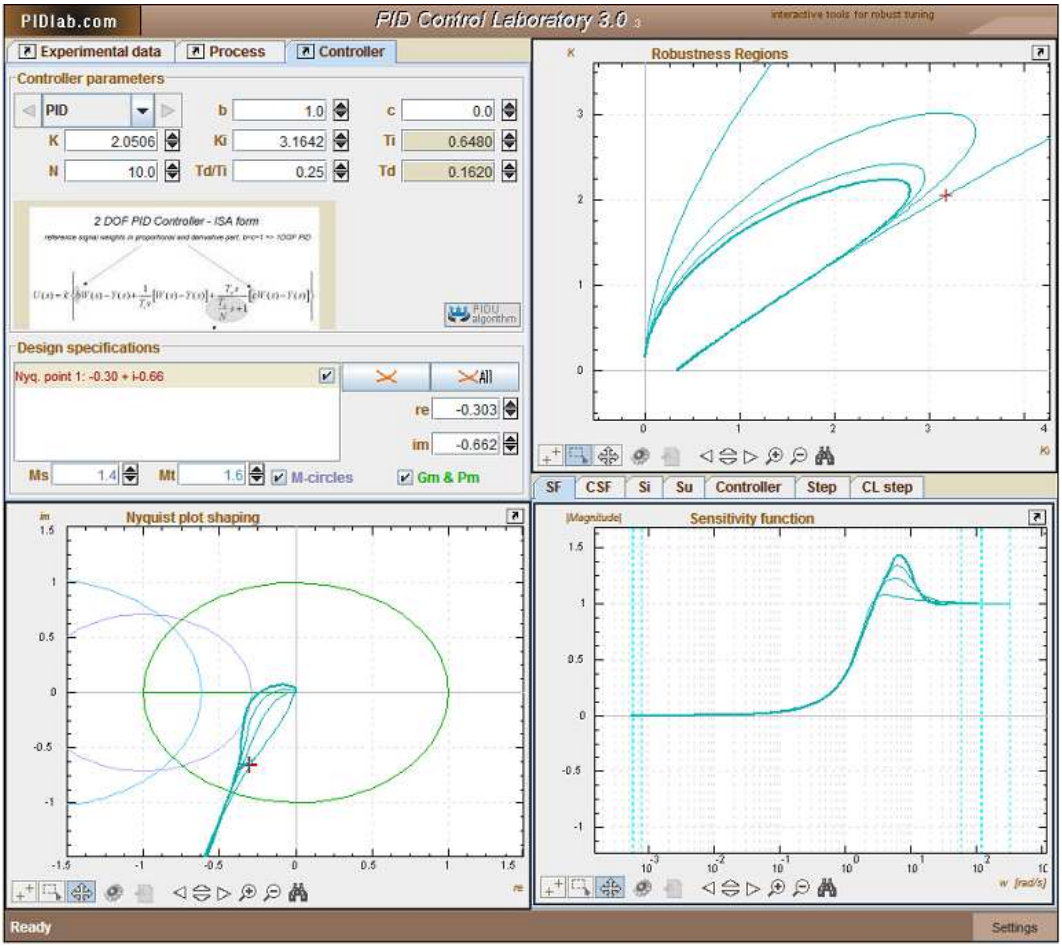

Fig. 4. Example of PIDlab applet for PID controller tuning. This applet adopts RexLib PIDU algorithm which was transformed manually into Java.

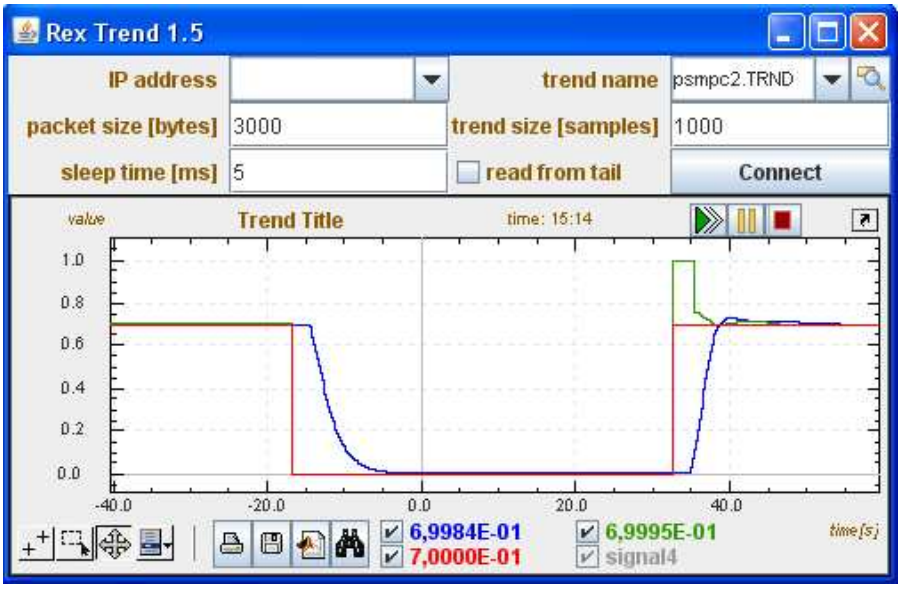

Fig. 6. Example of PSMPC demo using JavaREX trend from package `GUI.axes`.