PoST2ST: a web service for translating poST programs to the IEC 61131-3 Structured Text

Vladislav Bashev
Institute of Automation and Electrometry
SB RAS
Novosibirsk, Russia

Andrei Rozov
Institute of Automation and Electrometry
SB RAS
Novosibirsk, Russia

Vladimir Zyubin
Institute of Automation and Electrometry
SB RAS
Novosibirsk, Russia

***Abstract*—This paper describes a web application (http://post2st.iae.nsk.su) for translation of the poST (process-oriented Structured Text) language, a process-oriented extension for the ST (Structured Text) language from the IEC 61131-3 language set. The poST language, developed at the Institute of Automation and Electrometry SB RAS, is a language for describing programs for cyber-physical systems implemented on the basis of PLCs. The service allows one to create small programs in the poST language without installing software on the user's side, which simplifies the initial acquaintance of users with the poST language. The service provides checking the syntax and semantics rules of programs, and generating ST code and PLCopen XML Exchange format, which can be used in existing IEC 61131-3 development tools such as CoDeSys. The relevance of such a service is substantiated, its graphical interface and functionality are described.**

***Keywords*—web application, process-oriented programming, PLC languages, IEC 61131-3, Structured Text**

I. Introduction

The growing complexity of modern control systems and the tendency for their widespread use motivates researchers to develop new approaches to their design, programming and building.

The isolation of this field of research from computational programming is due to the following features. Software employed in control systems is open (i. e. communicate with an external environment), event-driven, and concurrent (have to process a set of independent events occurring in the external environment).

An approach aimed at addressing these features of control software has been developed within the process-oriented paradigm [1].

Process-oriented programming (POP) involves specifying control software with a set of concurrently running processes. Internally the processes have a state-machine-like structure and are equipped with operations for managing time intervals and inter-process communication. Concurrent behavior of the system is arranged via consequent execution of active process states on each program cycle. Compared to other known state-machine-based approaches, such as CSP [2], Input / Output Automata [3], Harels State-charts [4], Hybrid Automata [5], Esterel [6], Calculus of Communicating Systems [7], and their extensions [8, 9], the POP approach combines system concurrency on the global scale with local linearity of behavior within each process.

This work was supported by the Russian Ministry of Education and Science, project no. AAAA-A19-119120290056-0.

POP provides a conceptual basis for multiple domain-specific programming languages (DSLs) that are intended for natural control software specification.

Within this paradigm new C-like languages Reflex and IndustrialC [10, 11] have been developed along with the Pascal-like language poST [12]. The Reflex language targets PC-based control software for large-scale industrial applications, while IndustrialC is tailored for microcontroller-based embedded systems.

As practice shows, the Reflex language can be successfully used in industrial applications and offers a number of significant advantages in control software programming [13, 14, 15]. However, its widespread use in practice is hindered by the conservative nature of the domain. The developer community tends to be wary of introducing any new emerging technology to the process. Historically, the majority of control software is still implemented within the so called PLC-approach, that is based on the IEC 61131-3 languages [16], and PLC manufacturers are reluctant to deviate from this standard.

The IEC 61131-3 includes two textual languages (assembler-like Instruction List and Pascal-like Structured Text), and three graphical languages (relay racks imitating Ladder Diagram, Function Block Diagram based on the data flow control approach, and Petri-net based Sequential Function Chart).

None of the IEC 61131-3 languages use C-like notations, and an attempt to include a C-like language in the IEC 61131-3 standard will fail even if the language is an extremely powerful one.

In order to face this challenge we proposed to adapt the process-oriented approach for the ST procedural programming language in the same way as it was done for the C language in case of Reflex. The process-oriented extension of ST was called the poST language.

The poST language can therefore be of particular interest to the PLC community as it extends the Structured Text language from IEC 61131-3. The additional attractiveness of the poST language is due to the wide popularity of the ST language. According to the CoDeSys GmbH (former 3S-SmartSoftware Solutions GmbH), the ST language is regularly used by up to 70% of users, and the number is constantly growing [17].

The poST language combines advantages of the process-oriented paradigm with conventional syntax of the ST language and can be easily adopted by the PLC community. The poST language assumes that a poST-program is a set of weakly connected concurrent processes, structurally and functionally corresponding to the technological description of the plant. Each process is specified by a set of states. The states are specified by a sequence of the ST constructs, extended by TIMEOUT operation, SET STATE operation, and START / STOP / check state operations to communicate with other processes.

For the poST language we have already developed an Eclipse-based IDE [18], including a parser and syntax-directed editor. Code generation modules for the C and ST languages have been implemented. The generated ST-code can be automatically converted in the PLCopen XML Exchange format [19], which makes integration with the IEC 61131-3 tools easier. The approach assumes that the generated code will be translated to executable form and uploaded to the target platform with an existing C or IEC 61131-3 toolchain.

Therefore, aside from the standard toolchain, the user currently needs to install the poST IDE on their local computer. This additional required effort limits the number of users who would try to familiarize themselves with the poST language.

To solve this problem, we propose to use network technologies and provide web applications for translating poST language into ST language and PLCopen XML Exchange format.

In the first part of the paper, the advantages of poST are briefly described, the specifics of programming on existing IEC 61131-3 tools are analyzed, and requirements for the functionality of a web application are formulated. The second part describes the architecture of the web application being developed. The third part is devoted to the implementation, interface and functionality of the web application.

II. Requirements

Currently, the issue of integrating the poST language into the CoDeSys environment [20] is under research. We developed a translator into the ST language and a wrapper of the ST-files to the PLCopen XML Exchange format. We are using DSL technologies [21] based on the Eclipse / Xtext framework [22]. As a result we have poST IDE and poST to ST jar-translator, which is an independent java program [23] to be used in console mode.

The following requirements were formulated:

(1) the functionality of the application should be accessed by the user through a web browser with an active network connection only

(2) web browser independence of the web application

(3) user interface should provide:

  (a) poST examples and patterns

  (b) editor window for poST-program

  (c) uploading poST program from the user local PC into the editor window

  (d) downloading the poST-program, generated ST-programs, and PLCopen XML Exchange file

  (e) window for the translator error and warning messages

  (f) user feedback form

(4) independence from the specific IEC 61131-3 IDE.

III. PoST2ST Web Application Architecture and Implementation

A. Architecture

The client-server architecture [24] of the poST2ST web application is depicted in Fig. 1. The client-server interaction is based on the standard HTTP protocol [25] with HTML [26].

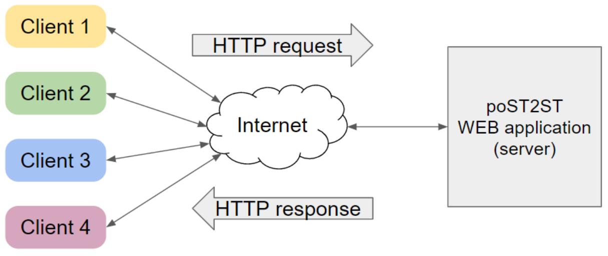


Fig. 1. Web application architecture

The web application is deployed on the following address: http://post2st.iae.nsk.su.

B. Client Side

The client side of the web application is represented by the user interface (Fig.2). The user interface is displayed in a browser. Upon accessing the resource's network address, the browser receives and interprets the HTML/CSS/JavaScript code of the start page.

The interface (Fig.2) has three text windows:

(1) poST window for the editable poST source code

(2) ST window for the resulting ST code

(3) Message window for diagnostic messages from the translator.

At the upper right part (above the windows) of the page there are the “About” button and the link to the site of the Institute of Automation and Electrometry of the SB RAS.

At the bottom right part (under the windows) of the page there is a block of buttons (Fig.3), consisting of three columns:

(1) “Translate to” and “Open” column with the following buttons:

  (a) “Translate” button to start poST to ST translation

  (b) “Choose file” button to select a poST file from the user's local PC

  (c) “Open” button to upload the selected poST-program in the window

(2) “Downloads” column with the following buttons:

(a) “poST code” button to download the source poST-program

(b) “ST code” button to download the resulting ST code

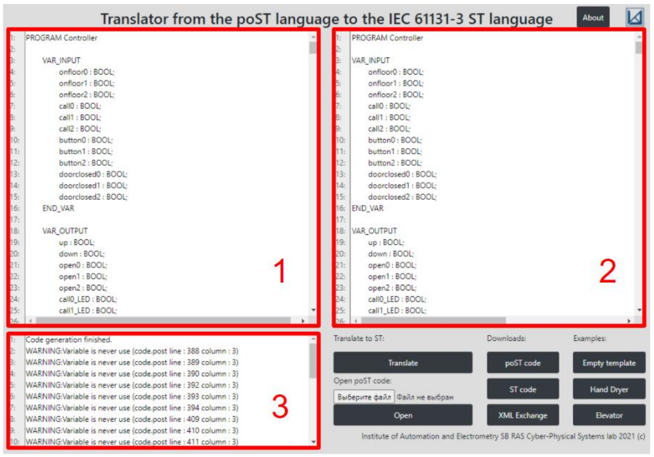
(c) “XML Exchange” button to download the resulting XML file (PLCopen XML Exchange format).

(3) “Examples” column with the following buttons:

(a) “Empty template” button to initialize the poST window with poST program template

(b) “Hand Dryer” button to initialize the poST window with the Hand Dryer poST program

(c) “Elevator” button to initialize the poST window with the Three-floor elevator poST program



```
PROGRAM Controller

    VAR_INPUT
        onfloor0 : BOOL;
        onfloor1 : BOOL;
        onfloor2 : BOOL;
        call0 : BOOL;
        call1 : BOOL;
        call2 : BOOL;
        button0 : BOOL;
        button1 : BOOL;
        button2 : BOOL;
        doorclosed0 : BOOL;
        doorclosed1 : BOOL;
        doorclosed2 : BOOL;
    END_VAR

    VAR_OUTPUT
        up : BOOL;
        down : BOOL;
        open0 : BOOL;
        open1 : BOOL;
        open2 : BOOL;
        call0_LED : BOOL;
        call1_LED : BOOL;
```

```
PROGRAM Controller

VAR_INPUT
    onfloor0 : BOOL;
    onfloor1 : BOOL;
    onfloor2 : BOOL;
    call0 : BOOL;
    call1 : BOOL;
    call2 : BOOL;
    button0 : BOOL;
    button1 : BOOL;
    button2 : BOOL;
    doorclosed0 : BOOL;
    doorclosed1 : BOOL;
    doorclosed2 : BOOL;
END_VAR

VAR_OUTPUT
    up : BOOL;
    down : BOOL;
    open0 : BOOL;
    open1 : BOOL;
    open2 : BOOL;
    call0_LED : BOOL;
    call1_LED : BOOL;
```

```
Code generation finished.
WARNING:Variable is never use (code.post line : 388 column : 3)
WARNING:Variable is never use (code.post line : 389 column : 3)
WARNING:Variable is never use (code.post line : 390 column : 3)
WARNING:Variable is never use (code.post line : 392 column : 3)
WARNING:Variable is never use (code.post line : 393 column : 3)
WARNING:Variable is never use (code.post line : 394 column : 3)
WARNING:Variable is never use (code.post line : 409 column : 3)
WARNING:Variable is never use (code.post line : 410 column : 3)
WARNING:Variable is never use (code.post line : 411 column : 3)
```

Fig. 2. User’s interface

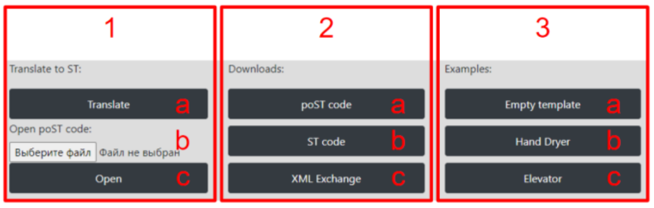


Fig. 3. Block of buttons

The user’s interface is specified in HTML using Bootstrap CSS styles [27] and JavaScript for line numbering.

C. Server Side

Upon first access, the server opens a unique session for the user. For this session, the server generates a unique random UUID [28] and allocates disk space for the temporary user's directory. This UUID is saved in encrypted form as a client's cookie [29] and is used to identify the current working directory for the user (Fig. 4).

The user directory contains:

(1) the poST code file

(2) the ST code file (if generated)

(3) the PLCopen XML Exchange format file (if generated)

(4) the translator log-file (if generated).

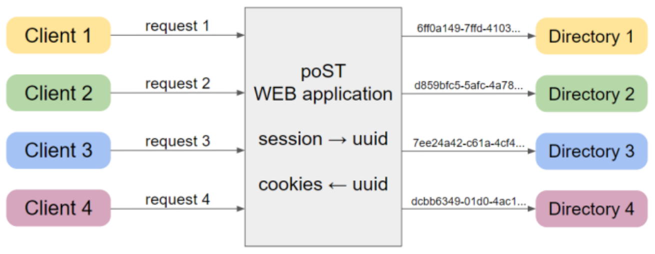


Fig. 4. A mechanism for identify a user to a directory

At the beginning the server generates the initial HTML-code for the client. If the poST, ST and log files exist, the server inserts them into the initial HTML-code.

The implementation of the server side of the application is based on the Flask framework [30].

Upon the user’s command, the browser sends a request to the server, which contains the command identifier and data. The request from the client side is passed to the user request processing module (URPM). Upon receipt of the request, the URPM processes it depending on the command.

Upon the "Translate" command, the browser sends the edited poST-code within the request to the server. The “Translate” URPM copies the file to the directory and executes the “Traslate” Bash-script [31] through Shell. The script sets the user’s directory, redirects the output of the poST to ST jar-translator to the log-file, calls the poST to ST jar-translator for the poST-code. The poST to ST jar-translator generates the resulting ST-code, XML-file, and log-file. Once the bash script finishes, URPM reads the files, generates the HTML-page and sends it to the client (Fig. 5).

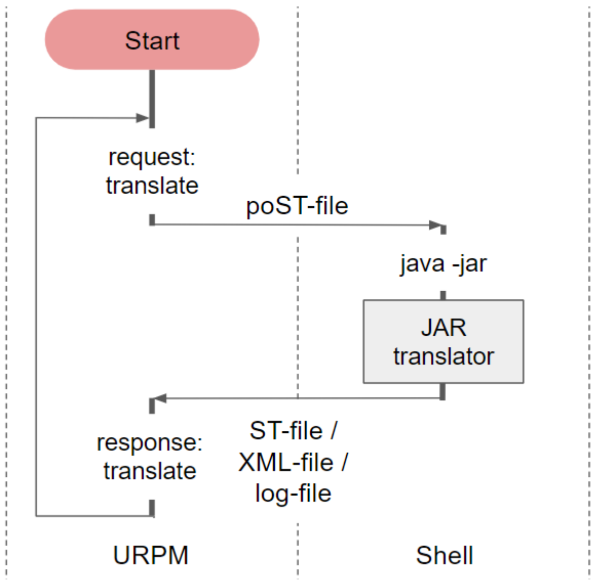


Fig. 5. User’s interface

The "Download poST", "Download ST", and "Download XML" URPMs are implemented by the standard browser functions. The URPMs save the files from the server.

The "Empty template", "Hand Dryer", "Elevator" URPMs use the corresponding files that are stored on the server side.

The poST to ST jar-translator is executed using JVM 1.8 under Linux / Ubuntu.

The source codes of the project can be found through the github repository [32].

IV. Conclusion

This paper describes a web application for translation of the poST language to the IEC 61131-3 ST language. The service allows one to create small programs in the poST language without installing software on the user's side, which simplifies the initial acquaintance of users with the poST language. The service provides checking the syntax and semantics rules of programs, and generating ST code and PLCopen XML Exchange format, which can be used in existing IEC 61131-3 development tools such as CoDeSys.

The service is implemented with client-server architecture using the Flask framework and based on the poST to ST jar-translator obtained in the Eclipse/Xtext framework.

The poST to ST jar-translator is executed using JVM 1.8 under Linux / Ubuntu.

The web application allows users to test the poST language without installing additional software.

We plan to use the proposed approach for promotion and testing of multiple other tools developed within future research projects.

Acknowledgment

Authors are very grateful for the charitable support they received from the JetBrains Foundation.

References

[1] V. E. Zyubin, "Hyper-automaton: a Model of Control Algorithms," Siberian Conference on Control and Communications, Tomsk, 2007, pp. 51-57, doi: 10.1109/SIBCON.2007.371297.

[2] C. A. R. Hoare, "Communicating Sequential Processes," Prentice-Hall Int., 1985.

[3] N. Lynch, M. Tuttle, "An Introduction to Input/Output Automata," CWI Quarterly, 1989, vol. 2, pp. 219–246.

[4] D. Harel, "Statecharts a Visual Formalism for Complex Systems," Science of Computer Programming 8. Elsevier Science Publishers B.V., North-Holland, 1987, pp. 231–274.

[5] R. Milner, "Communication and Concurrency," Series in Computer Science. Prentice Hall, 1989.

[6] G. Berry, "The Foundations of Esterel," Proof, Language, and Interaction: Essays in Honour of Robin Milner, MIT Press, Foundations of Computing Series, 2000, pp. 425–454.

[7] D. K. Kaynar, N. Lynch, R. Segala, F. Vaandrager, "Timed I/O Automata: A Mathematical Framework for Modeling and Analyzing Real-Time Systems," Proc of 2003 IEEE 24th International Real-Time Systems Symposium (RTSS’03), IEEE Computer Society Cancun, Mexico, 2003, pp. 166–177.

[8] L. Kof, B. Schtz, "Combining Aspects of Reactive Systems," Proc. of Andrei Ershov Fifth Int. Conf. Perspectives of System Informatics. Novosibirsk, 2003, pp. 239–243.

[9] T. A. Henzinger, "The Theory of Hybrid Automata," Verification of Digital and Hybrid Systems. NATO ASI Series (Series F: Computer and Systems Sciences), Springer, Berlin, Heidelberg. 2000, vol. 170, pp. 265–292.

[10] I. Anureev, N. Garanina, T. Liakh, A. Rozov, H. Schulte, V. Zyubin, "Towards Safe Cyber-Physical Systems: the Reflex Language and Its Transformational Semantics," in International Siberian Conference on Control and Communications (SIBCON), Tomsk, Russia, 2019, pp. 1-6, doi: 10.1109/SIBCON.2019.8729633.

[11] A. S. Rozov, V. E. Zyubin, "Adaptation of the Process-Oriented Approach to the Development of Embedded Microcontroller Systems," Optoelectronics. Instrumentation and Data Processing, 2019, Vol. 55, No. 2, pp. 198–204. DOI: 10.3103/S8756699019020122

[12] V. Bashev, I. Anureev, V. Zyubin, "The Post Language: Process-Oriented Extension for IEC 61131-3 Structured Text," 2020 International Russian Automation Conference (RusAutoCon), Sochi, Russia, 2020, pp. 994-999, doi: 10.1109/RusAutoCon49822.2020.9208049.

[13] T. V. Liakh, A. S. Rozov, V. E. Zyubin, "Reflex Language: a Practical Notation for Cyber-Physical Systems," System Informatics, No. 12 (2018) pp. 85-104

[14] P. G. Kovadlo, A. A. Lubkov, A. N. Bevzov et al., "Automation system for the large solar vacuum telescope," Optoelectron.Instrument.Proc. 52, 187–195 (2016). https://doi.org/10.3103/S8756699016020126

[15] T. V. Liah and V. E. Zyubin, "The reflex language usage to automate the large solar vacuum telescope," 2016 17th International Conference of Young Specialists on Micro/Nanotechnologies and Electron Devices (EDM), Erlagol, Russia, 2016, pp. 137-139, doi: 10.1109/EDM.2016.7538711.

[16] "IEC 61131-3 Programmable Controllers – Part 3: Programming languages," International Standard, Second Edition, 2003.

[17] I. Petrov, R. Wagner, "Debugging applied PLC software in CoDeSys (part 3)," Industrial Automatic Control Systems and Controllers, 4, pp.34-36 NAUCHTEKHLITIZDAT, 2006

[18] J. Wiegand, "Eclipse: A platform for integrating development tools," in IBM Systems Journal, vol. 43, no. 2, pp. 371-383, 2004, doi: 10.1147/sj.432.0371.

[19] M. Marcos, E. Estevez, F. Perez, E. V. Der Wal, "XML exchange of control programs," in IEEE Industrial Electronics Magazine, vol. 3, no. 4, pp. 32-35, Dec. 2009, doi: 10.1109/MIE.2009.934794

[20] D. H. Hanssen, "Programmable Logic Controllers: A Practical Approach to IEC 61131-3 using CoDeSys," Wylie & Co., 2015.

[21] C. Schmitt, S. Kuckuk, H. Köstler, F. Hannig, J. Teich, "An Evaluation of Domain-Specific Language Technologies for Code Generation," 14th International Conference on Computational Science and Its Applications, Guimaraes, 2014, pp. 18-26, doi: 10.1109/ICCSA.2014.16.

[22] M. Eysholdt, H. Behrens, "Xtext: implement your language faster than the quick and dirty way," Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, OOPSLA, ACM, 2010, New York, USA, 307–309, doi: 10.1145/1869542.1869625.

[23] K. Arnold, J. Gosling, "The Java Programming Language," The Java Series, Addison-Wesley Publishing Company, Reading, Massachusetts, 1996.

[24] H. S. Oluwatosin, "Client-server model," IOSRJ Comput. Eng 16.1 (2014): 2278-8727.

[25] R. Fielding et al., "RFC2616: Hypertext Transfer Protocol--HTTP/1.1." (1999).

[26] D. Raggett, A. Le Hors, I. Jacobs, "HTML 4.01 Specification," W3C recommendation 24 (1999).

[27] S. Bhaumik, "Bootstrap essentials," Packt Publishing Ltd, 2015.

[28] P. Leach, M. Mealling, R. Salz. "A universally unique identifier (uuid) urn namespace." (2005): 1.

[29] J. S. Park, R. Sandhu. "Secure cookies on the Web," IEEE internet computing 4.4 (2000): 36-44.

[30] M. Grinberg, "Flask web development: developing web applications with python," O'Reilly Media, Inc.", 2018.

[31] C. Newham, B. Rosenblatt, "Learning the bash shell: Unix shell programming," O'Reilly Media, Inc.", 2005.

[32] Github project repository, "poST language translation web application", URL: https://github.com/Vlad264/flask_poST_webIDE