*Generating process diagrams for control software in the Reflex language*

Sofia Evgenievna Belenkaia   
*Novosibirsk state university*  
*Institute of Automation and Electrometry of the Siberian Branch of the Russian Academy of Sciences (IA&E SB RAS)*Novosibirsk, Russia  
s.belenkaia8@g.nsu.ru

Vladimir Evgenievich Zyubin   
*Institute of Automation and Electrometry of the Siberian Branch of the Russian Academy of Sciences (IA&E SB RAS)*Novosibirsk, Russia  
zyubin@iae.nsk.su

Andrei Sergeevich Rozov  
*Institute of Automation and Electrometry of the Siberian Branch of the Russian Academy of Sciences (IA&E SB RAS)*Novosibirsk, Russia  
rozov@iae.nsk.su

*Abstract*— Many domain-specific languages (DSL) are used in the sphere of industrial automation and embedded systems, e.g. process-oriented programming languages. Usually there are tasks of reverse engineering and code refactoring in a process of developing and supporting projects, so, we need documentation to the source code. Automatic documentation tool may decrease the time for creating diagrams and protect them from human’s mistakes. (human error)

In this work we analyze available graph description languages and present an approach to automatic generation of graphical documentation for specialized programming languages. The presented approach involves generating of diagrams as graph forms of the source code and using already available external tools for laying out, visualization and editing purposes. A software module was created for the Reflex programming language to generate a process data communication diagram, control diagram and states diagram. Also, parts of graphs that don’t share nodes are automatically written to different files. The module automates the creation of the graphical documentation and simplifies the support of Reflex projects.

Practical testing has shown that using the developed module helps to decrease the time expenses for creating process diagrams (from a few hours to a couple of seconds) and guarantees that there are no errors caused by human factor.

Keywords— control software, process diagrams, graphical documentation, process-oriented programming, code refactoring, reverse-engineering, code analysis

# Introduction

Software development projects can often benefit from reverse engineering techniques. Such methods may be utilized in code refactoring and documentation both in development and maintenance phases of a project. For commonly used object-oriented languages such as Java, C++, C#, etc. various software tools are available for automatic generation graphical documentation. In many cases these tools are integrated into the development environment for the language.

Developing code analysis tools in industrial automation, writing control software and programming embedded systems is of especially high interest for problem-oriented languages (DSL). In particular, this problem is relevant for Reflex process-oriented programming (POP) language [3][4][5], which is actively developed in the Institute of Automation and Electrometry of the Siberian Branch of the Russian Academy of Sciences. POP [1][2][3] is intended to write a control program as a set of intercommunicated processes. While process-oriented programming increases a quality of the control software, code analysis of programs provides an additional safety level. In current projects code analysis is performed manually, which takes a lot of time, and the resulting documentation is likely to contain errors because of human factor.

# Purpose of work and tasks

The purpose of this work was to (In this work we develop…) develop an approach to create software modules for the auto documentation and to develop a software module for building process diagrams according to the specification in the Reflex language for testing our method. To achieve that goal, we have set several tasks. Firstly, to analyze: specific features of process-oriented programming language Reflex, existing tools for generating diagrams from the source code, and types of notations for diagrams which are used for code analysis. To develop the module (to determine requirements for the software module, to develop diagram notation for showing communications between processes, to choose the file format for saving diagrams). Then, to develop an architecture of the software module, to implement it and to test the software module.

# Specific Reflex language features.

Control algorithm is represented by a hyperprocess. Program name in the Reflex language has to be written after the reserved word ‘program’. Then, the body of the program is put inside the braces and consists of attributes of the hyper process specification and of descriptions of processes (process definitions). Attributes may be input/output ports, a period of process starting cycle and constants.   
  
(describe -> define)

The processes are defined sequentially. The process, which was written first, is in the active state. All other processes in the first moment of running the program are in the passive states. State functions are defined sequentially inside the process body. State name is put(specified?) after the reserved word ‘state’. State body is described inside the braces and consists of events and reactions to events, which are defined by the standard C operators and some special Reflex operators.

Reflex language has such operators as start/stop <process name>, set state <state name> for changing the state of the current process, and timeout <time specification>. Process can import variables from another processes using operator ‘from proc <process name> <names of variables>’.

A Reflex program consists of descriptions of processes, which are defined by state machines. Processes run in the cooperative model of multithreading and communicate with each other via shared variables and special control statements (i.e. process can start and stop each other and themselves, and import variables from another processes). For more information see in [3][4][5].

# Analysis of existing tools for generating diagrams from code

We have found out that not all tools for generating diagrams from the source code for common programming languages have the ability for code generation, dynamic diagram building and hiding components for diagram, but all of them have the ability to modify diagrams. The results of our analysis of existing tools for diagram generation from the source code for common programming languages are shown in the Table 1.

1. tools for generate diagrams by code for common languages

| Tool’s name | Tool’s abilities | | | |
| --- | --- | --- | --- | --- |
| Code generation | ***Modify diagrams*** | ***Hiding diagram’s components*** | Dynamic building |
| Class Designer for Visual Studio | - | + | + | + |
| Astah UML | + | + | Not found | Not found |
| MagicDraw | + | + | Not found | + |
| Software Ideas Modeler | + | + | + | + |
| BOUML | + | + | + | - |
| Visual paradigm | + | + | Not found | Not found |
| Rational Rose | + | + | + | + |
| Enterprise Architect | + | + | Not found | Not found |
| IntelliJ Idea | - | + | + | + |
| Sybase PowerDesigner | + | + | + | - |
| NetBeans | - | + | - | - |
| Lab view | + | + | - | Not found |
| Altova UModel 2008 | + | + | Not found | Not found |

By analyzing software development tools for control software and embedded systems programming, we have found, that even among commercial tools, a very small percentage of them provide reverse engineering functionality. In General, reverse engineering is supported by object-oriented SCADA packages, which have little to no relation to the sphere of the control and embedded software.

# requirements to the software module

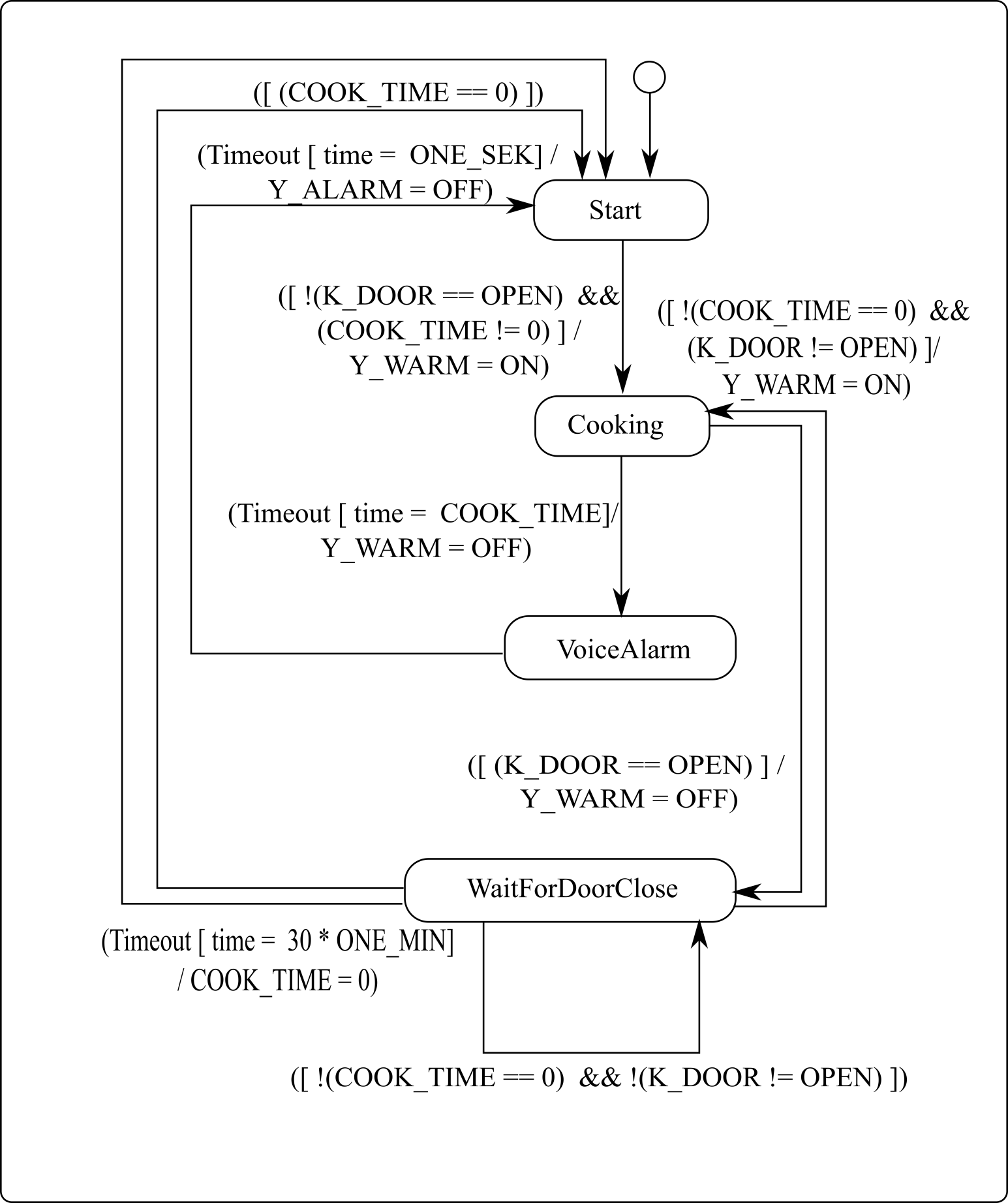
Requirements to the software module have been formulated based on the analysis results. The module needs to implement the following capabilities: (список лучше заменить на перечисление)

* Diagram modification;
* Automatic layout;
* Generation of the following diagram types:
  + Data communication diagram;
  + Process control diagram;
  + Processes states diagram;
* Saving diagrams to the file;
* Dividing diagrams into parts and saving them to different files if parts do not have any shared nodes.

# Process diagrams

## Process states diagram

For the process states diagram we use UML statecharts diagram [6][7]. An example of such diagram is shown on the Fig. 1.



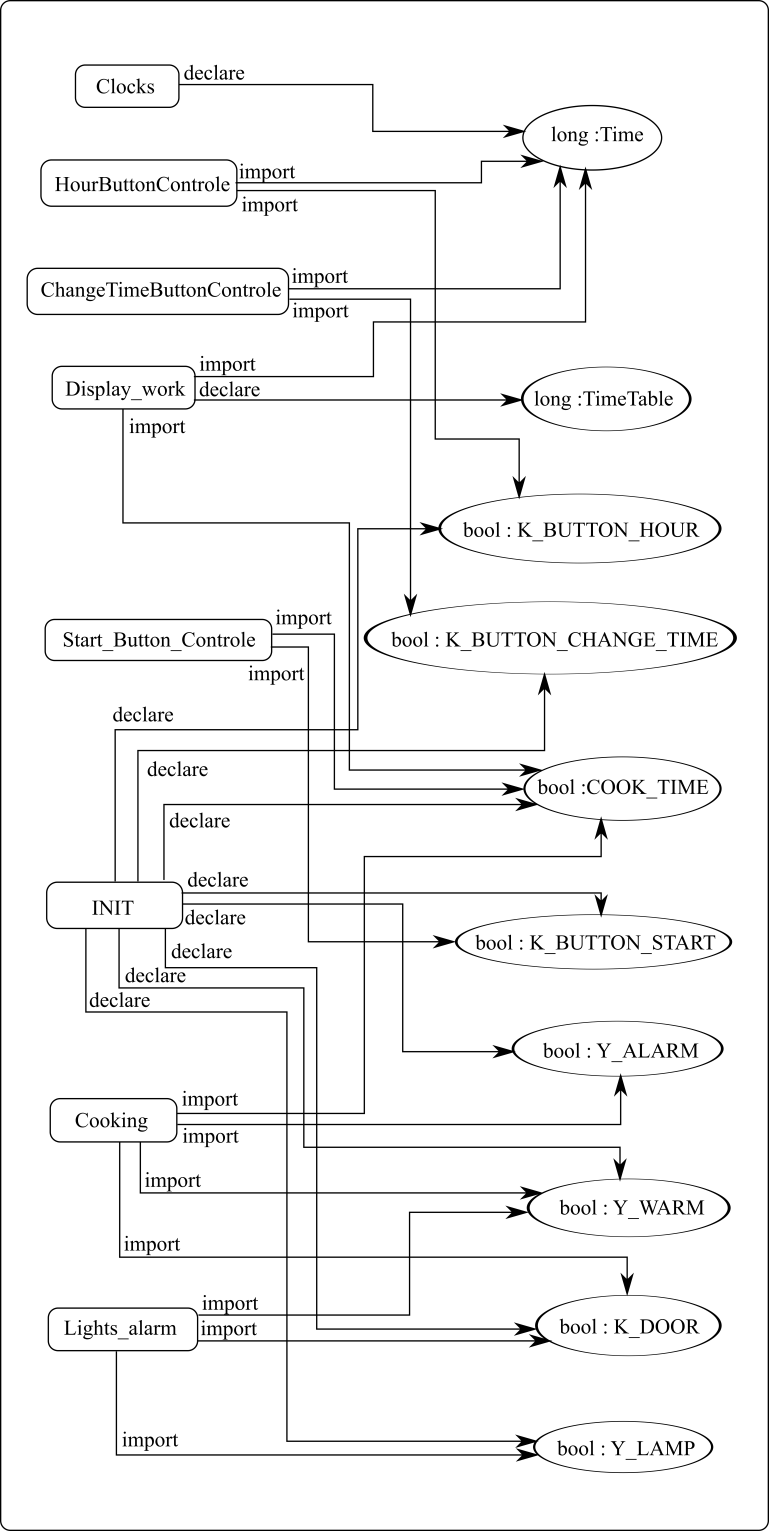
1. Example of a process state diagram

Names of process states are inside the nodes. They are connected by arrows, with conditional expressions (switch conditions) above them in brackets. Actions executed upon transition to that state are marked after the slash.

## Data communication diagram

A graphical notation has been developed for the data communication diagram. The notation is based on the statecharts diagram [6][7] (where we have borrowed a shape for process nodes) and on the activity UML diagram (where we have borrowed a shape for variable nodes and arrows labeling). An example of such diagram is shown on the Fig. 2.

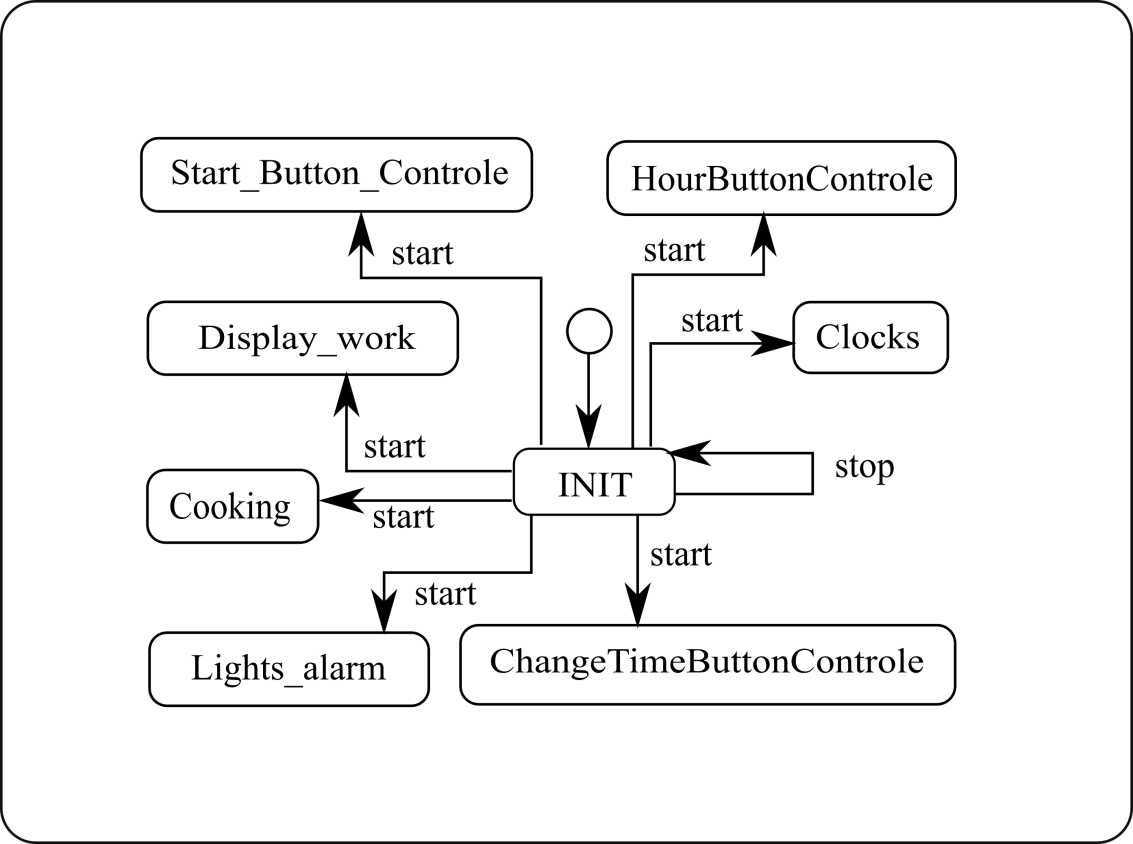
Processes names are marked in round-corner nodes, names of variables are written in elliptical diagram nodes. Process nodes are connected with variable nodes by arrows, and connection type is labeled above them (e.g. imported variables are marked as ‘import’, declared variables are marked as ‘declare’).   
  
  
  
  
  
  
  
  
  
  
  
(убрать пустоту)



1. Example of a data communication diagram

## Process control diagram

We have been (have analyzed?) analyzing graphic notations for diagrams to develop a process control diagram. Our diagram is based on the statecharts diagram [6][7], from where we have borrowed a start node labeling, and the activity UML diagram, from where we adopt the shape for nodes and marking. An example of such diagram is shown on the Fig. 3.



1. Example of a process control diagram

# Graph storage formats

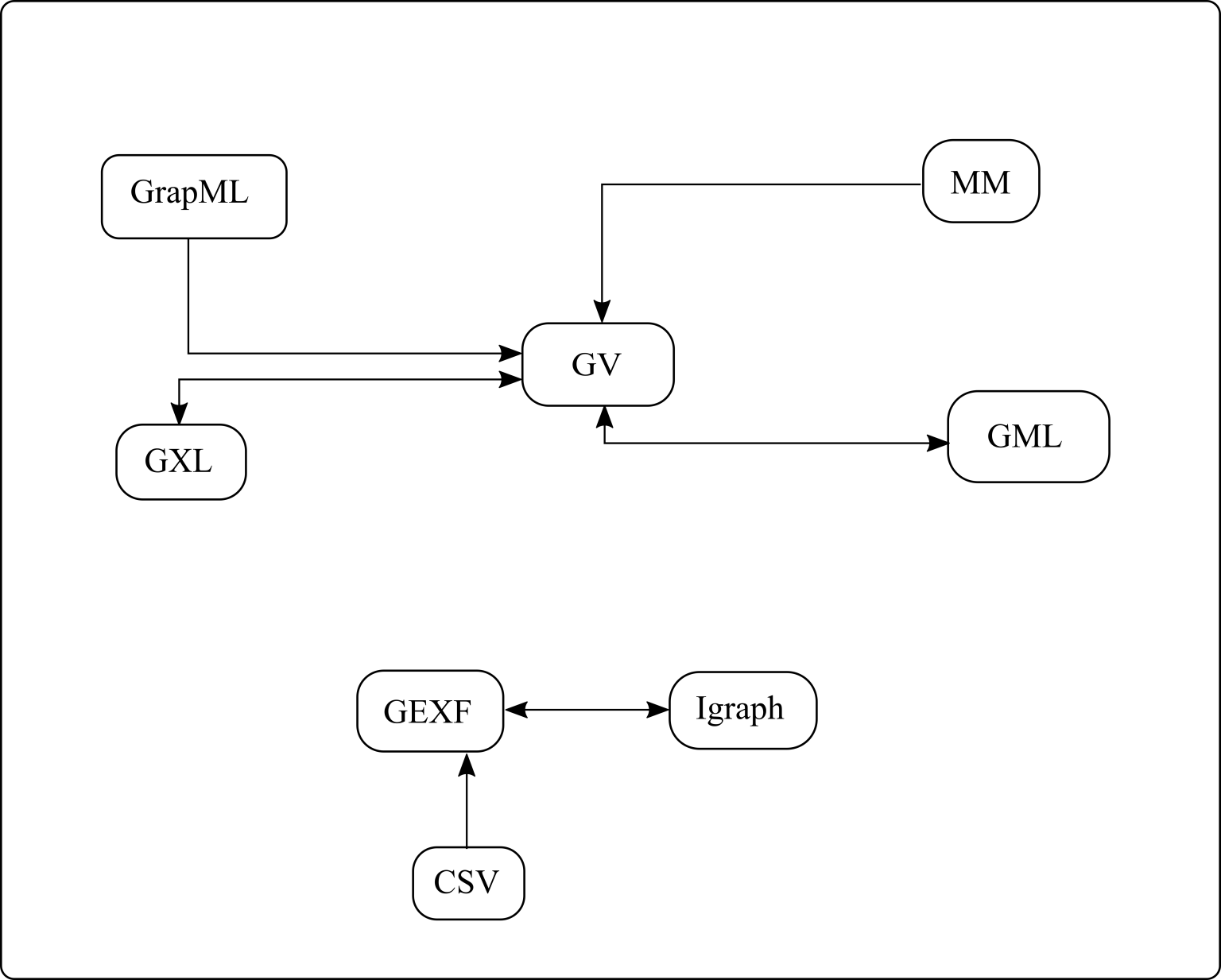
To choose the graph storage format for output diagram files for the software module, we have performed analysis of graph representation formats. The result of analysis is shown in the Table 2. The following formats were considered: GraphML [9][12], GV [16], XGML [13], GML [17], Node list [20], Edge list [20], PAJEC [14], LEDA [15,21,22], TLP [18][19], GW [15][21][22] and GEXF[23-25].

1. Graph representation formats

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Format name | Format’s abilities | | | | |
| Saving coordinates | Right node shape | Oriented graph | Edge labels | Tools that use format |
| GraphML | + | + | + | + | Gephi, Igraph, OGDF, Yed, NetworkX |
| GV | - | + | + | + | GraphViz, Gephi, Igraph, OGDF, ZGRViewer, NetworkX |
| XGML | + | + | + | + | Yed |
| GML | + | + | + | + | Gephi, Igraph, OGDF, Yed, NetworkX, Tulip, LEDA |
| Node list | - | - | + | - | Yed, NetDraw |
| Edge list | - | - | + | - | OGDF, Yed, NetworkX, NetDraw |
| PAJEC | + | + | + | + | Gephi, Igraph, NetworkX, NetDraw |
| LEDA | + | + | + | + | Igraph, OGDF, NetworkX, LEDA |
| TLP | + | + | + | + | OGDF, Gephi, Tulip |
| GW | + | + | + | + | LEDA |
| GEXF | + | + | + | - | Gephi, OGDF, NetworkX, Tulip |

An evaluation criterion like an ability to save coordinates is important because it makes possible to save graph layout (by a third-party layout tool) and diagram’s modifications after closing the file. We also analyzed notation criteria to understand which formats may storage graphs in our notations. The last criterion was a list of tools that are able to use the format [8][10][11].

An ability to convert one file format to another is showed on the Fig. 4. We found that here are two families of graph file formats, inside which conversion is possible.



1. An ability to convert one file formats

GraphML [9][12] and GML [17] file formats look most suitable for use in this work. We have made an analysis of these file formats, the result of which is shown on the Table 3.

1. Comparison of GraphML and GML file formats

| Comparison criterion | Name of the file format | |
| --- | --- | --- |
| GML | GraphML |
| File size | x | 2 \* x |
| Easy to read by human | + | - |
| Can store links | - | + |

GML has a human-readable syntax; GraphML has a XML-like syntax. GraphML diagrams have twice the size of the same GML diagrams. Also GraphML allows storing links, and that is important, because in our diagrams in the each process node we want to have a link to the state diagram for that process.

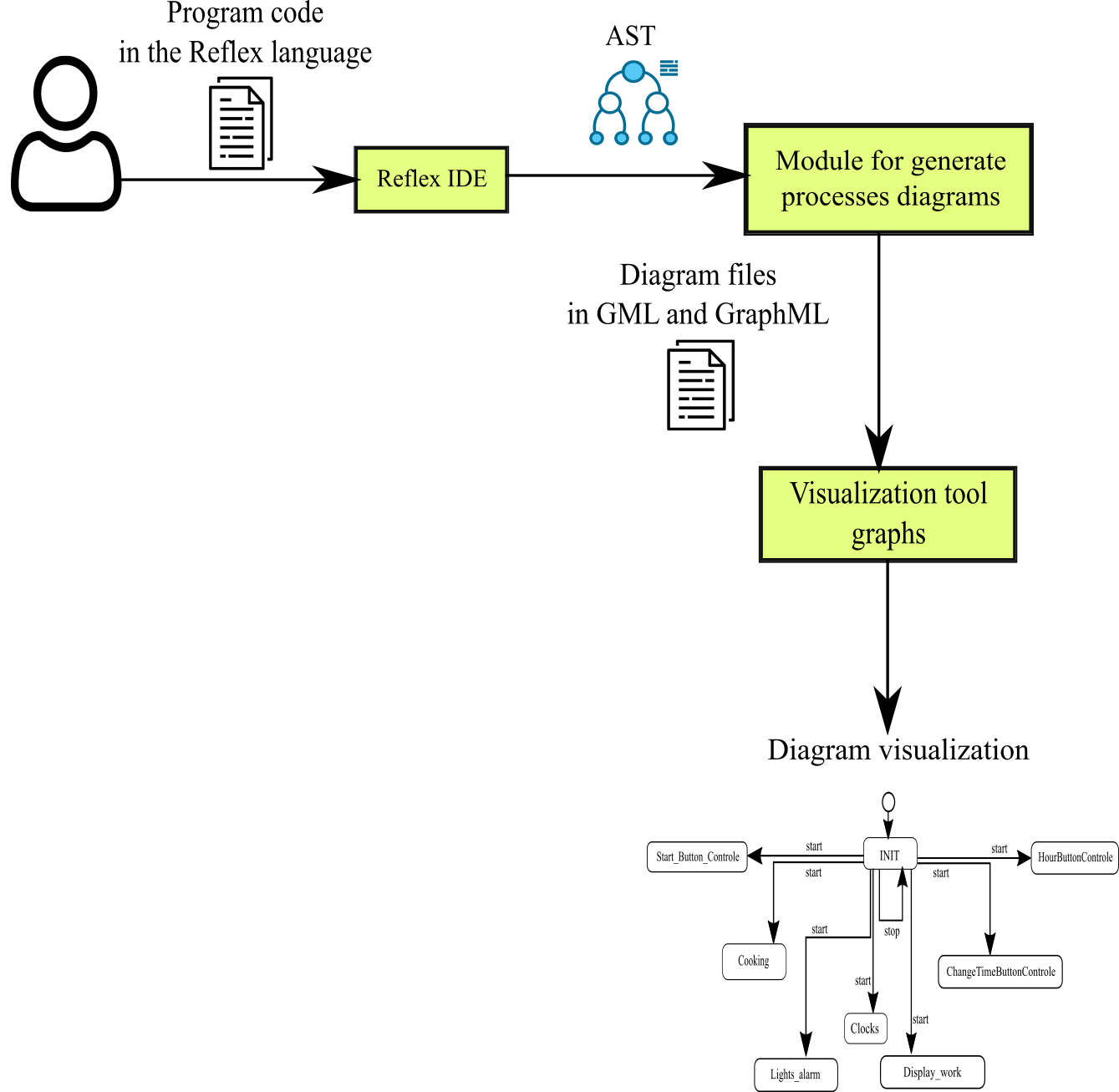
As a result, GML has been chosen for the process state diagram because of small sizes of files in that format (the module generates such diagram for the every process in the program). GraphML has been chosen for process communication and control diagrams, because it allows storing the links in graph nodes.

# Architecture of the software module

The developed module uses the Model–view–controller (MVC) software design pattern, which enables quick modification of the architecture. Firstly, the controller starts the model generator for process state diagrams, then, the view part creates GML files of those diagrams. After that, the controller starts the model generators for process communication and control diagrams. Then it uses a graph separator module to divide models of the diagram if / in case / should they have independent parts. After that the diagram generators create GraphML files of diagrams with links in the each process node to the state diagram for that process (which was generated before).

# Implementation of the sofware module

The module has been implemented as an Eclipse plugin (to simplify integration in the Reflex IDE) using Xtext and Xtend technologies. After opening the Reflex IDE and writing the code, the user can click the save file button, and then process data communication, control and states diagrams will be automatically generated. We recommend opening them with the yEd [10] tool that supports automatic graph layout. The sequence of user actions is shown on the Fig. 5.



1. Model usage scheme (избавиться от пустоты слева внизу)

# The conclusion

We have developed an approach to creating graphic automatic documentation tool for the domain-specific languages(точка?) via the using available external graph visualization tools for rendering diagrams which were created by the developed software module.

We have analyzed available tools for creating diagrams by the source code for common programming languages and graph storage formats for choosing the most suitable file format for that work. Also process-oriented programming language Reflex was analyzed from the side of the main Reflex features for visualization.

As a result, a software module for building process diagrams according to the specification in the Reflex language was created. The module creates processes process data communication, control and states diagrams. It automates the creation of the graphical documentation and simplifies the support of projects in the Reflex language. Practical testing has shown that using the module decreases the time for creating processes diagrams (from a few hours to a couple of seconds) and guarantees that there are no errors caused by human factors.

So, our approach was successfully tested on the Reflex language and may be used for other programming languages.

##### References

1. A. Rozov, V. Zyubin and D. Nefedov, "Hyperprocess-Based Approach for Embedded Microcontroller Programming", Vestnik NSU. Series: Information Technologies, vol. 15, no. 4, pp. 64-73, 2017. Available: 10.25205/1818-7900-2017-15-4-64-73.
2. A. Rozov, T. Lyakh, D. Krasnov and E. Sanzhiev, "A Practical Application of IndustrialC Language in Vacuum Deposition Plant Automation", Vestnik NSU. Series: Information Technologies, vol. 15, no. 3, pp. 90-99, 2017. Available: 10.25205/1818-7900-2017-15-3-90-99.
3. V. Zyubin, Процесс-ориентированное программирование: Учебное пособие. Novosibirsk: NSU, 2011. ?(https://www.system-informatics.ru/en/article/229)?
4. T. Lyakh, V. Zyubin and M. Sizov, "Опыт применения языка Reﬂex при автоматизации Большого солнечного вакуумного телескопа", Промышленные АСУ и контроллеры, no. 7, pp. 37-43, 2016. [Accessed 27 May 2020].
5. V. Zyubin, "Язык Рефлекс. Математическая модель алгоритмов управления", Датчики и системы, no. 5, pp. 24-30, 2006. [Accessed 27 May 2020].
6. D. Harel, "Statecharts: a visual formalism for complex systems", Science of Computer Programming, vol. 8, no. 3, pp. 231-274, 1987. Available: 10.1016/0167-6423(87)90035-9.
7. G. Lüttgen, "Modeling and verification using UML Statecharts. By Doron Drusinsky. Published by Newnes Publishers, 2006. ISBN: 0-7506-7617-5, 306 pages. Price £39.99. Hard Cover.", Software Testing, Verification and Reliability, vol. 18, no. 3, pp. 189-190, 2008. Available: 10.1002/stvr.377.
8. R. Tamassia, Handbook of graph drawing and visualization. CRC Press.
9. V. Kasyanov, Grapg representation language GraphML: basic tools 1. 2012.
10. t. yWorks, "yEd Graph Editor", yWorks, the diagramming experts, 2020. [Online]. Available: https://www.yworks.com/products/yed. [Accessed: 27- May- 2020].
11. "Визуализация больших графов для самых маленьких", Habr.com, 2020. [Online]. Available: https://habr.com/ru/company/ods/blog/464715/. [Accessed: 27- May- 2020].
12. "The GraphML File Format", Graphml.graphdrawing.org, 2020. [Online]. Available: http://graphml.graphdrawing.org. [Accessed: 27- May- 2020].
13. "XGML", Docs.yworks.com, 2020. [Online]. Available: https://docs.yworks.com/yfiles/doc/developers-guide/xgml.html. [Accessed: 27- May- 2020].
14. V. Batagelj and A. Mrvar, "Program Package: Pajek - Spider / Pajek to EPS", Vlado.fmf.uni-lj.si, 2020. [Online]. Available: http://vlado.fmf.uni-lj.si/pub/networks/pajek/doc/draweps.htm. [Accessed: 27- May- 2020].
15. "LEDA Guide: Native File Format for Graphs", Algorithmic-solutions.info, 2020. [Online]. Available: http://www.algorithmic-solutions.info/leda\_guide/graphs/leda\_native\_graph\_fileformat.html. [Accessed: 27- May- 2020].
16. "The DOT Language", Graphviz - Graph Visualization Software, 2020. [Online]. Available: https://graphviz.gitlab.io/\_pages/doc/info/lang.html. [Accessed: 27- May- 2020].
17. "GML", Docs.yworks.com, 2020. [Online]. Available: https://docs.yworks.com/yfiles/doc/developers-guide/gml.html. [Accessed: 27- May- 2020].
18. D. Auber and P. Mary, "Tulip software graph format (TLP)", Tulip.labri.fr, 2020. [Online]. Available: https://tulip.labri.fr/TulipDrupal/?q=tlp-file-format. [Accessed: 27- May- 2020].
19. D. Auber and P. Mary, "LGPL", Tulip.labri.fr, 2020. [Online]. Available: https://tulip.labri.fr/TulipDrupal/?q=licence. [Accessed: 27- May- 2020].
20. A. Prokhorov and N. Larichev, "Компьютерная визуализация социальных сетей", compress.ru, 2020. [Online]. Available: https://compress.ru/article.aspx?id=16593#NetDraw. [Accessed: 27- May- 2020]. (in Russian)
21. "AlgoSol - LEDA Graphs for Java", Algorithmic-solutions.com, 2020. [Online]. Available: https://algorithmic-solutions.com/index.php/products/leda-graphs-for-java. [Accessed: 27- May- 2020].
22. "AlgoSol - LEDA for C++", Algorithmic-solutions.com, 2020. [Online]. Available: https://algorithmic-solutions.com/index.php/products/leda-for-c. [Accessed: 27- May- 2020].
23. S. Heymann, "GEXF File Format", Gephi.org, 2020. [Online]. Available: https://gephi.org/gexf/format/schema.html. [Accessed: 27- May- 2020].
24. G. Woodhull, J. Ellson, E. Gansner, E. Koutsofios and S. North, "Graphviz and dynagraph – static and dynamic graph drawing tools", Citeseerx.ist.psu.edu, 2020. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.96.3776. [Accessed: 27- May- 2020].
25. S. Heymann et al., "GEXF 1.2 draft Primer", Gephi.org, 2020. [Online]. Available: https://gephi.org/gexf/1.2draft/gexf-12draft-primer.pdf. [Accessed: 27- May- 2020].