Constructing graph models for software system development and analysis

Andrey V Pogrebnoy

Tomsk Polytechnic University, 30, Lenina ave., Tomsk, 634050, Russia

E-mail: avpogrebnoy@gmail.com

Abstract. We propose a concept for creating the instrumentation for functional and structural decisions rationale during the software system (SS) development. We propose to develop SS simultaneously on two models – functional (FM) and structural (SM). FM is a source code of the SS. Adequate representation of the FM in the form of a graph model (GM) is made automatically and called SM. The problem of creating and visualizing GM is considered from the point of applying it as a uniform platform for the adequate representation of the SS source code. We propose three levels of GM detailing: GM1 – for visual analysis of the source code and for SS version control, GM2 – for resources optimization and analysis of connections between SS components, GM3 – for analysis of the SS functioning in dynamics. The paper includes examples of constructing all levels of GM.

1. Introduction

A wide range of software systems (SS), which interact with external environment and work in real time, is considered. External environment includes data sources and receivers. SS functioning depends on information from sources (client requests, operators, signals from sensors, detectors, video cameras, etc.), incoming in random or determined moments in time. System and network resources of the SS are spatially distributed and are limited.

A life cycle of the SS can be represented as a sequence: SS development; SS operation and collecting information for its improvement; developing of the new SS version while supporting a previous one. Version control systems (VCS), such as Git, SVN, are used for storing versions of the software [1-3]. VCS helps a developer to store and compare versions and review changes made to the SS source code.

Traditionally, an evolutional approach is used for SS development, which is based on iterative project development. In [4], the hybrid modeling concept is proposed to support an evolutional approach. According to the concept, SS development performed as an evolution of two models – functional (FM) and structural (SM). FM corresponds to the source code text form and SM represents a source code as a graph model. This allows executing FM at any step of the evolution and using SM for visual perception and determining system characteristics.

Thus, the graph model as a SM can be applied to achieve two main goals. First, it includes visual perception and analysis of the structural part of the source code semantics. According to results of the analysis, necessary changes are made to the FM. The second and more important goal is connected with the analysis of characteristics of the SS functioning by analytical methods. In the first place, it refers to analysis and optimization of the load characteristics of the computational and network

resources and to defining time estimations for complying real time conditions.

Hybrid modeling is an extension to the capabilities of the system structural modeling technology called SML [5]. The extension includes development of methods of creating SM as a graph model, visual analysis of the FM structure, determining characteristics of the SS work, evolution formalization of SS project models. As a result of solving that problems, a developer receives effective tools for supporting an evolutional approach of the SS development.

The paper focuses on building and visualization of graph models in the process of SS development and analysis basing on simultaneous application of FM and SM.

2. Problem definition

Graph models are widely used for software characteristics analysis and optimization. Examples are problems of code optimization in translators, effective memory usage, determining separate program characteristics, debugging and visualization tools.

For those cases, adequacy of the source code and its graph model is limited by the requirements of the particular problem. It is obvious that there also no possibility of the formal transition from the model to the program. Changes in the model cannot be automatically reflected to the program, i.e. models are not constructive. Each graph model is unique, i.e. there is no succession between models. It should also be pointed out that, for example, attempts of Petri nets application and their modifications [6] for SS dynamics analysis are very time consuming, wherein results are not constructive.

In the paper, the problem of the graph model creation considered from the point of its application as a uniform platform of the source code view resulting into tools for SS development support, version control, determining resources requirements, SS functioning in dynamics. Another requirement is that disadvantages of models, described earlier should not be applicable to the platform.

The building process of the graph model (GM) as a uniform platform has to be performed automatically by source code syntax analysis. Resulting GM corresponds to the first level model – GM1. For solving specific problems, other levels of GM can be built basing on GM1. The automatic creation of the GM1 and generation of the GMs for specific problems is one of the most important directions of the SML technology, where graph model was built manually.

3. Theory

The main idea of structural modeling is based on splitting the SS project into functional and structural parts. According to this, SS development is performed simultaneously on two models – functional (FM) and structural (SM).

Structural modeling can be described as some type of the hybrid modeling concept, which combines imitation and analytical methods. In the concept, SM is built at the basis of the current FM's variant and is used for analysis and optimization of the corresponding variant of the SS project. The following evolution of the FM is performed considering resulting characteristics and FM as an executable version of the SS. Results of work of the version must correspond to characteristics, acquired with help of SM.

Thus, SM during another iteration of the evolution process of the SS development acts as a tool for making decisions for extending and improving FM. Decisions, which answer the question "What SS must do?" is called functional. That includes decisions of development, extension and possible SS functional change and its software implementation.

All other decisions, which answer the question "How this functional must be executed?", are called structural. Those decisions are connected with making links between SS functions, organizing their cooperative work with limitations on network and computing resources. Making functional decisions cannot be formalized and performed by the project developer manually. In conditions of the project separation for functional and structural parts, the evolution process of the SS development is mainly connected with making structural decisions. Compared with the program implementation of the SS as FM, making structural decisions is more undetermined. That is why it is so important that in the structural modeling technology, structural decisions are made according to results of the SM analysis,

made with the help of analytical methods.

In the hybrid modeling concept, methods of the analytical analysis and SS project optimization is made at SM, which is built at the basis of FM automatically. Achieved results are used for making decisions of implementing changes and additions to FM, wherein SM is changed automatically. Imitation modeling of the SS is performed on FM and correspondence between results of project evolution, achieved for FM and SM, is checked.

For a project developer, SM is also acting as a visual guide for the SS development. Visual perception of the graph model gives the developer the possibility to easily find some incorrectness or errors in the source code of the analyzed part of the FM. In that context, FM can be considered as a reliable tool for checking whether another iteration of the SS evolution gave a positive result.

While developing FM, an application strategy of the SM is defined by many factors. Among them there are: SS complicity, computing resources distribution, real time conditions hardness, interaction with external environment conditions, project developer preferences. For simple SS, the application of the GM1 solely can be enough. For more complex and distributed systems, a developer might need analysis based on GM2. For complex and dynamic systems with a lot of interactions with external environment, the application of GM3 is recommended.

Project developer preferences can concern not only the selection of the GM level and the corresponding analysis tool, but also the possibilities of visual support of the FM development process in first place as a GM1. That is why, a scenario of the interactive access to the visualization tools while developing FM would be defined by the project developer.

4. Building a graph model

4.1. Building GM1

Building and visualization of the GM1 follow the process of the FM development and is performed sequentially for each completed syntactic construction (SC). SC consists of simple and complex operators, classes, class methods, class variables, class instances, which are basic elements of the source code and assumed as vertices in GM1.

Simple operators and higher level SC will be named modules for convenience. In GM, each module corresponds to one vertex. That is why GM at any level is represented as a module structure. At the GM1 level, modules corresponds to simple operators and control operators. And at the second level, each module corresponds to one SC or to the union of them. The third level is a SS dynamic model, where at the second level graph model, real time processes are marked out and their launch and interaction conditions are set.

Visualization and the building process of the GM1 are performed with the development of another SC in FM. For the SC source code, analysis is performed, GM1 is built and the result is attached to the main GM1, built before. While analyzing the SC source code, the list of variables and constants $D = \{d_i\}$, $i = 1, 2,\ldots, m$, m – amount of data d_i in list D is formed.

The list of simple operators and control operators $M = \{m_j\}$, $j = 1, 2,\ldots, n$, n – amount of modules m_j in list M is also formed. For each module m_j, the list of data d_i, which is processed by it, is defined. In GM this is reflected as a set of information connections $S = \{s_{ij}\}$. Connection s_{ij} corresponds to arc (d_i, m_j) if d_i is an input for m_j, and to arc (m_j, d_i) if d_i is an output of m_j.

A modules execution sequence in GM is reflected as a set of control connections $U = \{u_{j1, j2}\}$. Connection $u_{j1, j2}$ corresponds to arc (m_{j1}, m_{j2}), which means that module m_{j2} is executed after module m_{j1}. Set U can include control connections only for cases, when compliance of modules execution priority is a necessary condition of source code correct execution. In cases, when modules execution priority is uniquely determined by the data processing sequence or this priority can be arbitrary, corresponding connections in set U are omitted.

Therefore, GM1 described by four sets (D, M, S, U), which represents a source code structure and contains all information defining its execution. When necessary, the source code can be restored by this description. Adequacy of the software representation as a GM1 allows applying the graph theory

method for the software structural analysis in isomorphism and similarity categories

Figure 1 shows an example of the source code fragment in the Java language and GM1 created for it. In the left source code and corresponding GM1 for the *main* program, the method is showed. In the right source code and GM1 for *Hash* class, which is called from *main*, the method is showed. The program for each element of massive *args* is called method *calc* and sums up the result.

```
Hash hash = new Hash(10);

int sum = 0;

for (int i = 0; i < args.length; i++) {
  sum += hash.calc(args[i]);
}
```

```
class Hash {
  private int b;
  Hash(int b) {
    this.b = b;
  }
  int calc(int a) {
    return (a + b) * 3;
  }
}
```

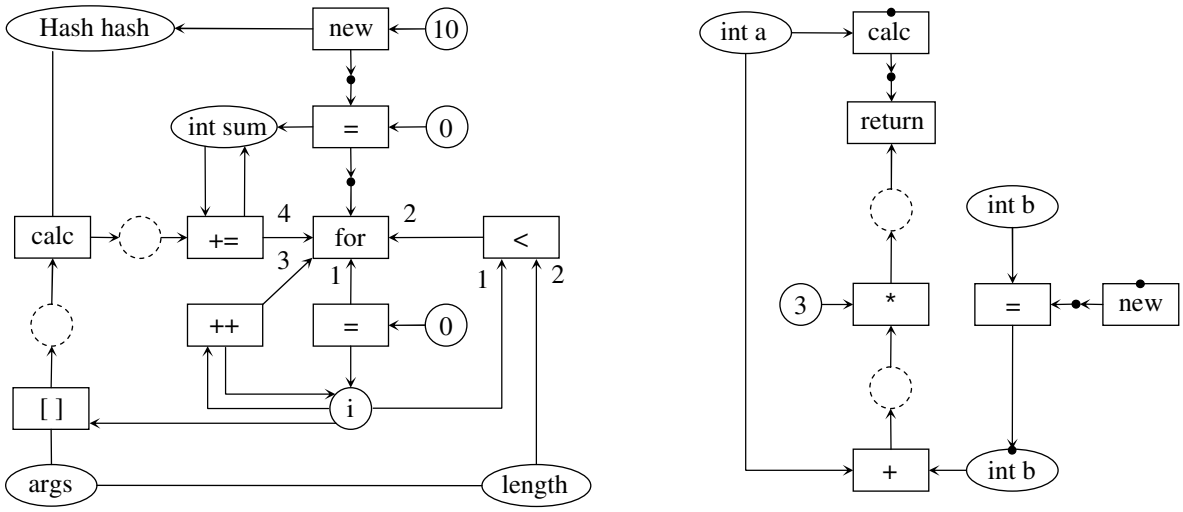

Figure 1. An example of the GM1 for the FM fragment.

Data in GM1 (closed curves) has three types of view. The solid line corresponds to the local variable. The solid line with a point reflects the class property. The temporary variable without a name is represented as a dotted line. Modules are showed as two kinds of rectangles. The rectangle without a point corresponds to the module, which executes an operator or method. The rectangle with a point corresponds to the class method declaration. Informational connections are showed with an arrow. In cases there is a need to distinguish informational inputs in the module, the number of the input is showed near the arrow. Connection without an orientation is applied for showing belonging of the module or property to the object.

Control connections (m_{j1}, m_{j2}) between modules are marked with points, which is used to simulate variables. Such variables correspond to appearance of the signal of module m_{j1} indicating that it has finished its job and module m_{j2} can be executed. Introduction of such variables allows keeping GM1 as a bipartite graph. Bipartite property of the GM1 simplifies solving some of analysis problems. A set of visualization elements shown in Figure 1 represents only those of Java's SC, which are used in the example. This set for the language also includes other elements, which are not presented in the example.

Besides natural goals of the visual analysis and comparison with corresponding FM fragments, GM1 is used for solving problems of the VCS. The first group of problems can be solved, because the graph form of the source code view does not depend on formatting, used by different developers of the

SS. This allows excluding the effect of the formatting settings, produced on the VCS. The second group of problems is connected with possibility to compare versions basing on the GM1. Graph model analysis as a part of the SS allows making more intelligent decisions than in traditional VCS, reducing the amount of conflicts which require developer's interference.

4.2. Building GM2.

GM1 to GM2 transformation is performed by uniting sets of GM1 modules, representing separate SC. Wherein each set is reduced to one module and is included in the GM2 module structure considering external connections. In the process of constructing the GM2 project, a developer can use an UML diagram [7], when necessary, for selecting sets of GM1 modules for uniting in GM2 modules. This allows one to sequentially bring the GM2 module structure to the state, more appropriate for the SS analysis.

Figure 2 shows an example, containing both visual views of GM2 and GM3 fragments. The example consists of 7 modules (rectangles), 16 variables (circles) and 2 external devices, represented as consumable or shared resources (cylinders). In GM2, modules of set $M = \{m_j\}$, $j = 1, 2,\ldots, 7$ are active objects, and a set of variables $D = \{d_i\}$, $i = 1, 2,\ldots, 16$ is representing inputs and outputs, which modules are interacting with. The interaction is defined by the set of connections $S = \{s_{ij}\}$. Thus, GM2 has a bipartite structure. Note that GM2 bipartity is also saved if in GM1 $U \neq \varnothing$. This achieved by introducing vertices, reflecting signals transition. In Figure 2, such vertex with number 9 corresponds to transition (m_4, m_5), which points out that module m_5 must be executed after m_4.

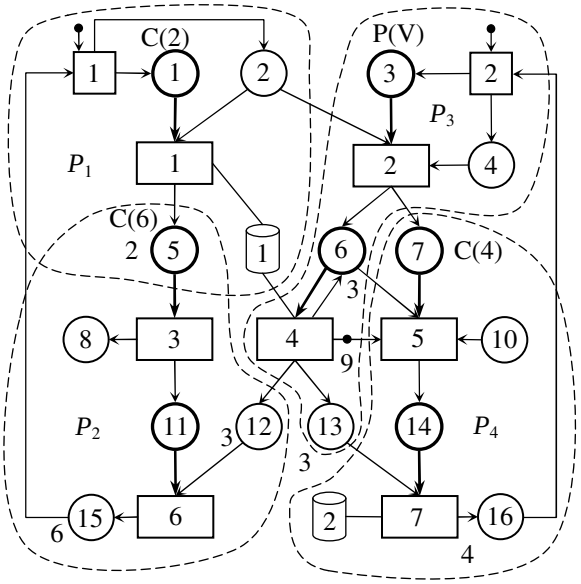

Figure 2. An example of GM2 and GM3 fragments visualization

In structural modeling SS, characteristics analysis is performed basing on SM, which, besides GM2, includes information about module parameters and their start conditions, variables income conditions and required memory. Rules of modules access to resources must be present. In Figure 2, these are modules m_1, m_4 and m_7. SM also must have information about available memory, computing and network resources parameters, external environment interaction conditions.

The analysis based on GM2 can be performed in three directions: intermodule connections analysis; resource and software load correlation; modeling SS work at the functional level. Let us consider possible transformation of the GM2 for performing analysis in those directions. The first direction includes building dependency graph [8], call graph [9] and control flow graph [10].

Problems of the section direction connected with approximate estimation of the resources, necessary for executing SS load with quality characteristics, are set. Main parameters estimating resources quality characteristics are processors performance, memory, the amount of network data transferred, the network bandwidth. The SS functioning quality is estimated mostly by time

characteristics. Methods of solving problems of that direction for distributed systems with more hard real time limitations are described in [11].

The third direction of the analysis performed at the basis of SS functioning is imitation under GM2 control. Such imitation in structure modeling technology is named modeling at functional level [11]. The main goal of the modeling is reduced to acquiring a full picture (diagram) of the resource consumed in the SS execution process. In the diagram, each resource is represented as time axis, where resources consumption of modules execution and data transfer in the network are showed.

4.3. Building GM3.

Transformation of GM2 to dynamic model GM3 is based on representing models as a set of dynamic objects, working in parallel and interacting between each other. Each module can be started: cyclically after a defined number of discrete time tacts; with probability rules of the event triggers, with the determinate sequence of events; when matching predefined conditions.

In each triggering module, the data state forms for its output parameters. According to this, the data states can enter at the module input cyclically (R-input), probabilistically (P-input), determinately (D-input), under condition (C-input). conditions R, P, D, C, listed below, are accompanied by the list of parameters and added to corresponding data inputs.

For simplifying SS functioning analysis problems in structural modeling, objects count reduction is performed by uniting modules, having worked dynamically corresponding to general rules, in one object. In structural modeling such objects are defined as a real time computing process (RT-process). In [5], the following definition is given for them: RT-process – is a dynamic object, consisting of one module or a set of interacting modules, having conformed with strategies of launch conditions, inputs income and outputs refresh conditions and performing functionally finished operation at acceptable for SS work time.

According to the definition for the GM3 example in Figure 2, four RT-processes are formed (dotted lines). Processes p_1, p_2, p_4 are launched cyclically with intervals of 2, 6, 4 tacts. Process p_3 is launched probabilistically according to the rule, defined in record V, with outputs d_{12}, d_{13}, d_6 having calculation time limitation of 3 tacts. This means, that total execution time of modules m_2 and m_4 cannot exceed 3 tacts since the moment of the process launched by d_3, defined by parameters, is record V. In Figure 2, external environment imitators are showed with squares. Inputs of modules, which are used for SS dynamic equations, are showed in bold.

Analysis, optimization and evolution of the SS dynamic model represented as GM3 is performed with the help of recurrence equations which define the model work dynamics in the matrix form [5]. The model state change trajectory is built by dynamics equations for different values of parameters (model variants). A lot of different trajectory quality estimations are proposed. Basing on those estimations, decisions for changing parameters values and building an acceptable model variant can be made.

5. Conclusion

The paper contains results of the initial step of the research of creating SS development and analysis of support tools, based on representing the program as a graph. For this, the SS source code, as a functional model (FM), was written, which was accompanied by building an adequate structural model (SM) and its visualization as a graph model (GM). During SS development, decisions are being made basing on the GM and FM visual perception and on results of SM analysis. For solving SM analysis problems, three levels of GM are proposed: GM1 as a bade platform of describing the structure of the source code, GM2 and GM3 for solving problems of analysis, optimization and evolution of the SS project.

Results of the research are basis for GM building and visualization methods algorithmization and programing. Methods of SS analysis, optimization and evolution developed with help of structure modeling technology must get significant improvement and adaptation for working with FM and SM. Problems connected with integration of proposed tools with existing programing systems and VCS are still need to be solved.

Researches of creating the platform at the basis of GM1, which will become common to all popular programing languages, looks promising. It should be noted that problems of GM1 automatic building and visualization as well as problems of reverse transfer from GM1 to the source code are not trivial

and require more deep integration with programing systems and VCS.

References

[1] Eric Sink 2011 *Version Control by Example*, ed Brody Finney (Champaign: Pyrenean Gold Press) p 288
[2] Jon Loeliger, Matthew McCullough 2012 *Version Control with Git, 2nd Edition*, ed Andy Oram (Sebastopol: O'Reilly Media) p 456
[3] Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato 2008 *Version Control with Subversion, 2nd Edition* (Sebastopol: O'Reilly Media) p 432
[4] Pogrebnoy AV 2015 *Proc. of the XIII Int. Conf. "Molodezh i sovremennie informatsionnie tekhnologii"* **1** 16-17
[5] Pogrebnoy A V, Pogrebnoy D V, Pogrebnoy V K 2012 *Uravneniya dinamiki funktsionirovaniya SRV, predstavlennoy na yazike SML* (Saarbrücken: LAP LAMBERT Academic Publishing) p 120
[6] Tadao Murata 1989 *Proc. of the IEEE* **77(4)** 541-580
[7] Grady Booch, James Rumbaugh, Ivar Jacobson 2005 *The Unified Modeling Language User Guide (2nd Edition)* (Boston: Addison-Wesley Professional) p 496
[8] Jeanee Ferrante, Karl J. Ottenstein, Joe D. Warren 1987 The program dependence graph and its use in optimization *ACM Trans. on Program. Lang. Syst.* **9(3)** 319-349
[9] David Grove, Craig Chambers 2001 *ACM Trans. on Program. Lang. Syst.* **23(6)** 685-746
[10] Richárd Dévai, Judit Jász, Csaba Nagy, and Rudolf Ferenc 2014 *Acta Cybernetica* **21** 419-437
[11] Pogrebnoy V K 2011 *Avtomatizirovannoe proektirovanie raspredelennikh sistem realnogo vremeni* (Tomsk: Izdatelstvo TPU) p 312