INFORMATION TECHNOLOGY OF SOFTWARE ARCHITECTURE STRUCTURAL SYNTHESIS OF INFORMATION SYSTEM

Sergey Chainikov
Department of System Engineering
Kharkiv National University of Radioelectronics
14 Nauki ave., Kharkiv, Ukraine, 61166
chajnikov@kture.kharkov.ua

Andrey Solodovnikov
Department of Medical and Biological Physics and Medical Information Science
Kharkiv National Medical University
4 Nauki ave., Kharkiv, Ukraine, 61166
andreysldvnk@yandex.ru

Abstract

Information technology of information system software architecture structural synthesis is proposed. It is used for evolutionary models of the software lifecycle, which provides configuration and formation of software to control the realization and recovery of computing processes in parallel and distributed computing resources structures. The technology is applied in the framework of the software requirements analysis, design of architecture, design and integration of software.

Method of combining vertices for multilevel graph model of software architecture and automata-based method of checking performance limitations to software are based on the advanced graph model of software architecture. These methods are proposed in the framework of information technology and allow forming a rational structure of the program, as well as checking for compliance with the functional and non-functional requirements of the end user.

The essence of proposed information technology is in displaying of the customer's requirements in the current version of the graph model of program complex structure and providing a reconfiguration of the system modules. This process is based on the analysis and processing of the graph model, software module specifications, formation of software structure in accordance with the graph model, software verification and its compilation.

Keywords: software, information system, configuring, evolutionary requirements, graph model.

DOI: 10.21303/2461-4262.2016.00125

1. Introduction

Recently, such software design technology of information system (IS) like SSADM, Meris, TDD, Gherkin and etc. [1–3] have become widespread. Existing technologies allow creating component-oriented, service-oriented methods, generation method used also for the design and configuration of software IS. Application of these methods is due to their efficiency, low labor and time expenditures to produce quality software.

However, existing methods for software building show the effectiveness in the case of the requirements set before the start of software architecture design. In the conditions of dynamically changing requirements, using the evolutionary software life cycle (LC) requires a significant amount of code, material and labor costs to reconfigure users' workstations, which deprives these methods are significant advantages [1, 3]. It also is not considered application of techniques for simplify software structure based on a graph model analysis of its architecture in order to reduce the time for design and configuration [4].

One of the new directions in the design and development are methods of automated software synthesis with accounting of changing customer requirements, the main advantage of which is the possibility of a preliminary assessment of the architectural and functional features of the system [4–6].

In this context, it is appropriate to use the methods of formalizing flexible software architecture with the ability to dynamically change based on the multilevel graph model of software architecture to address the issues arising in the conditions of evolutionary requirements.

Such methods allow forming a preliminary view of the system prior to its generation, providing dynamic configuration and the ability to control the implementation and back of computational processes (CP) for the tasks of system users.

Known methods for the software IS synthesis are used primarily for the formation of a system with a component-oriented or service-oriented architecture, but the principles and models that are the basis of the method of formalizing software architecture can be adapted to meet the challenges that arise in the process of automated software synthesis.

It is therefore highly *relevant* to use information technology (IT) of software architecture structural synthesis using the corresponding graph models.

The aim of the research is to develop models, methods and IT to improve the efficiency of IS software architecture design for structured tasks. At the same time such objectives of the study are formulated:

– development of multilevel graph model of IS software architecture in the multilevel form;

– development of method for combining vertices of software architecture graph model based on an assessment of indicators of complexity and connectivity of software modules;

– development of IT of software architecture structural synthesis with the ability to software configuration under evolutionary requirements.

2. Materials and methods

2. 1. Graph multilevel model of IS software architecture

Graph model is a set of vertices, which compares the basic software modules executed sequentially or independently of each other. Focused arcs between them determine the type of communications of data and are described by a couple of sets of software modules characteristics. Graph model is formed with functionality that excess for a given subject area (SA) and provides a flexible configuration of the system in terms of evolutionary requirements.

The data about requirements specification to the final software product [7], information obtained at the stages of pre-research, existing architectural solution and patterns are required to build the model. The final form of the graph model is defined by expression in the form:

$$G=F\,(G_{in}), \tag{1}$$

where $G=F(G_{in})$ – defines a set of operations on the initial graph model, which allows to reduce it to the multilevel form with supervertices, and G_{in} is an initial graph model in the form:

$$G_{in}=(V_{in}, X_{in}), \tag{2}$$

where the V_{in} – set of vertices v, which matched software modules (or, in a general sense, CP) and X_{in} – set of directed arcs $x_{ij} = (v_i, v_j)$ of the G_{in} graph determining data dependencies between software modules. Formation a plurality of vertices and edges of the graph (2) takes place on the basis of selection of prelevant functional subsystems of investigated object in SA and the establishment of data flow between them. For each i-th vertex of the model (1) two vector data are highlighted:

$$D_i^{in} = \left\{d_1^{in}, d_2^{in}, \ldots d_k^{in}\right\}, \tag{3}$$

vector of input data invariable during operation and

$$D_i^{out} = \left\{d_1^{out}, d_2^{out}, \ldots d_k^{out}\right\}, \tag{4}$$

vector of output data that had been modified in the process of the program module operation.

In the case of evolutionary requirements, defining a modification of the architecture and functionality expansion, graph model (1) is supplemented by a subset of new software modules

$V^M = \{v_i^m\}$ and a subset of modified modules $V^M = \{v_i^m\}$ that allow configuring the software structure for the new version of the requirements. For the new version of program structure a subset of vertices $V^E \subseteq V$ are excluded from the graph (1), which are replaced by a subset V^M, that is, removed an older version of the software modules. The same operation is done with the arcs $X^M \subseteq X$. This allows defining the constant portion of the graph G:

$$G^C = \left(V^C, X^C\right), G^C \subseteq G,$$

where $V^C = V \setminus V^M$ and $X^C = X \setminus X^M$. The resulting graph model has the following main features. It hasn't pendant vertices, for which there are following condition:

$$\forall v \in V, \{V' \subseteq V : |V'| > 1 : \exists v_i \in V' : \deg^-(v_i) = 0\}. \quad (5)$$

There are no isolated vertices, for which there are following condition:

$$\forall v \in V, \{\exists v_i \in V \,|\, \deg^+(v_i) = \deg^-(v_i) = 0\}. \quad (6)$$

There are no paired arcs

$$\forall i, j \in N, \{v_i, v_j \in V \,|\, \exists! x_{ij} \in X\}. \quad (7)$$

And there are no vertices with loops

$$\forall i \in N, \{v_i \in V \,|\, x_{ii} \notin X\}. \quad (8)$$

In addition, all vertices for the acyclic graph model, for which is set the number based on the topological sorting and satisfying condition

$$\forall i, j \in N, i < j, \{\exists v_i, v_j \in V : \mu[v_i, v_j] \,|\, \deg^-(v_i) = \deg^+(v_i) =$$
$$= \deg^-(v_{i+1}) = \deg^+(v_{i+1}) = \ldots = \deg^-(v_j) = \deg^+(v_j) = 1\}, \quad (9)$$

where $\mu[v_i, v_j]$ – the route from the vertex v_i to the vertex v_j, combined into one supervertex. The purpose of the proposed graph model is description of software architecture. This description allows evaluating software to the direct aggregation and provide dynamic configuration of user workstations, which is an advantage of the proposed graph model. Reducing graph model to the form (2) to ensure the feasibility of the properties (5)–(9) is based on the proposed method of combining graph vertices of graph multilevel model of IS software architecture on the basis of estimation of complexity and connectivity of software modules [8].

2. 2. Method of combining the vertices of graph multilevel model of IS software architecture

This method is designed to reduce the time required for system development by integrating the vertices in the supervertices on the basis of the modified Kosaraju's algorithm that further processing graph model based on a comprehensive criterion of effectiveness evaluation for software architecture [8]:

$$K_c = \frac{K_{gen}}{K_{gen}^{'}} \cdot \frac{K_{compl}^{'}}{K_{compl}}, \quad (10)$$

where $K_{gen}^{'}$ and $K_{compl}^{'}$ – the average value of complexity index and the generalized criterion obtained after optimization of the graph structure and K_{compl} and K_{gen} – the value of criteria to optimization.

The method of combining the vertices of graph model of IS software architecture includes the following stages:

Stage 1: removal of topological inaccuracies for graph model in accordance with the formulas (5), (6).

Stage 2: obtaining number of modules for each i-th software module, the number of elements and data structures updated by i-th module and system complexity index values.

Stage 3: calculation of the average complexity index (10).

Stage 4: calculation of the criteria K_{compl} and K_{gen}, and get the value of complex criterion of evaluation of the effectiveness of software architecture (10) for a given version of the requirements for system configuration.

Stage 5: search for strongly connected components of a given graph and condensation of component.

Stage 6: search for a pair of vertices on the condition (7) after the graph hasn't any strong connection component (for any two vertices v_i and v_j no two coexisting oriented ways $\mu[v_i, v_j]$ and $\mu[v_j, v_i]$).

Stage 7: calculation of the value of the complex criteria for evaluating the effectiveness of software architecture (14) for obtained pair of vertices.

Stage 8: combining two vertices in one supervertex if the criterion value is strictly less than one, and repeat from stage 3 as long as the value of complex criterion (10) would not meet the configuration requirements.

Complexity assessment is conducted for the proposed method. It is coincided with the Kosaraju's algorithm and corresponds to the magnitude $O(n)$. The advantage of the proposed method is that the method can reduce the time for configuration by simplifying the structure of the graph model, relying on its topological features and configuration requirements for the system [8].

Graph model of IS software, processed by the method of combining vertices allows describing the static aspect of the generated software. To assess the behavioral properties of the system, to verify the non-functional requirements it is necessary to build an automata-based model of the interaction of the software modules based on automata-based method.

2. 3. Automata-based methods of constraint checking to generated software

Automata-based methods use to design the CP realizer, simulating scenarios of interaction of software modules and comparing the simulation results with non-functional requirements for PC. The method uses a multilevel graph model of IS software architecture. The vertices for this model are defined. These vertices are responsible for formation of the user dialogue and formation of corresponding subgraphs of modules implementing related problems of the system. The method is based on a finite automata-based model (FA), which is set by the standard set of elements [9]:

$$A = \{S, X, Y, s_0, \delta, \lambda\},$$

where S – the finite non-empty set of states; X – a finite non-empty set of input signals (input alphabet); Y – a non-empty finite set of output signals (output alphabet); $s_0 \in S$ – an initial state; $\delta : S \times X \to S$ – transition function; $\lambda : S \times X \to Y$ – output function. At the same set of states, transitions, input and output signals are defined graphically by transition diagram, and the structure of finite automata, relationship between nested automata using class diagrams. For the implementation of direct tasks (AForwardSM automaton) and reverse (recovery of results – AbackwardSM automaton) control of CP it is proposed to use the main control FA, which provides interaction of the two nested FA. The main tasks that implemented by the proposed FA are formulated on the basis of specific structure.

For control FA there are:

1) forming a subgraph of tasks;
2) checking the status of the problem;
3) checking the results for correctness;
4) change in the task status.

For AForwardSM automaton there are:

1) CP running;

2) CP data archiving.

For ABackwardSM automaton it is archive data recovery for each vertex of corresponding levels.

The proposed method includes the following stages:

Stage 1: selection of basic software states and conditions of transition on the basis of input and output vertices of graph model.

Stage 2: formation of control FA structure and its nested sub-automata realizing the behavior of the system, taking into account the specifications of the data of graph model vertices.

Stage 3: defining the principles of FA interaction with software modules based on architectural patterns of object-oriented programming for organizing distributed or parallel execution of the program.

Stage 4: formation of initial information required for FA organization, such as an adjacency matrix of graph multilevel model of IS software architecture and the transposed adjacency matrix to control forward and reverse FA process.

Stage 5: development of architecture for the future IS software, taking into account the relationship between graph model and control FA.

Stage 6: software generation on the basis of obtained structure.

The proposed automata-based models have the following required properties.

For AForwardSM automaton there are:

1) CP $P(v_i)$ for a given i-th vertex will be running only at the user's request;

2) guarantee that the $P(v_i)$ process is necessarily ever being running if there are no calculation errors;

3) guarantee that the $P(v_i)$ process will be running only when the v_i vertex is in a "not running" or "made";

4) the vertex will ever need to be in a state of "not running" or "made";

5) data archiving process will be carried out for each required vertex of the graph model;

6) for CP sequences based on the existing multilevel graph model it can distinguish a CP that is activated only if all the previous processes have completed their work;

7) for the entire set of software CP can identify a CP subset, which are placed on a single level of multilevel graph model and do not have the information dependencies of each other, allowing running them simultaneously.

For ABackwardSM automaton there are:

1) CP data recovery (backtracking) will be executed only at the user's request;

2) CP data recovery will be executed only if there is an archive data for each vertex of CP subgraph;

3) for each vertex of the selected subgraph will data recovery only in the state, if another user has not been enabled the top for himself ("not running" status) and if there are no data recovery ("not recovered" status), or the data has already been recovered earlier ("recovered" status).

On the basis of defined and designated FA states (**Table 1**) and the events for which FA transit in these conditions (**Table 2**) properties of the model based on temporal logic LTL, which meet the requirements of software behavior are formalized and complemented by new features in the case of regular versions of the non-functional requirements.

Table 1

FA transitions

Designation	Description
e1	permission is received
e5	run-time error
e6	number of vertices less than the maximum for the current level
e7	number of the current level less than amount of levels
e16	CP conditions and parameters are set
e43	Data are read
e81/e82	Archive exists/no exists
e101/e102	archive data is not restored/restoring
e103	archive data are restored
e121/e122/e123	CP is not running/running/executed

Table 2

FA states

Designation	Description
o3.zf22	CP initialization for AForwardSM automaton
o3.zf24	transition to the next CP
o3.zf28/o3.zf31	CP running/waiting for CP completion
o3.zf35/o3.zf36	data archiving/data retrieving
o2.zf45	Data recovery for ABackwardSM automaton

For AForwardSM automaton there are:

1) $\neg(\neg e1 U o3.zf28)$;

2) $\neg(e5 U o3.zf28)$;

3) $\neg(\neg(e121 \vee e123) U o3.zf28) = \neg((\neg e121 \wedge \neg e123) U o3.zf28)$;

4) $Fo3.zf31$;

5) $e43 Fo3.zf35$;

6) $F_{ov}^{AV} = \neg(F_1^{AV} \wedge F_2^{AV} \wedge ... \wedge F_n^{AV}) = \neg F_1^{AV} \vee \neg F_2^{AV} \vee ... \vee \neg F_n^{AV}$,

where $F^{AV} = \neg(\neg e122 U o3.zf31) \wedge \neg(e121 U o3.zf22) U (\neg(\neg e16 U o3.zf28))$;

7) $F_{ov}^{PV} = \neg(F_1^{PV} \wedge F_2^{PV} \wedge ... \wedge F_n^{PV}) = \neg F_1^{PV} \vee \neg F_2^{PV} \vee ... \vee \neg F_n^{PV}$,

where $F^{PV} = \neg(\neg(\neg(\neg(e6 \wedge e7) U o3.zf28)) U (\neg(\neg e122 U (o3.zf24 \wedge o3.zf36)))) =$

$= \neg((\neg e6 \vee \neg e7) U o3.zf28 U (\neg(\neg e122 U (o3.zf24 \wedge o3.zf36))))$.

Formalization of property 6 is due to the fact that in the case where there are n vertices, such that $\forall k = \overline{m,n}; \exists x_{ki} = (v_k, v_i)$, where m and n are integers and indegree $\deg^+(v_i) = n$, general logical statement is that both of the properties for all n arcs. Property 7 is formalized in such a way, as if to give a subset of vertices $v_i \in V_r$, where $i = \overline{m,n}$ belonging to the same level and between which there is no focused arc, in general, the logical formula will also appear as a conjunction of formulas correlated with the given vertices.

For ABackwardSM automaton the properties are formalized as follows:

1) $\neg(\neg e1 U o2.zf45)$;

2) $\neg(\neg e82 U o2.zf45) \vee \neg(\neg e81 U o2.f45)$;

3) $\neg(\neg(e121 \wedge (e101 \vee e103) U o2.zf45) \vee \neg((e122 \wedge e102) U o2.f45) =$

$= \neg((\neg e121 \vee \neg e101 \wedge \neg e103) U o2.zf45) \vee \neg((e122 \wedge e102) U o2.f45)$.

Formulated properties are the basis for the determination of CP execution, so automaton model, developed in accordance with the proposed method allows forming restrictions to software structure in response to evolutionary requirements and improving its operational reliability, based on the data of graph model of IS software architecture that distinguishes this method from BDD technology.

2. 4. IT of IS software architecture structural synthesis

IT of IS software architecture structural synthesis in conditions of evolutionary requirements allows to reflect the requirements of the current version at the graph model and provide a reconfiguration of software modules. The technology includes the following stages (**Fig. 1**):

Stage 1: formation of the initial graph model of IS software architecture;

Stage 2: analysis and procession of the graph model;

Stage 3: condensation and obtaining multilevel model;

Stage 4: preparing of the metafile of program modules specifications;

Stage 5: preparing of the project specification metafile;

Stage 6: composition of software architecture in accordance with graph model;

Stage 7: verification of software architecture;

Stage 8: software compile phase.

The first stage of IT is provided by information from available documents describing the requirements of the customer: primary specifications, diagrams of static and dynamic SA modeling. On the basis of the functional and architectural requirements for IS software for essence of SA model the functions of the objects that form the graph model, which is checked for topological cor-

rectness in the second stage, are defined. In the third stage (**Fig. 1**) based on customer requirements the method of combining vertices of graph model is carried out and rational graph structure of the graph is formed. In the fourth stage on the basis of the obtained structure the program modules are descripted using specification metafiles.

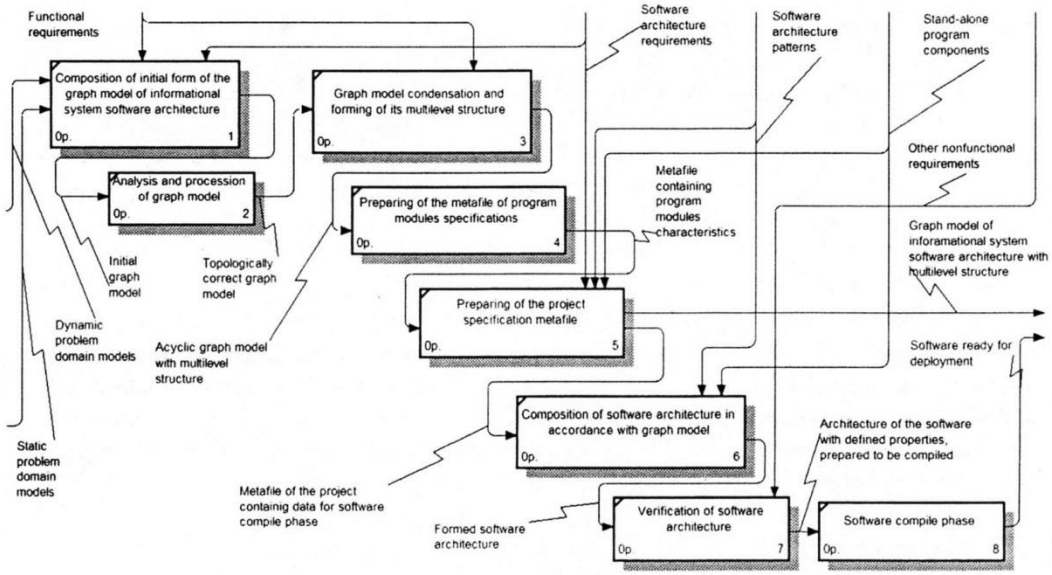

Fig. 1. IT stages of IS software architecture structural synthesis

In the fifth stage, using architectural patterns, metafile of software project that necessary for software generation and dynamic configuration, which describes the principles of interaction between modules, is generated. This file is input to the process of verification of non-functional requirements on the basis of assessment to verify compliance with the restrictions to the generated IS software in the seventh stage, after successful completion of which software is generated and the alienation of the graph model for the end user of the generated system is executed. If there is a new version of the requirements, the process is repeated from the third stage.

3. Experiments

IT enables the synthesis of the graph model of IS software architecture and dynamically configures its architecture and functionality of the user workstations in a customer's evolutionary requirements. Checking the applicability of the proposed IT to synthesize of software architecture "Automated system for pre-production of complex electronic devices" using the information about SA, the percentage of manual system configuration required for dynamic configuration was calculated (**Fig. 2**).

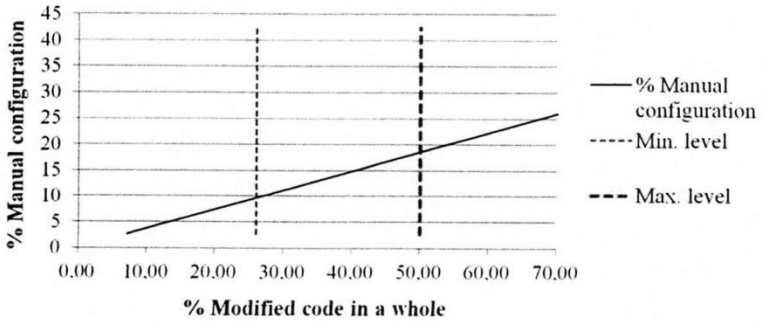

Fig. 2. Percentage of manual system configuration on the change percentage of the source code

The indicator calculation is based on LOC estimation and compared to the standard method of system configuration based on the report about implementation of ERP-systems in the production for 2015 [10].

4. Results of research

According to a report about implementation of ERP-systems [10], the majority of enterprises (on average 44 % of enterprises in all industries, and 38 % of the enterprises of mechanical engineering) require 26 % to 50 % change in the source code according to the settings and requirements of system customers (**Fig. 2**). The proposed technology reduced the percentage of manual work of staff to 10 % for the modification of source code in a conventional manner at a rate of 26 % and to 18 % for the corresponding 50 % limits. That is, the proposed IT significantly reduces the time required for reconfiguration of IS software by 36 %.

5. Discussion of results

IT of IS software architecture structural synthesis in terms of evolutionary requirements, which, unlike existing technologies, is complemented by analysis and processing stages of SA model to synthesize the graph model on IS software architecture, configuration of the structure to reflect the current functional requirements and verification of non-functional requirements based on estimation of sequences of module interaction within the requirements analysis process, architecture design, design and aggregation of software, which provides a significant reduction in the cost of software changes.

6. Conclusions

The proposed technology allows to significantly reducing time for software configuring by 36 % and is used as an addition to existing technologies of design and development of software, meaning its use for the evolutionary life cycle of technologies such as BDD, TDD, etc. within the requirements analysis process to the software, architecture design, design and aggregation of software.

Determination of principles for control, rollback and synchronization of CP in parallel computing environments, as well as methods of forming a flexible software architecture with its adaptation for service-oriented and cloud architectures is the subject of further research.

References

[1] Amodeo, E. (2015). Learning Behavior-driven Development with javascript + Code. Birmingham: Packt Publishing, 392.

[2] Bek, K. (2003). Ekstremalnoe prohrammyrovanye: razrabotka cherez testyrovanye. Byblyoteka prohrammysta. Sankt Petersburg: Piter, 224.

[3] Smart, J. F. (2015). BDD in Action: Behavior-driven development for the whole software lifecycle. Manning Publications, Shelter Island, NY, 384.

[4] Hilliard, R., Malavolta I., Muccini, H., Pelliccione, P. (2012). On the Composition and Reuse of Viewpoints across Architecture Frameworks. Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), Helsinki, IEEE, 131–140. doi: 10.1109/wicsa-ecsa.212.21

[5] Lavryshcheva, E. M. (2008). Sborochnoe prohrammyrovanye. Osnovy yndustryy prohrammnykh produktov. Kyiv: Naukova Dumka, 372.

[6] Lavryshcheva, E. M. (2013). Software Engineering kompiuternykh system. Paradyhmy, tekhnolohyy i CASE-sredstva prohrammyrovanyia. Kyiv: Naukova Dumka, 283.

[7] Coelho, K., Batista, T. (2011). From Requirements to Architecture for Software Product Lines. 9th Working IEEE/IFIP Conference on Software Architecture (WICSA), Boulder, CO: IEEE, 282–289. doi: 10.1109/wicsa.2011.60

[8] Solodovnykov, A. S. (2014). K voprosu otsenyvanyia effektyvnosty i slozhnosty struktury prohrammnoho sredstva. Problemy informatsiinykh tekhnolohiy, 2 (16), 125–129.

[9] Shelehov, V. I. (2014). Yazyk i tekhnolohyia avtomatnoho prohrammyrovanyia. Prohrammnaia inzheneryia, 4, 3–15.

[10] The 2015 Manufacturing ERP Report. (2015). Panorama Consulting Solutions, Denver, Colorado, 13.