VYTAUTAS MAGNUS UNIVERSITY
INSTITUTE OF MATHEMATICS AND INFORMATICS

Vaidas Giedrimas

THE COMPONENT-BASED SOFTWARE GENERATION METHOD

Summary of Doctoral Dissertation
Physical Sciences (P 000)
Informatics (09 P)
Informatics, System Theory (P 175)

Vilnius, 2010

This research was accomplished in the period from 2002 to 2006 at Institute of Mathematics and Informatics.
The right for the doctoral studies in informatics was granted to the Institute of Mathematics and Informatics together with Vytautas Magnus University by Government of the Republic of Lithuania, Decree No. 1285, issued on the 13th of December 2004.
Dissertation defended by an external student.

The dissertation will be defended at the public session of the Scientific Council in the field of Informatics in the Seminars Room of the Institute of Mathematics and Informatics at 11:30 a.m. on May 5, 2010.
Address: Akademijos str. 4–203, Vilnius, LT-08663 Lithuania

The summary of Dissertation is sent on April 2, 2010.
The dissertation is available at Martynas Mažvydas National Library of Lithuania, the Library of the Institute of Mathematics and Informatics, the Library of Vytautas Magnus University.

VYTAUTO DIDŽIOJO UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS INSTITUTAS

Vaidas Giedrimas

KOMPONENTINIŲ PROGRAMŲ SISTEMŲ GENERAVIMO METODAS

Daktaro disertacijos santrauka
Fiziniai mokslai (P 000)
Informatika (09 P)
Informatika, sistemų teorija (P 175)

Vilnius, 2010

Disertacija rengta 2002–2006 m. Matematikos ir informatikos institute.
Doktorantūros teisė suteikta kartu su Vytauto Didžiojo universitetu 2004 m. gruodžio 13 d. Lietuvos Respublikos Vyriausybės nutarimu nr. 1285.
Disertacija ginama eksternu.

Disertacija bus ginama viešame informatikos mokslo krypties tarybos posėdyje 2010 m. gegužės 5 d. 11.30 val.
Adresas: Akademijos g. 4–203, Vilnius, LT-08663 Lietuva

Disertacijos santrauka išsiuntinėta 2010 m. balandžio 2 d.
Disertaciją galima peržiūrėti Lietuvos nacionalinėje Martyno Mažvydo bibliotekoje, Vytauto Didžiojo universiteto bibliotekoje, Matematikos ir informatikos instituto bibliotekoje.

RESEARCH PROBLEM AND MOTIVATION

Contemporary software development is critical in both time and quality aspects. Software depends on the dynamics of business domains, so it must be flexible, easily scalable and easily maintainable. The evolution of software engineering has passed through the various stages, including structured and object-oriented development, component-based software engineering (CBSE). Each stage is related with some level of granularity of modules. The CBSE uses modules of higher granularity – the components.

The CBSE enables to reduce significantly time required for the development, testing of component-based software, to create robust systems. However to use component-based paradigm is not enough. The methods, typically used for component-based software development are error-prone and cannot ensure high software quality. The existing component-based software generation methods focus on the generation performance instead of quality. There is the need for component-based software generation method to ensure consistence of specification and generation results also.

The research field of this dissertation is the automated software development methods.

The research object is the problem of the automated software composition.

RESEARCH OBJECTIVES

The purpose of this research is to create the component-based system generation method taking into account only the structural properties of the components, independent from the concrete component model properties and to ensure consistency between the specification and generated software.

The research problems are the following:

1. To investigate the properties of software component models and the characteristics of component-based software development process.
2. To perform comparative analysis of the software generation methods and to define the properties of these methods required for the automated component-based development.
3. To develop a method for generating the component-based software.
4. To implement the proposed method in order to experimentally evaluate its properties on the prototype.

RESEARCH HYPOTHESES

1. The component-based software generation problem and the solution can be stated in terms of *abstract component model* insomuch that the solution can be applied to the particular component model using the operations of concretization, refinement and complement.
2. The generation method based on *Curry-Howard* protocol can be used to achieve the consistency of component-based system specification and implementation.

RESEARCH METHODOLOGY

The information *search*, *systematization*, *comparative analysis* and *generalization* methods have been used to collect and present the facts about component-based paradigm and software generation methods.

The *clustering* and *expert evaluation methods* have been used to create the abstract software component model. The *clustering method* has been used to preliminary manage groups of the component models and their properties. The clustering has been performed using *SPSS 16.0*. In order the clustering cannot assess the factor of the component model semantics, the *expert evaluation method* has been used as a main method for this purpose.

The *mathematical modeling* methods have been used to create and elaborate the logical type theory and the constructive type theory required for *Curry-Howard* protocol implementation.

The *experiment* and *generalization* methods have been used to experimental evaluation of the SoCoSyS system as the implementation of the component-based software generation method. *The comparative analysis* method has been used to assess SoCoSyS and other similar systems.

RESEARCH FINDINGS

This work contributes following innovations:

- The different viewpoint of the software synthesis is proposed – to use one software generation method using the terms of the abstract software component model instead of using many synthesis methods for each software component model. The abstract software component model for the component-based software generation is defined.

- The *Curry-Howard* protocol is implemented for the component-based paradigm. There exist implementations of this protocol as proofs-as-programs method generalization for structural and functional programming paradigms, but the protocol has never been adopted for the component-based software engineering before.
- The proposed method for component-based software generation encapsulates two generative methods: deductive and transformative synthesis. The application of inductive method application is hypothesized also.

PRACTICAL IMPORTANCE

The results of this work are important for practical software engineering:

- The method proposed in this work enable to reduce the time required for the component-based software composition and to improve the quality of the results.
- The SoCoSyS system, created as the implementation of this method, can be directly applied to .NET component-based software development. There are possibilities to adapt SoCoSyS to other component models (e. g. WS, CCM etc.).

APPROBATION AND PUBLICATIONS

The research results were presented at the following national and international conferences and other events:

1. ***Komponento modelis struktūrinės programų sintezės kontekste***. (2003 08 29, XI'th scientific conference of Lithuanian Computer Society (KoDi'2003)
2. ***Programų sistemų automatizuoto surinkimo iš gatavų komponentų uždavinys*** (2003 06 20, XLIV LMD conference)
3. ***Formal specification of .NET component.*** (2004 08 10, International summer school ESSCSS'04, Estonia)
4. ***Component model and its formalization for structural synthesis*** (2004 10 14 XLV'th international scientific conference of Ryga Technical university, Latvia)
5. ***Komponento specifikacijos formalizavimas*** (2004 06 18, XLV LMD conference)

6. ***NET komponento specifikacija programų sintezės kontekste***. (2005 01 26, Information Technologies-2005 (IT'2005) conference).
7. ***An application of SSP method to component-based development process***. (2005 08 09, ESSCSS'05, Estonia)
8. ***Component-based Software Generation: The Structural Synthesis Approach.*** (2005 09 27, International conference GPCE'05, Estonia)
9. ***Komponentinių programų struktūrinės sintezės teorinės problemos***. (2005 06 20, XLVI LMD)
10. ***Architectures of Component-Based Structural Synthesis Systems.*** (2006 07 03, Baltic DB&IS'06)
11. ***Component-based Software Synthesis: the union of two methods.*** (2006 08 07, ESSCSS'06, Estonia).
12. ***Induktyvinis metodas komponentinių programų sintezėje.*** (2006 06 20, XLVII LMD)
13. ***Generavimo metodų panaudojimas kuriant .NET komponentines programų sistemas.*** (2007 08 14, KoDi'2007)
14. ***Grid tinklo panaudojimas programinės įrangos sintezei.*** (2008 11 29, XIV'th Lithuanian Symposium on Arts and Sciences (MKS'14), USA)
15. ***Modelines architektūros naudojimas kuriant komponentines programų sistemas.*** (2009 09 26, KoDi'2009)

The main results of the dissertation were published in 11 papers: **7** papers published in the referred publications from the international referred database list approved by the Science Council of Lithuania; **3** papers published in proceedings of reviewed international conferences publications; **1** paper published in the proceedings of the local scientific conference. The detail list of publications is available in the end of this summary.

STRUCTURE OF THE DISSERTATION

This dissertation is written in Lithuanian. It consist **6** chapters, the list of references and appendixes. There are **168** pages of the text, **33** figures, **17** tables and **190** bibliographical sources.

SYNOPSIS

The **chapter** 1 is introductory. It presents the research problem and motivation, the aim and the objectives, research findings and the practical importance. The information about the approbation of the dissertation and the publications is provided also.

The **Chapter** 2 presents the problem statement. First, it provides the definition of the software component and the analysis of selected component models. The selection performed according to various taxonomies in order to cover many component models with the model-specific properties. During the analysis of selected component models, representing existing classes of component models, 28 concepts are identified in total. Each component model is described using from 3 to 12 concepts. This multiplicity of the component models is the burdensome factor for the automated component-based software development process.

Second subsection discuses four component abstraction levels. Two levels – *component specification* and *component implementation level* – are chosen for further analysis in this dissertation. The *component deployment* and the *component object* levels are not investigated in detail as they are loosely related with the design-time process.

Third subsection is dedicated to the component development process. This process is similar to usual software development process though has the following singularities determined: the components are developed for mandatory reuse; the complete set of system requirements is not known in the component development time; there is a need of a very precise component specification and documentation.

Finally, the component-based software engineering process is described, having special focus on the component-based software development (CBSD) process (Figure 1). The CBSD process focuses on the reuse of the components instead of the implementation of new components. The reuse is the key factor that enables to significantly reduce time required for the development, testing of component-based software, to create robust systems. Main disadvantages of CBSD process are as follows: the unsolved problem of the detailed component search; the big amount of the resources needed to adapt the components if they fit the specification partially only.

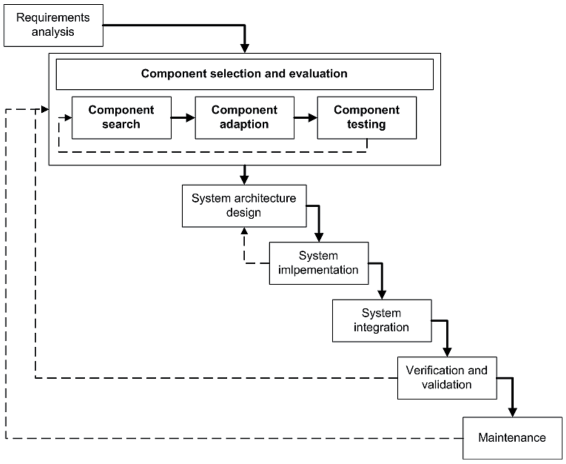

Figure 1. The life cycle of component-based software development

The **Chapter 3** is analytical. The object of the analysis is the existing software generation methods (Figure 2). For the historical reasons describing particular methods in this dissertation the term *generation* is interchanged with *synthesis*.

This comparative analysis of the software generation methods starts from the fundamental approaches of *E. Dijkstra*, *C.A.R. Hoare*, *M. Charpentier*, *R. Backhouse* and *J. Schumann.* Many contemporary methods were developed on the basis of these methods and this analysis helped to compare the properties of other generative methods.

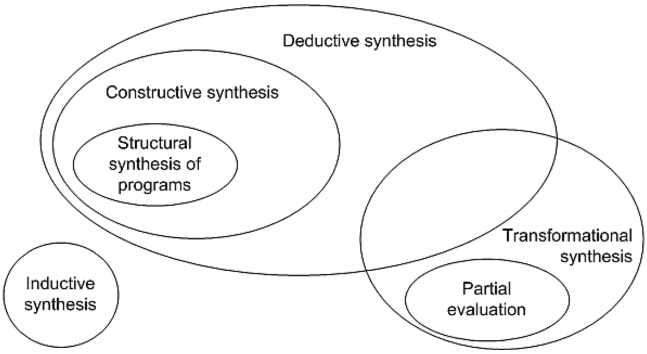

Figure 2. The relations of software generation methods

Next, the deductive, inductive and transformational generation methods are analyzed, having special focus on the class of the deductive methods.

The structural synthesis of programs (SSP) method is a subset of the set of deductive methods. SSP is based on the idea that the programs can be constructed taking into account only their structural properties, i.e. input/output variables. Each pre-programmed module is supplied with an axiom stating under which condition it can be applied, and what values it computes. The axiom does not describe the relationship between the input and output values, thus details of the implementation are hidden. The structure of the program is synthesized, and the concrete program is built from the pre-programmed modules. There are four algorithms for the proof-search. SSP method has been implemented in several systems to generate structural, object-oriented and service-oriented software. For the component-based software SSP can be applied because it operates with the "black-box" abstractions and are able to ensure the consistency of the component-based system specification and the implementation.

There is the class of the deductive methods based on the automated theorem prover (ATP) with the constructive logic. This class is called as constructive synthesis class and the methods are called as *proofs-as programs* methods. SSP is subset of this class also.

All *proofs-as programs* methods is based on the *Curry-Howard* isomorphism, which states the correspondence between systems of the formal logic as encountered in proof theory and the computational calculi as found in type theory. Although the *proofs-as-programs* methods have been developed independently by various researchers, they are generalized by the *Curry-Howard* protocol. The protocol denotes the main rules of the method application to number of paradigms. There is stated that *Curry-Howard* protocol can be used for the automated component-based software development although it never has been used for the component-oriented paradigm.

Inductive synthesis is the third software synthesis method, covered in the analysis. The study of this method showed that it cannot ensure the consistence of the specification and synthesis results. However the inductive method can be applied as the complement of the deductive methods helping to solve the following problems: the problem of incomplete specifications and the problem of missing components.

Last software generation method analyzed in this chapter is the transformational synthesis. Although some researchers classify this method as the subset of the set of deductive methods, we follow the statement that these methods are different. The transformation-based methods enable to shrink the gap between the concepts of business domain for which the software is developing and the concepts of the implementation domain (component models, frameworks, operating systems etc.).

The **Chapter 4** describes the research design. The main aim of this chapter is to describe the *Curry-Howard* protocol-based generative method for the automated component-based software composition, proposed by the author of dissertation.

We argue that it is preferably to create one component-based software synthesis system for the abstract software component model (Figure 3, b), instead of to create many systems for each specific component-model (Figure 3, a).

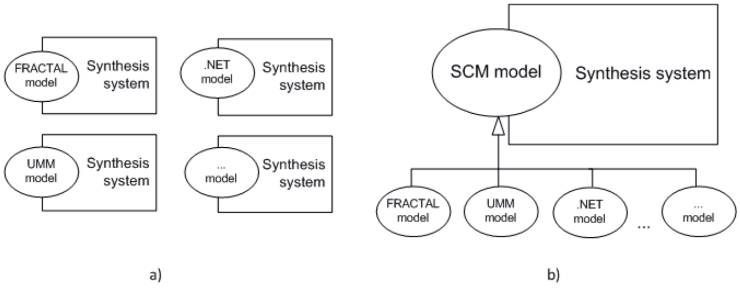

Figure 3. Possible models of interaction between software synthesis systems and component models

SOFTWARE COMPONENT MODEL

The formal abstract *software component model* (SCM) is created using the results of the analysis have done in the Chapter 1 (Table 1). To develop this component model two scientific methods have been used: the *clustering* and the *expert assessment*. Resulting *software component model* involves properties of other component models and provides possibility to perform the software generation having in mind the properties of the abstract component model only. Model consists of the following concepts: component, port, connection, constraint, credential and component-based system. The static and dynamic viewpoints of the software component model are described using formal methods and the UML diagrams (Figure 4).

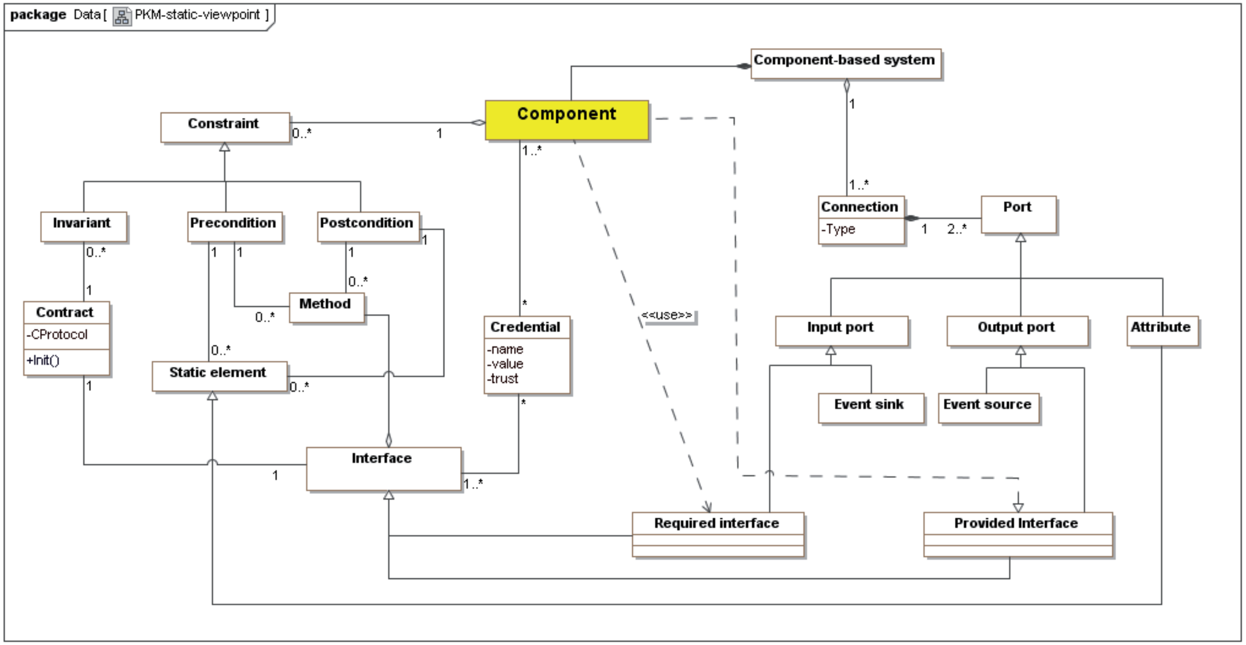

Figure 4. Software component model for component specification and implementation levels

Table 1. The elements of component models

	Structural entities							Descriptive entities										Connecting entities							Processing entities			
	Components							Interfaces				Contract							Wires									
	Basic	Source	Sink	Complex	Configuration	Object	Kens	Required	Provided	Service	Design	Basic c.	System c.	User c.	Implm. c.	NFP	Control atributes	Port	Binding	Mapping	Glue-code	Membrane	Gates	Script	Process	Sandbox	Transaction	Coordination
Aguirre & Maibaum	+	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-
Berger	+	-	-	+	+	-	-	-	-	-	-	-	-	+	-	-	-	-	+	-	-	-	+	-	-	+	-	-
Cervantes	+	-	-	-	-	-	-	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-
Cox & Song	+	+	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-	-
FRACTAL	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	+	+	-	-	-	-	-
Yoshida & Honiden	+	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-
Lau group	+	-	-	+	-	+	-	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Mahmood	+	-	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Moschoyiannis	+	-	-	+	-	+	-	+	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	+
Nierstrasz group	+	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	+	-	-	-	+
ObjectWeb	+	-	-	-	-	+	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PECOS	+	-	-	+	-	-	-	-	+	-	-	-	-	-	-	+	-	+	-	-	-	-	-	-	-	-	-	-
Poernomo	+	-	-	-	-	-	+	-	-	-	-	+	-	-	-	-	-	-	+	+	-	-	+	-	-	-	-	-
Salzmann	+	-	-	+	+	-	-	-	-	+	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	+	+	-
Szypersky	+	-	-	+	-	+	-	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-
UML	+	-	-		-	+	-	-	+	-	-	-	-	+	+	-	-	-	-	-	-	-	-	-	-	-	-	-
UniFrame	-	-	-	-	-	+	-	-	+	+	-	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	+
UNU/IIST	+	-	-	+	-	-	-	+	+	-	+	+	+	-	-	-	-	-	-	-	+	-	-	-	+	-	-	-
Whitehead	+	-	-	-	+	+	-	+	+	+	-	-	-	-	-	-	-	-	-	-	+	-	-	+	+	+	+	+

Main concept of the component model is the component. Component is defined as a tuple:

$$C = \left\langle PRT_{out}, PRT_{in}, \Delta, \Omega^{C} \right\rangle ,$$

where *PTR*$_{out}$ is the set of the output ports, *PRT*$_{in}$ – the set of the input ports, Δ – the set of the constraints, Ω^{C} – the set of the credentials component.

The port is general concept for components communication. Each required interface, event sink or static property can be considered as the input port and each provided interface, event source or static property can be considered as the input port. The sets of the input and output ports are defined as the unions:

$$PRT_{in} = \Im_{in} \cup E_{in} \cup S$$
$$PRT_{out} = \Im_{out} \cup E_{out} \cup S \text{ ,}$$

where

$$\Im_{out} = \left\{ I_1^{out}, I_2^{out}, \ldots, I_n^{out} \right\}; \Im_{in} = \left\{ I_1^{in}, I_2^{in}, \ldots, I_m^{in} \right\},$$

are set of the provided and required interfaces, *E*$_{in}$ – set of the event sinks, *E*$_{out}$ – set of the event sources, *S* – set of the static properties.

Each interface can be defined as tuple:

$$I_i = \left\langle O_i, S_i, \Xi, \Omega_i^I, \right\rangle ,$$

where *O*$_i$ is the set of the interface operations:

$$O_i = \{o_{i,1}, o_{i,2}, \ldots, o_{i,k}\} ;$$
$$o_{i,j} = \left\langle P, R, PreC, PostC, \Omega_{i,j}^{O} \right\rangle ,$$

S$_i$ – static element of the interface (e.g. property), *Θ* – set of the invariants of the interface. Each interface operation is described using set of the parameters *P*, set of the results *R*, precondition *PreC*, postcondition *PostC* and credential Ω^{O}.

The credentials in this model are used to express non-functional properties although they can be used to express other properties also.

There are three types of the credentials: credentials of the component (Ω^C), credentials of the interface (Ω^I) and credentials of the operation (Ω^O). Despite of the type each credential can be defined as tuple:

$$\Omega^C = \langle KV, KR, KPTM \rangle ,$$

where *KV* is the name of the it, *KR* – value and *KPTM* – confidence (trust), expressed by assessment method: expert view, experimental evidence etc.

The *component-based system* in this model is defined using the non-empty set of the couples:

$$\langle KS_i, JK_i \rangle ,$$

where KS_i is the list of the components included in the *i*-th couple, JK_i is the glue-code abstraction. Glue-code abstraction can be considered in two different ways: as a wrapper and as a script. In this dissertation the wrapper-like implementation of the glue-code abstraction have been chosen. In this case JK_i is defined as a set of the connections:

$$jng_{i,j} = \langle P_{in}, P_{out}, JTipas \rangle , \quad jng_{i,j} \in JK_i,$$

where each connection is defined by the connection type *JTipas*, and couple of the ports P_{in}, P_{out}.

This abstract software component model (SCM) is described in the first section of the Chapter 4 in detail.

CURRY-HOWARD PROTOCOL IMPLEMENTATION FOR THE COMPONENT-BASED PARADIGM

Second section of the Chapter 4 is dedicated to the implementation of *Curry-Howard* protocol for component paradigm. The protocol implementation process involves the following steps:

1. The logical calculus must be chosen for further deductive reasoning;
2. The logical type theory (LTT) for this calculus must be defined.
3. The implementation language (e.g. programming language) must be chosen and defined as the Computational type theory.

4. The proof that Curry-Howard protocol to hold over the domains must be performed. Usually this proof is based on the two mapping tables: *etype* and *extract*.

For component-based implementation of *Curry-Howard* protocol the intuitionistic propositional calculus (ITS) has been chosen as deductive reasoning base:

$$ITS = \langle Formulae(ITS), \vdash_{ITS}, DR \rangle \,,$$

where *Formulae(ITS)* – formulae of the intuitionistic propositional calculus, DR – deduction rules (Table 2).

Table 2. ITS inference rules (DR)

Introduction rule	Elimination rule
$\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} (\wedge I)$	$\dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} (\wedge E_L) \quad \dfrac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} (\wedge E_R)$
$\dfrac{\Gamma, u : A \vdash B}{\Gamma \vdash A \rightarrow B} (\rightarrow I^u)$	$\dfrac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} (\rightarrow E)$

Each propositional variable (notated by capital letter, e.g. *A*) corresponds to particular port of the component. Each software component can be defined by the set of the axioms. If the relations of the input and output ports are known in advance (e.g. Figure 4, b) the set consist of few axioms:

$$((A1 \wedge D) \wedge R) \rightarrow I1$$
$$A2 \rightarrow E$$
$$R \rightarrow I2 \,.$$

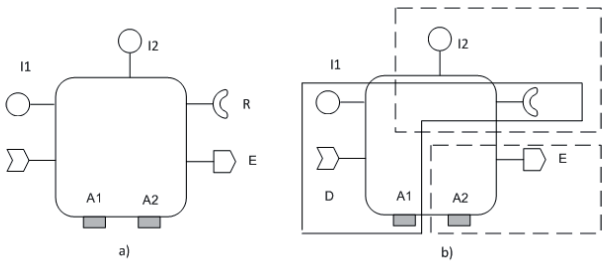

Figure 4. The example of the logical relations of the component ports

If the relations of the ports are unknown (e.g. Figure 4, a), the component is defined by only one axiom:

$$A1 \wedge A2 \wedge D \wedge R \to I1 \wedge I2 \wedge E$$

There is special case of the propositional variables – control variables, which are notated using square brackets, e.g.

[PageIsPrinted].

Second step performed in the process of the *Curry-Howard* protocol implementation to component-based paradigms – Logical type theory (LTT) for the *ITS* is defined:

$$LTT(ITS) = \left\langle PT(LTT), Formulae(LTT), (.)^{(.)}, \vdash_{LTT}, PTR, \rhd \right\rangle,$$

where *PT(LTT)* is the set of the proof-terms, *Formulae(LTT)* – set of the types of the proof-terms, $(.)^{(.)}$ – typing relation between proof-terms and types, PTR – inference rules. Possible proof-terms and it types have shown in the Table 3. The inference rules of the LTT are listed in the Table 4.

Table 3. LTT types.

$PT(LTT)$ element	LTT type
$\langle M, N \rangle$	$A \wedge B$
$fst\ N$	A , if $N^{A \wedge B}$
$snd\ N$	B , if $N^{A \wedge B}$
$\lambda u : A.M$	$A \to B$

Table 4. LTT inference rules (PTR)

	Introduction rules	Elimination rules
Conjunction	$\dfrac{\Gamma \vdash M^A \quad \Gamma \vdash N^B}{\Gamma \vdash < M, N >^{A \wedge B}} (\wedge I)$	$\dfrac{\Gamma \vdash M^{A \wedge B}}{\Gamma \vdash fst\ M^A} (\wedge E_L)$ $\dfrac{\Gamma \vdash M^{A \wedge B}}{\Gamma \vdash snd\ M^B} (\wedge E_R)$
Implication	$\dfrac{\Gamma, u^A \vdash M^B}{\Gamma \vdash \lambda u.M^{A \to B}} (\to I^u)$	$\dfrac{\Gamma \vdash A \to M^B \quad \Gamma \vdash N^A}{\Gamma \vdash MN^B} (\to E)$

The proof-terms can be normalized using reduction relation $\rhd$, which is defined by the following reduction rules:

$$\begin{aligned} fst\ (< a, b >^{(A \wedge B)}) &\quad \rhd_{LTT} \quad a^A \\ snd\ (< a, b >^{(A \wedge B)}) &\quad \rhd_{LTT} \quad b^B \end{aligned}$$

Third step performed in the process of the *Curry-Howard* protocol implementation to component-based paradigms – Computational type theory (CTT) is defined:

$$CTT = \langle Terms(CTT), Types(CTT), :, \vdash_{CTT}, TIR, \triangleright \rangle$$

Table 5. LTT to CTT terms mapping function *etype(F)*.

LTT term F	$etype(F)$
P	$Component$
(P)	$Component$
$(A \wedge B)$	$\begin{cases} etype(A) & \text{if } \neg H(B) \\ etype(B) & \text{if } \neg H(A) \\ etype(A) * etype(B) & \text{otherwise} \end{cases}$
$(A \rightarrow B)$	$\begin{cases} etype(B) & \text{if } \neg H(B) \\ etype(A) \rightarrow etype(B) & \text{otherwise} \end{cases}$

Finally, extraction maps between LTT and CTT terms (Table 5) and between LTT and CTT types are created (Table 6). Using these maps the correspondence between LTT and CTT can be proved. Maps also are important to the process of the CTT program generation using the LLT proof. Extraction map (Table 6) ensures that during the extraction the *Harrop* formulas (noted by predicate *H(A)*) are ignored. This helps to reject irrelevant, non-constructive information, which can be found in the LTT proofs.

Table 6. LTT to CTT formulae mapping function *extract()*.

LTT formulae (p^T)	$extract(p^T)$
Any proof-term p^T	$()$, if $H(T)$
u^A	$\begin{cases} x_u & \text{if } \neg H(A) \\ () & \text{if } H(A) \end{cases}$
$\langle a^A, b^B \rangle$	$(extract(a),\ extract(b))$
$fst\ a$	$fst\ (extract(a))$
$snd\ a$	$snd\ (extract(a))$
$c^{A \rightarrow B} a^A$	$\begin{cases} extract(c) & \text{if } H(A) \\ (extract(c)\ extract(a)) & \text{if } \neg H(A) \end{cases}$

THE ELEMENTS OF THE SSP, INDUCTIVE AND TRANSFORMATIONAL SYNTHESIS

The deductive component-based software method proposed in this dissertation is based on the *Curry-Howard* protocol implementation, however it encompass few elements of other methods. The 3rd and 4th section of the chapter 4 are dedicated to these elements.

Main element of the Structural Synthesis of Programs (SSP) used to our method is the set of the proof-search algorithms of SSP.

The disadvantages of the deduction-based methods: the problem of the incomplete specifications and the problem of the missing components can be reduced using inductive software generation methods.

Although there is no subsection dedicated to method of the transformational synthesis, elements of this method can be observable in the entire Chapter 4. First, practical *Curry-Howard* protocol implementation requires the transformation service to implement LTT and CTT mappings (Table 5 and Table 6). Next, the reasoning about components and component-based system is performed in terms of the abstract software component model, consequently to complete software generation process in terms of the particular component model transformations are needed also.

The Chapter 4 ends by two small examples of the method application to component-based software generation.

THE IMPLEMENTATION OF THE COMPONENT-BASED SOFTWARE GENERATION METHOD

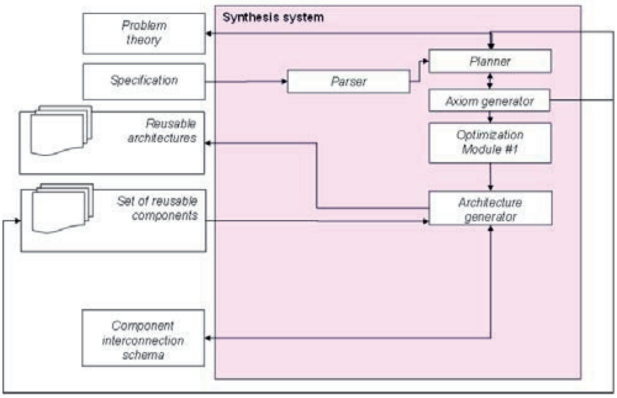

Figure 4. The SoCoSyS system for component specification abstraction level

The **Chapter 5** describes the research experiment. It have been demonstrated how to proposed component-based software generation method can be applied in practice.

In order to cover both component specification level (Figure 4) and component implementation level (Figure 5) two cases are investigated.

The SoCoSyS system has been developed to investigate these cases. SoCoSyS consists of the following parts: *specification parser*, *planner*, *axiom generator*, *generator of the architecture* and two *optimization modules*

The *parser* as an input takes functional requirements of the specification unfolds the input and output variables and makes the problem datastructures – a graph of the problem representation.

The *planner* is the main part of the synthesis system. It as an input takes the problem datastructures and domain theory datastructures as an input. The result of the planner is a constructive proof of the problem theorem. External theorem automated theorem prover (ATP) can be used as a planner. However the analysis of four ATPs (*HERBY & THEO, Isabelle, Coq, SNARK*) have shown that they fit not all requirements, i.e. few of them have not interfaces to communicate with other programs, few of them are interactive provers and cannot be used to purely automatic planning process. To fulfill all the requirements new automated theorem prover *SoCoProove* has been developed by the author of this dissertation. *SoCoProove* is simple automatic ATP, using proof-search algorithms of the Structural Synthesis of programs.

The goal of the *optimization module #1* is to eliminate unnecessary steps in the proof. These additional steps can be generated by assumption driven forward search by planner.

Architecture generator makes the component interconnection schema from the optimized proof.

The SoCoSyS system for component implementation abstraction level is more complex (Figure 5). It consist extra modules: *axiom generator*, *program generator* and *optimization module #2*.

Program generator makes the code of the program from the optimized proof. If generated program is a source-code-program, then some virtual machine or external compiler is needed.

The *optimization module #2* as an input takes non-functional requirements of a specification (via parser) and the results of the generator as an input. Having information about program architecture second optimizer can select the components that meet the non-functional requirements better and replace already used components in the program.

Axiom generator is optional module. The purpose of it is to minimize the problem of missing axioms. This problem has been discussed in the

Chapter 3 in detail. Existing synthesis system cannot give the solution for the users, even when 100 components and axioms, defining these components are present, and only 1 or 2 components are missing. In this case system returns the decision that component-based system cannot be generated. In that case *axiom generator* tries to add one or more virtual axioms to the set of the axioms and restarts the process of the *planning*. If the solution have found by the planner, it is clear that component-based system can be developed if particular component can by created. What input and output ports this component must have is defined by the virtual axiom. The implementation of the *axiom generator* is very complicated, therefore is not present in the SoCoSyS system temporary. It is the subject of the future work.

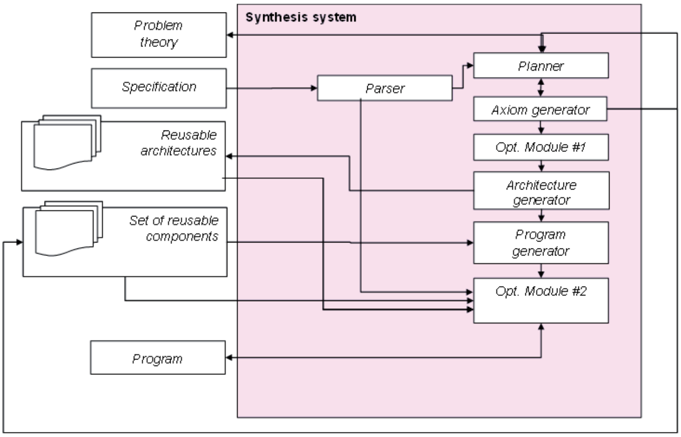

Figure 5. The SoCoSyS system for component implementation abstraction level

After SoCoSyS system is created, it has been used to generate software using the *.NET* components. *.NET* technology have been chosen independently, it do not put any additional constraints on the method. SoCoSyS system can be easy adapted to other component models also.

The experiments have being organized in two different stages.

First, the principal abilities of the system have been investigated. A set of .NET components with different input and output ports have been generated. For this purpose special tool *SoCoGen* have been used. *SoCoGen* generates both .NET components and specifications of them in the XML. These specifications have been used by SoCoSyS system to generate component-based software.

The experiment was repeated 50 times, using different number (from 10 to 200) of the components in the set. The number of different ports and the number of ports in one component have been varying also. This experiment has shown that the method can be implemented, and component-based software can be generated.

The goal of second stage of the experiment is to validate the method possibilities in semantic point of view. Therefore few synthesis problems from the pharmacy and multimedia domains has been solved using SoCoSys system. This experiment has been shown that method can be used for automated component-based software development.

SoCoSys system has been compared to four other systems implementing similar methods: *NORA/HAMMR*, *QUASAR*, *Component WorkBeanch (CWB)* and *CoSMIC*. The comparative analysis shows that

- In contrast of the *NORA/HAMMR,* based on deductive method too, SoCoSys system is dedicated to component-based software development.
- All analysed software generation systems are oriented to different aims, so the methods, they are using differs to. Only SoCoSys system implements the method which encompass elements of more that one generation method.
- The higher automatization level in the SoCoSys system has observed.

The **Chapter 6** contains the main conclusions.

At the end of the dissertation there are references and appendixes. First appendix exposes the list of the publications publishing the main results of dissertation. Second appendix shows the results of the component-models clustering in detail.

CONCLUSIONS AND FUTURE WORK

The following conclusions were drawn from this research:

1. The multiplicity of the component models is the burdensome factor for the automated component-based software development process:
   - During analysis of the selected component models, representing existing classes of the component models, 28 concepts are identified in total. Each component model is described using from 3 to 12 concepts.
   - Component-based software is developed using various composition forms. What the particular composition form

should be used depends on the component model and it abstraction level.

- There are four component abstraction levels. In this work two of them are investigated: component specification level and implementation level. The component deployment and component object levels are not investigated as they are loosely related with the design-time process.

2. This study of the properties of the software component models and singularities of the component-based development process has shown that:
   - The singularities of component development process are as follows: components are developed for mandatory reuse; the complete set of the system requirements is not known in the component development time; there is a need of very precise component specification and documentation.
   - Main disadvantages of the component-based software development process are as follows: the unsolved problem of the detailed component search; the big amount of resources needed to adapt components if they fit the specification partially only.
3. In order to solve particular problems of automated component based-system development the following generation methods can be used: structural synthesis, inductive synthesis and transformative synthesis.
   - *Deductive methods* operate with "black-box" abstractions. The usage of the deductive method ensures the consistency of component-based system specification.
   - Main disadvantages of the deduction-based method: the problem of incomplete specifications and the problem of missing components can be reduced using the *inductive* software generation *methods*.
   - The transformation-based methods enable to shrink the gap between the concepts of business domain for which software is developing and the concepts of the implementation domain (component models, frameworks, operating systems etc.).
4. The component-based software generation problem and the solution can be stated in terms of *abstract component model* insomuch that the solution can be applied to the particular component model using the operations of concretization, refinement and complement.

5. To achieve the consistency of the component-based system specification and the implementation, the method based on Curry-Howard protocol can be used. In this case the main parts of the protocol can be defined in the component-based terms: the logical type theory based on Intuitionistic propositional calculus, the Computational type theory for the extraction of implementations from the proofs.
6. Practical experiment with the SoCoSyS system as the implementation of the method presented in this dissertation shows that method can be used for the automated component-based software development.

This research has thrown up many questions in need of further investigation. Further work needs to be done to establish:

1. The hypothesis of the inductive generation method usage for the component-based development should be checked in the experimental research.
2. The method application for other software component levels: *component deployment* and *component object* must be investigated.

PUBLICATIONS

Articles in referred publications from the international referred database list approved by the Science Council of Lithuania

1. **Giedrimas, V.** (2009). Modelinės architektūros naudojimas kuriant komponentines programų sistemas. *Informacijos mokslai, 50*, p. 168–172.
2. **Giedrimas, V.** (2007). Generavimo metodų panaudojimas kuriant .NET komponentines programų sistemas. *Informacijos mokslai, 42–43*, p. 246–250.
3. **Giedrimas, V.** (2006). Induktyvinis metodas komponentinių programų sintezėje**.** *Liet. mat. rinkinys, t. 46, spec. nr.,* p. 103–106.
4. **Giedrimas, V.**, Lupeikienė, A. (2005). Komponentinių programų struktūrinės sintezės teorinės problemos. *Liet. mat. rinkinys, 45, spec. nr.*, p. 139–143.
5. **Giedrimas, V.**, Lupeikienė, A. (2004). Komponento specifikacijos formalizavimas**.** *Liet. mat. rinkinys*, *t. 44, spec. nr.*, p. 276–280.
6. **Giedrimas, V.** (2003). Programų sistemų automatizuoto surinkimo iš gatavų komponentų uždavinys *Liet. mat. rinkinys, t. 43, spec. nr.*, p. 228–232.
7. **Giedrimas, V.** (2003). Komponento modelis struktūrinės programų sintezės kontekste. *Informacijos mokslai, 26,* p. 246–250.

Publications in proceedings of reviewed international conferences

1. **Giedrimas, V.** (2006). Architectures of Component-Based Structural Synthesis Systems. Databases and Information Systems. *Seventh International Baltic Conference on Databases and Information Systems. Communications.* Vilnius: Technika, p. 331–315.
2. **Giedrimas, V.** (2005). Component-based Software Generation: The Structural Synthesis Approach. *Proceedings of 7th GPCE YRW*, Tallinn. ISBN 9985-894-89-8, p. 19–23.
3. Lupeikiene, A., **Giedrimas, V.** (2005). Component model and its formalisation for structural synthesis**.** *Scientific Proc. of Riga Technical University. Computer Science, vol. 22*. ISSN 1407-7493, p. 169–180.

Other publications

1. **Giedrimas, V.** (2005). NET komponento specifikacija programų sintezės kontekste. *Konferencijos „Informacinės technologijos 2005“ medžiaga*. Kaunas, p. 382–385.

Information about the author of the dissertation

Vaidas Giedrimas was born in August 31, 1976.

1994–1998: Studies at Šiauliai University, Faculty of Physics and Mathematics – Bachelor degree of Educology and qualification of teacher of Mathematics and Informatics.

1998–2000: Postgraduate studies at Šiauliai University, Faculty of Physics and Mathematics – Master degree of Educology.

2002–2006: PhD studies at Institute of Mathematics and Informatics, Software Engineering Department.

He is author of the book “Component-oriented programming” (in Lithuanian). From the 2002 he is a member of the Lithuanian Computer Society and Lithuanian Mathematical Society.

E-mail: vaigie@mi.su.lt

REZIUMĖ

Tyrimų sritis ir problemos aktualumas

Šiuolaikinis programinės įrangos kūrimo procesas yra kritinis tiek laiko, tiek kokybės požiūriu. Programų sistemos priklauso nuo dinamiškai besikeičiančio verslo, todėl turi būti paprastai modifikuojamos ir lengvai aptarnaujamos.

Vienas šios problemos sprendimo būdų – kurti komponentines programų sistemas. Tačiau ir komponentinė paradigma negali užtikrinti reikiamos programinės įrangos kokybės, todėl būtini nauji komponentinių programų sistemų kūrimo metodai.

Šios disertacijos tyrimų sritis yra komponentinių programų sistemų surinkimo iš komponentų proceso automatizavimas. Tyrimų objektas – programų sistemų automatizuoto surinkimo iš komponentų uždavinys.

Tyrimų tikslas ir uždaviniai

Tyrimų tikslas – sudaryti deduktyviosios sintezės metodą komponentinėms programoms kurti, atsižvelgiantį į struktūrines komponentų savybes ir nepriklausantį nuo konkretaus komponento modelio iš nagrinėjamos komponentų klasės.

Tikslui pasiekti išsikelti šie uždaviniai:

1. Ištirti programinių komponentų savybes ir komponentinių programų kūrimo proceso ypatumus.
2. Išanalizuoti programų sintezės ir generavimo metodus įvertinant šių metodų taikymo galimybes komponentinėms programoms kurti.
3. Pasiūlyti komponentinių programų surinkimo iš gatavų komponentų proceso automatizavimo metodą.
4. Siekiant eksperimentiškai patikrinti pasiūlytąjį metodą, sukurti ir įvertinti jį realizuojančios sistemos prototipą.

Mokslinis naujumas

Darbe gauti šie nauji moksliniai rezultatai:

- Pasiūlyta, kaip abstrahuojantis nuo konkrečių komponentų modelių ir jų savybių, programų generavimą vykdyti abstrakčiojo *Programinio Komponento Modelio* terminais.
- Siekiant užtikrinti komponentinių programų atitikimą specifikacijai, pasiūlyta pritaikyti *Curry-Howard* protokolą ir struktūrinės programų sintezės algoritmus automatizuotam programų kūrimui naudojant *Programinio komponento modelį*.
- Numatyta kelių programų generavimo metodų integravimo galimybė.

Praktinė darbo reikšmė

Praktiniu požiūriu disertacijos rezultatai reikšmingi dėl šių priežasčių:

- Siūlomas metodas įgalins sutrumpinti komponentinių programų sistemų kūrimo ir atnaujinimo laiką bei pagerinti jų kokybę;
- Realizuotoji SoCoSys sistema gali būti tiesiogiai naudojama sistemoms generuoti iš .NET komponentų. Dėl metodo universalumo SoCoSys sistema yra lengvai pritaikoma įvairiems komponento modeliams.

Darbo rezultatų aprobavimas ir publikavimas

Pagrindiniai darbo rezultatai yra publikuoti 11 mokslinių leidinių: 7 straipsniai – recenzuojamuose Lietuvos žurnaluose; 3 straipsniai – recenzuojamuose Lietuvos žurnaluose ir leidiniuose bei tarptautinių konferencijų darbuose; 1 straipsnis – kituose Lietuvos leidiniuose.

Darbo apimtis ir struktūra

Darbas parašytas lietuvių k. Jį sudaro 6 skyriai, literatūros sąrašas, publikacijų sąrašas ir priedai. Darbo apimtis – 168 puslapiai, 33 paveikslai ir 17 lentelių. Darbe cituojami 190 literatūros šaltinių.

Pirmajame skyriuje „**Įvadas**“ glaustai aprašoma tyrimų sritis ir aktualumas, tyrimo objektas, tikslas ir uždaviniai. Taip pat pateikiami publikacijų bei skaitytų pranešimų sąrašai.

Antrajame skyriuje („**Komponentinių programų sistemų kūrimo proceso samprata ir modeliai**“) pristatoma tyrimo problema. Šio skyriaus tikslas – išskirti komponentinių programų sistemų kūrimo proceso sampratos ypatumus ir realizavimo modelius. Analizuojama komponento modelių ir komponento abstrakcijos lygmenų įvairovė. Aprašoma komponentų ir komponentinių programų sistemų kūrimo specifika, analizuojamas tokių programų sistemų kūrimo uždavinys, identifikuojami probleminiai komponentinių programų sistemų gyvavimo ciklo etapai.

Trečiasis skyrius („**Programų sistemų generavimo metodų lyginamoji analizė**“) – analitinis. Šio skyriaus tikslas – įvertinti generavimo metodų savybių pritaikomumo komponentinėms programų sistemoms kurti ir metodų plėtros tokių sistemų kūrimui galimybes. Iš pradžių nagrinėjami klasikiniai formalieji metodai, kurie yra kitų – išvestinių – programų sintezės metodų pagrindas. Vėliau analizuojama deduktyvinės programų sintezės (dar vadinamos įrodomojo programavimo) uždavinių klasė ir formalią įrodomojo programavimo metodų kūrimo procedūrą apibrėžiantis *Curry-Howard* protokolas. Šiame skyriuje suformuota induktyvinės sintezės panaudojimo komponentinių programų generavimui hipotezė.

Ketvirtajame skyriuje („**Komponentinių programų kūrimo proceso automatizavimo metodas**“) aprašoma komponentinių programų sistemų generavimo metodika. Pirmiausia formalizuojamas *Programinio komponento modelis*. Disertacijoje pasiūlyta, kaip abstrahuojantis nuo konkrečių komponentų modelių ir jų savybių gauti abstraktų *Programinio komponento modelį*, pritaikytą automatizuotai komponentinių programų sintezei.

Šioje disertacijoje komponentinei paradigmai realizuotas *Curry-Howard* protokolas:

1. Kaip natūraliosios dedukcijos sistema pasirinktas intuicionistinis teiginių skaičiavimas (ITS).
2. ITS pagrindu aprašyta loginė tipų teorija.
3. Aprašyta skaičiuojamoji tipų teorija..
4. Sudarytos loginės tipų teorijos ir skaičiuojamosios tipų teorijos termų ir formulių atitikties lentelės.

Penktasis skyrius („**Komponentinių sistemų generavimo metodo realizacija**“) – eksperimentinis. Jame aprašoma SoCoSys sistema sukurta remiantis 3–4 skyriuose aprašytomis idėjomis, realizuojant 3 skyriuje pasiūlytą komponentinių programų sistemų generavimo metodą ir jos darbo rezultatai generuojant komponentines programas iš konkrečių komponentų aibės.

Sistema palyginta su kitomis komponentinių programų kūrimui skirtomis sistemomis.

Šeštajame skyriuje pateikiamos išvados:

1. Komponento modelių įvairovė yra veiksnys, apsunkinantis automatizuotą komponentinių programų sistemų kūrimo procesą.
    - Apžvelgus komponento modelius, reprezentuojančius žinomas modelių klases, nustatyta, kad jiems aprašyti iš viso naudojamos 28 sąvokos. Kiekvienas modelis aprašomas naudojant nuo 3 iki 12 sąvokų.
    - Komponentinės programos gaunamos naudojant skirtingas komponavimo formas. Kurios būtent formos yra naudojamos, priklauso nuo komponento modelio ir jo abstrakcijos lygmens.
    - Skiriami keturi komponento abstrakcijos lygmenys. Disertacijoje nagrinėjami *specifikacijos* ir *realizacijos* lygmenų komponentai. Įdiegto komponento ir komponentinio objekto lygmenys šioje disertacijoje nenagrinėjami, nes *įdiegto komponento* ir *komponentinio objekto* lygmenys glaudžiai susiję su sistemos veikimo etapu ir kūrimo procesui yra mažiau aktualūs.

2. Išanalizavus programinių komponentų savybes ir komponentinių programų kūrimo proceso ypatumus, nustatyta:
   - Pagrindiniai komponentų kūrimo proceso ypatumai yra šie: komponentai kuriami tam, kad būtų panaudoti daugiau nei vieną kartą; jie kuriami nežinant, kokie reikalavimai jiems bus keliami konkrečiose sistemose; komponentų specifikacija ir dokumentacija turi būti tiksli ir išsami.
   - Komponentinių programų sistemų kūrimo procesas, kuriame akcentuojamas ne suprojektuotos sistemos ir jos dalių realizavimas, bet jau sukurtų komponentų pakartotinis panaudojimas turi trūkumų: neišspręstas detalios komponentų paieškos uždavinys; komponentų adaptavimui reikalingos papildomos sąnaudos.
3. Konkrečioms komponentinių programų sistemų surinkimo proceso automatizavimo problemoms spręsti gali būti naudojami struktūrinės sintezės, induktyviosios sintezės ir transformacinės sintezės metodai:
   - Deduktyvieji metodai palaiko „juodosios dėžės“ abstrakcijas. Be to, taikant deduktyvųjį struktūrinės sintezės metodą, užtikrinama rezultatų (taikomųjų programų) atitiktis specifikacijai.
   - Induktyvusis metodas gali padėti spręsti šias deduktyviuoju metodu neišsprendžiamas problemas: specifikacijos neišsamumo problemą, neapibrėžtųjų komponentų problemą ir nefunkcinių reikalavimų problemą.
   - Transformacinė sintezė įgalina mažinti atotrūkį tarp dalykinės srities, kuriai kuriama programinė įranga, ir realizacinės srities (konkrečių komponento modelių, karkasų, operacinių sistemų ir t. t.) konceptų.
4. Programų sintezės uždavinys ir jo sprendimo būdas gali būti taip suformuluotas apibendrinto *programinio komponento modelio* terminais, kad gautas sprendinys gali būti pritaikomas kiekvienam iš apibendrintų komponentų modelių panaudojant konkretizavimo, patikslinimo ir papildymo operacijas.
5. Programų sintezės metodas, komponentinėje paradigmoje realizuojantis *Curry-Howard* protokolą, įgalina automatizuotu būdu kurti komponentines programas ir užtikrina šių programų atitiktį specifikacijai.
6. Komponentinių programų sistemų generavimo metodo eksperimentinė realizacija SoCoSyS sistemoje parodė, kad metodas tinka tokioms programų sistemoms kurti.

Trumpos žinios apie autorių

Vaidas Giedrimas gimė 1976 m. rugpjūčio 31 d.

1994–1998 m. studijavo Šiaulių universitete, Fizikos ir matematikos fakultete, įgijo Edukologijos bakalauro laipsnį bei matematikos ir informatikos mokytojo kvalifikaciją.

1998–2000 m. studijavo magistratūroje Šiaulių universitete, Fizikos ir matematikos fakultete, ir įgijo Edukologijos magistro laipsnį bei gimnazijos informatikos mokytojo kvalifikaciją.

2002–2006 m. studijavo doktorantūroje Matematikos ir informatikos institute, Programų sistemų inžinerijos skyriuje.

Jis yra mokomosios knygos „Komponentinis programavimas“ autorius. Nuo 2002 m. – Lietuvos matematikų draugijos ir Kompiuterininkų sąjungos narys.

El. paštas vaigie@mi.su.lt.

Vaidas Giedrimas

THE COMPONENT-BASED SOFTWARE GENERATION METHOD

Summary of Doctoral Dissertation
Physical Sciences (P 000)
Informatics (09 P)
Informatics, System Theory (P 175)

SL 843. 2010-04-02. 2 aut. apsk. l. Tiražas 80. Užsakymas 25.
Išleido VšĮ Šiaulių universiteto leidykla, Vilniaus g. 88, LT-76285 Šiauliai.
El. p. leidykla@cr.su.lt, tel. (8 ~ 41) 59 57 90, faks. (8 ~ 41) 52 09 80.
Spausdino UAB „Šiaulių knygrišykla“, P. Lukšio g. 9A, LT-76207 Šiauliai.