Legacy Model Reconfiguration Using Graph-Theoretic

Azween Bin Abdullah†

University Technology Petronas, Bandar Seri Iskandar, 31750, Tronoh, Perak, Malaysia

Summary
We present a theory of software model preservation and isomorphism (enhancement) and proving correctness of model of reengineering using graph and set theory (GS). A lot of software applications will be in the legacy state every five years as a result of changes in technology and business processes. Modeling has become a de facto standard in software engineering. Developing, deriving from existing models, composing and proving correctness of models are part and parcel of the software development process. We describe a specification technique that combines the methods of adaptive modeling and model composition.

Key words:
Model morphism, operational and failure consistencies, model composition, graphnet.

1. Introduction

Formal methods for the specification and verification of hardware and software systems are becoming more and more important as systems increase in size and complexity. Modeling on the other hand consists of the specification, model checking techniques for verification, analysis of properties, code generation and execution of models. In model analysis, fundamental concepts such as composition, abstraction and reusability of models, model verification and verification of properties are inherent model derivation activities. Model management is an important, but often neglected activity in requirements analysis and design. Generally defining a model encompasses multiple views or ways of doing things and keeping track of their relationships between and among different views and managing consistency as they evolve are major challenges (Barr, 1999).

Always all model analysis investigates the invariants and persistent properties that we wish to be inherent. No one talks about failure properties of models. This technique is one part of a general computational theory of conceptual design called model-based analogy. Structure-behavior-function (SBF) models of software systems, for example, specify the structure, the functions, and the internal causal behaviors that explain how the program structure results in its function (Goel, 1991).

The basis for the identification of failure consistency problems is obtained by relating sub-models to failure aspects and thereby discovering which sub-models model the same aspects of the system. After identifying failure consistency or inconsistency, we compare the cost factor. If failure is inconsistent then check for semantic consistency and accept the model that has the least overall cost factor and minimum failure consistency and maximum operational consistency match.

2. Basic Notions and Elements of The Graphnet Model

This section introduces the formal notions associated with the analysis model. We begin by identifying elements of the model which correspond to parts of the real system to be modeled.

2.1 Definition 1.1

A compnode is a 4-tuple $\partial = \{\beta, \delta, \alpha, \gamma\}$ made of four finite sets, β and δ, α, γ. The elements of β and δ are respectively called failures and operations.

α : Set of active agents. These are external and internal agents that will generate error conditions.

β : Set of failure properties for each node. These are conditions that will generate errors for each node. Each node might correspond to a coherent and comprehensive system (component or subsystem).

γ : Set of lethal events. These are error conditions that will affect the availability of the system.

δ : Set of operational properties. These are properties that we want to be present in the system or user requirements.

2.2 Definition 1.2

A non-deterministic graphnet is a 4-tuple $\psi = \{\phi, q, F, \omega\}$ where

ϕ: a finite set of compnode.

q: start compnode, a member of ϕ, $q \in \phi$.

F: a subset of ϕ, is a set of final compnodes.

Manuscript received November 5, 2008
Manuscript revised November 20, 2008

ω: A transition function f: $\partial(\gamma) \rightarrow \partial(\gamma)$, is a function that takes a compnode in ϕ and a parametric input from γ as argument and returns a subset of ϕ.

Beside this set-theoretic definition, graphnet are usually encountered as graphical objects. A decorated graph (Figure 1) is associated to a given graphnet $\partial = \{\beta, \delta, \alpha, \gamma\}$ as follows.

1. Its objects are compnodes and interactive transitions (undirected path). Components are pictured as circles, while interactive transitions are represented by an undirected line. For an interactive transition, there must be a minimum of two nodes.
2. An interactive transition ω is an undirected path between two nodes if there are parametric value exchange or service calls. It is an interactive transition because the transition is a two way process.
3. If x and y are compnodes and ω an interactive transition, there is a line from x to y. Such arrows are decorated with the cost-factor (CF).

CF_i : The bi-directional associated cost of failure between two nodes(path) if a failure event is triggered. The cost is the same both ways.

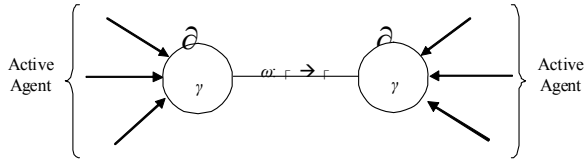


Fig. 1 A graphical representation of a graphnet with interactive transition.

2.3 Definition 1.3

Let $\psi = \{\phi, q, F, \omega\}$ be a non-deterministic graphnet. A critical path in the graphnet is a closed-path in the graphnet that has the highest total cost factor (TCF) value.

Example 1.4 The non-deterministic graphnet in Figure 1 can be specified formally as

$$\psi = \{\{N_1 \ldots N_6\}, N_1, \{N_2 \ldots N_6\}, \omega\}$$

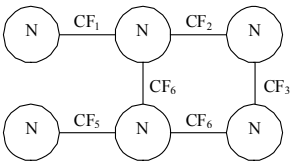


Fig. 2 A graphnet with 6 compnode with interactive transitions and its associated cost-factors.

Path 1: $N_1 - N_2 = CF_1$
Path 2: $N_1 - N_2 - N_3 = CF_1 + CF_2$
Path 3: $N_1 - N_2 - N_3 - N_6 = CF_1 + CF_2 + CF_3$
Path 4: $N_1 - N_2 - N_3 - N_6 - N_5 = CF_1 + CF_2 + CF_3 + CF_6$
Path 5: $N_1 - N_2 - N_3 - N_6 - N_5 - N_4 = CF_1 + CF_2 + CF_3 + CF_6 + CF_5$
Path 6: $N_1 - N_2 - N_3 - N_6 - N_5 - N_2 = CF_1 + CF_2 + CF_3 + CF_6 + CF_4$
Path 7: $N_1 - N_2 - N_5 = CF_1 + CF_4$
Path 8: N1-N2-N5-N4 = CF1 + CF4 +CF5

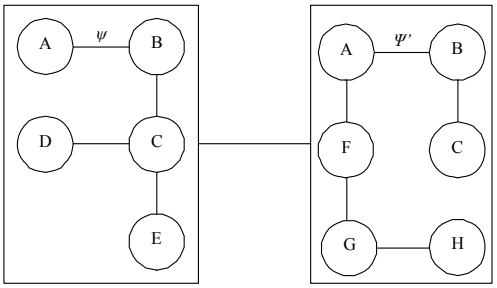
The critical path is Max $\{Path_1 \ldots Path_n\}$ and the mean cost of failure for the model

$$M = \frac{\sum_{i=1}^{n} Path_i}{n}$$

2.4 Definition 1.4

Let $\psi = \{\phi, q, F, \omega\}$ be a graphnet. Its associated transformed graph morphism is the graphnet ψ' that have the following properties.

1. $\delta' = \delta \cup \{n\}$ where the cardinality of set $\{n\} \geqq 1$. The new model in Figure 3(b) should perform something more by a factor of at least one compared to the old model in Figure 3(a).
2. M' = M ie the mean cost of failure for the new model must be less than the mean cost of the old model.
3. $\bigcup_{i=1}^{n} \delta' \beta_i < \bigcup_{k=1}^{m} \delta \beta_k$ ie the sum total of all failure properties in Figure 3(b) must be less than the sum total of the failure properties in Figure 3(a).
   The total failure properties of the old model in Figure 3(a) is $\Omega = \bigcup_{k=1}^{m} \delta \beta_k$ where m= cardinality of ϕ.



3(a) 3(b)

Fig. 3(a) The model before model transformation and Fig. 3(b) after the model transformation.

2.5 Checking Consistency of Models

Model rebuilding generally involves the addition, deletion of content modification of compnodes (without affecting the number of compnodes in the original model). When we have decided on the structure of the new model after performing graph morphism by adding or deleting compnodes, we have to ensure that the plugged-in and deleted compnodes will maintain the consistency requirements for the new models. The new model should have at least the same properties as the old model. Having a model with the same properties as before only gives us an isomorphic structure which may or may not be cost-effective and adds up its complexity. Compnode addition may involve the addition of new compnodes or replacement of a compnode with more than one compnodes. In consistency checking we will only look at the operational properties.

There are 4 cases to assess with regards to model consistency checking.

Case 1: Replacement or addition of one or more compnodes, excluding the start compnode.

(a). The replacement of a single compnode.

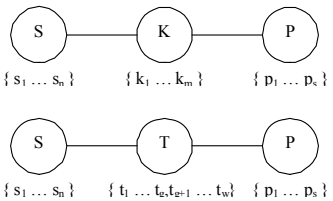


Fig. 4 .

In the figure 4 above compnode K is replaced with a new compnode T. Compnode T has completely new properties that have to be consistent with compnode S and compnode P.

ω:$\{s_1 ...s_n\} \rightarrow \{t_1 \ldots t_g\}$ and ω:$\{ t_{g+1} \ldots t_w\} \rightarrow \{ p_1 \ldots p_s\}$.

(b). The replacement on a compnode with more than one compnodes.

Let us assume that we have three compnodes, S, K and P each having the sets of operational properties $\{s_1 ...s_n\}$, $\{k_1 \ldots k_m\}$ and $\{p_1 \ldots p_s\}$ respectively. Figure 5 shows the links among the compnodes.

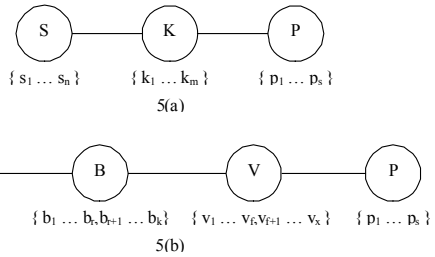


Fig. 5(a) The original model, 5(b) the compnode K is replaced with compnodes B and V.

$B \cup V = \{k_1 \ldots k_m\}$

If $|\{k_1 \ldots k_m\}| <= |\{B \cup V\}|$ then there are some properties in the new compnode that were not present in the previous node. We call these excess properties as *residual properties*. We assume that when we plug-in the compnodes B and V, there is no need for consistency checking. We need to check for consistency in the S---B and B---P link.

ω**:**$\{s_1 ...s_n\} \rightarrow \{b_1 \ldots b_r\}$

ω**:**$\{s_1 ...s_n\} \rightarrow \{b_1 \ldots b_r\} \cup \{k_1 \ldots k_m\}$. This is consistent if there is a bijection.

$\{b_1 \ldots b_r\} / \{k_1 \ldots k_m\}$ = {residual properties}. These properties are necessary for consistency link with V and are not present in the compnode K.

Similarly, ω:$\{k_1 \ldots k_m\} \rightarrow \{p_1 \ldots p_s\}$

ω:$\{v_{f+1} \ldots v_x\} \rightarrow \{p_1 \ldots p_s\}$

Case 2: Deletion of one or more compnodes, excluding the start compnode.

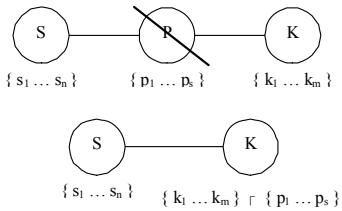


Fig. 6 Compnode deletion and re-attachment.

In Figure 6 the operation properties set of compnode P is partitioned into two sets, $\{p_1 \ldots p_j\}$ and $\{p_{j+1} \ldots p_s\}$. The set $\{p_1 \ldots p_j\}$ correspond to linking properties with operations properties of compnode S and $\{p_{j+1} \ldots p_s\}$ corresponds to linking properties of compnode K. $\{p_1 \ldots p_j\}$ must be present in compnode K to enable it to be linked with compnode S and $\{p_{j+1} \ldots p_s\}$ must be present in compnode S to be linked with compnode K. We then have to perform an bijective mapping from S to K to check for consistency.

$\omega: S\{s_1 \ldots s_n\} 4 \{p_{j+1} \ldots p_s\} \rightarrow K\{\{k_1 \ldots k_m\} 4 \{p_1 \ldots p_j\}\}$

Case 3: Replacement and deletion of the start compnodes. We can neither delete nor replace the start compnode but only enhance or degrade its operation properties. Enhancement of properties, means additions of operational properties and degrade means the deletion of operational properties. In both cases, we will then have to perform an injective mapping with the attached link as shown in Figure 7.

Enhancement : $\{s_1 \ldots s_n\} \cup \{s_p\}$ where $|\{s_p\}| >= 1$
Degrade : $\{s_1 \ldots s_n\} \setminus \{s_p\}$ where $|\{s_p\}| >= 1$ and $s_p \in \{s_1 \ldots s_n\}$

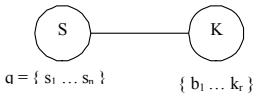


Fig. 7 The enhance and degradation of a compnode.

Case 4: Replacement and deletion of the final compnodes.

We cannot delete a final compnode as it would leave a dangling graphnet. We can enhance or degrade its operation properties, just like the case of start nodes or replace it with a different compnode. In the first instance, we use the procedure of case 3 and for the latter we use the procedure of case 1(a).

3. Case Studies

Two case studies have been selected to show our approach. The first case study will show the case where the model is refactored by adding new elements while the second case study shows the deletion of model elements to improve certain aspect of the system. The first case study is also an example of reengineering legacy system. The case studies presented here are from two real projects.

3.1 Organization Appraisal System

Organization Appraisal System (OAS) is an information system that helps organization in managing their performance. OAS is based on Balanced Scorecard technique, where the organization's performance is measured using Key Performance Indicator (KPI). The organization's performance is measured annually and quarterly (achievement of every three months). Part of the KPI is calculating the difference between plan (achievement targeted by organization) and actual (actual achievement of organization after certain period of time) value. There are several formula used in calculating the difference:

- Negative value with positive meaning
- Negative value with negative meaning
- Current performance = total of quarterly achievement
- Current performance = average of quarterly achievement
- Current performance = the latest quarterly achievement

The calculation will depend on the nature of the KPI.

In the previous version of OAS, there are two classes that represent the KPI, KPI and QuarterKPI. Method to calculate the difference exists in both classes and in this method IF statements are used to control which formula to be use in calculating the difference. Figure 8 show the KPI and QuarterKPI classes in the previous version of OAS.

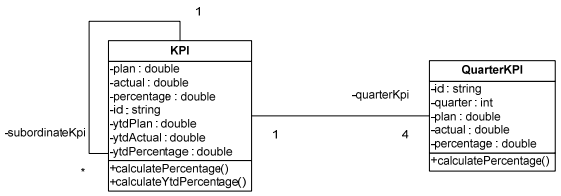


Fig. 8 KPI and QuarterKPI classes.

IF statements to control the formula used in calculating the difference is not the best solution because changing the formulas of certain KPI will affect other KPI that used different formula. To overcome this problem, Strategy design pattern can be used. Strategy design pattern will create individual classes for the formula. Having individual classes to represent the formulas allow the effects of changes is confined to only the KPI that uses the formula, thus reducing the errors as result of the changes. Another benefit of Strategy design pattern is in introducing new formula. New formula can be added to the system by creating a class to represent the new formula. Figure 9 shows the changes in OAS.

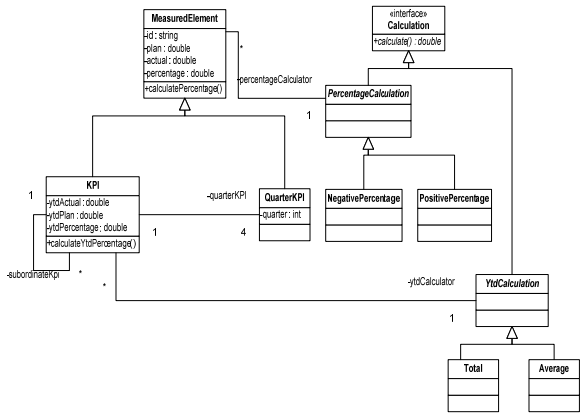


Fig. 9 KPI and QuarterKPI classes.

OAS is an example of addition of multiple compnodes. Each class in Figure 8 and Figure 9 are compnodes. MeasuredElement, and the hierarchy of Calculation classes are residual properties, compnodes that did not exist in Figure 8. Model in Figure 9 is also consistent with the model in Figure 8 because KPI and QuarterKPI class did not lose any of its properties (attributes, and operation).

3.2 Publication Monitoring System

Libraries and universities usually keep record of different types of publication. Publication Monitoring System (PMS) is one such system that keeps an inventory of publications. Different types of publication that can be stored by PMS are books, journals, and conference papers.

The current design of PMS is shown in Figure 10. From the design we can see that for every type of publication there will be two classes involve, one class is for the user interface and another to represent each publication. The justification for such design is different publication have different set of data. For example books have ISBN, while conference papers have conference name and keywords.

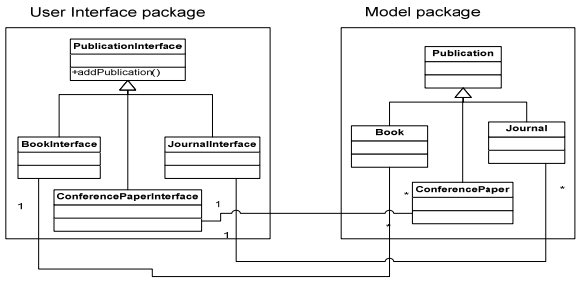


Fig. 10 Design of PMS.

XMLTemplate framework (Ab. Rahim et al, 2007) is a new framework created for systems such as PMS. One notable feature that makes PMS suitable for XMLTemplate is PMS has functionalities that work for different type of the same element. In the PMS case it is publications and the functionalities are add, delete and edit. Figure 11 show the design for PMS when using XMLTemplate framework.

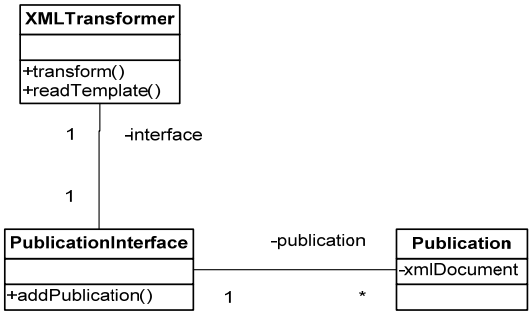


Fig. 11 PMS design using XMLTemplate framework.

How XMLTemplate framework work is each type of publication will have XML document that store user interface information. XMLTransformer class will read the template and create the user interface. When a user entered data for a new publication the data will be stored in XML format. This will allow Publication class to represent all types of publication. The advantages of XMLTemplate framework are:

1. user interface can be customize at runtime
2. new type of publication can be added at runtime by just adding a new XML document
3. number of software elements that need to be develop is reduced

PMS is the example where elements in the model are deleted. Similar to the first example, all classes are considered as compnodes. By using XMLTemplate framework BookInterface, JournalInterface, ConferencePaperInterface, Book, ConferencePaper and Journal are deleted from the model (as shown in Figure 11). To keep the model in Figure 11 consistent with model in Figure 10, two elements are added which is XMLTransformer class and xmlDocument attribute in Publication.

4. Model Analysis and Transformation

Our model transformation and reengineering process entails a four-stage hierarchical analysis process. The first stage involves the identification of the compnodes to determine the overall structure of the model and the

subsequent determination the graphnet. The second stage involves the determination of the critical path in the graphnet. This assessment will give us a better understanding of the path that would have the most cost-effect implication on our model. A refinement of the critical path is necessary to determine whether a new model-segment can be found with a lower cost-failure value. The fourth stage involves the determination of a new model (model composition) using the set-theoretic process and model consistency.

Our main focus here is on addition and removal operations since we believe that by combining these operations most of the changes in a system can be modeled. Specifically, we focus on evolution process that causes addition or removal of components in a system. Though, using the model we present here, it is also possible to formalize model evolution that may cause relation between compnodes to change.

There are four probable cases that may arise in the analysis.

1. If failure properties and operational properties are the same for two models then the model is isomorphic.
2. If failure properties are consistent and operational properties are different then reject the model.
3. If failure properties are not consistent and operational properties are consistent then choose the model with the minimal failure cost impact.
4. If failure properties and operational properties are not consistent than choose the model with the minimal failure cost.

There is a possibility that we might get a model with a high failure cost and consistent with additional operational properties. In this case we have to remodel and continue the process again with the additional properties. In this case we talk of composing a subgraph (model) to the old graph (model).

5. Related Approaches

In this paper, we have discussed the issue of modeling of software components with a focus on how to achieve consistency of interaction on the model level. We first discussed the issue of modeling in general and presented a specification mechanism and identified that the composition of software components and their interaction can be studied on the model level if the abstraction from system properties is preserved.

6. Conclusion

Model driven software engineering primarily deals with manipulation and transformation of models. In the current state of research on MDE, however, there is an urgent need for more disciplined and more formal techniques to support a wide range of model evolution activities. These include model refactoring, model inconsistency management, model versioning and merging, co-evolution of models, incremental model analysis and verification and many more. A desirable property of the enabling or supporting mechanisms for these activities is that they should remain, as much as possible, ‘agnostic’ of the particular modeling language of interest. This makes them more robust to evolution of the modeling language, and allows them to be applicable to a wider variety of models.

In this paper we have proposed a method of graph transformation as an underlying theory and technology for model evolution. Due to its solid formal foundation, combined with the fact that models are frequently represented in a graph-based way, graph transformation seems to be a natural choice for supporting model evolution. We try to validate this claim by exploring how the technique of graph transformation can be used to support model refactoring and model inconsistency resolution for reengineering legacy systems.

7. Future Work

The current version of the new strategy for generating and acquiring functional models is limited in at two aspects, but we are planning to address these limitations: (1) the structure of the new design can have one or more additional components relative to those in the original model, but it cannot have fewer components and (2) the additional components are connected serially with other components corresponding to the components in the original design. We also plan to formally analyze properties of the strategy of composing newer models by components additions and their properties.

References

[1] Alanen, M., Porres, I. 2003. Difference and union of models. In Stevens, P., Whittle, J., Booch, G., eds.: UML 2003 – The Unified Modeling Language. Volume 2863 of Lecture Notes in computer Science., Spinger-Verlag: 2-17.
[2] Barr, Michael and Charles Wells. 1988. Category Theory for Computing Science. Prentice Hall.
[3] Barr, Michael and Charles Wells. 1999. Category Theory for Computing Science. Les Publications CRM Montreal, Third Edition.
[4] Belady, L. Lehman, M.M. 1976. A Model of large program development. IBM Sys. J.15(1): 225-252.
[5] Bennet, K.H., Rajlich, V. 2000. Software maintenance and evolution: a roadmap. ICSE – Future of SE Track: 73-87.

[6] Ehrig, H. 1979. Introduction to the algebraic theory of graph grammars. In Claus, V., Ehrig, H., Rozenberg, G., eds.: Graph Grammars and Their Application to Computer Science and Biology. Volume 73 of Lecture Notes in Computer Science., Spinger-Verlag: 1-69.
[7] Fiadeiro, J.L. 2005. Categories for Software Engineering. Springer, Leicester, United Kingdom.
[8] Goel, Ashok K. 1991. A Theory of Incremental Model Learning. In Proceedings of The Eighth International Conference on Machine Learning, Los Altos, CA, Morgan Kaufmann: 605-609.
[9] Jacobson, I., Booch, G., Rumbaugh, J. 1999. The Unified Software Development Process. Addison Wesley Professional.
[10] Kemerer, Chris F. and Sandra Slaughter. 1999. An Empirical Approach to Studying Software Evolution. IEEE Transactions on Software Engineering, Vol. 25: 493-509.
[11] Lehman, M.M., Perry, D.E., Ramil, J.C.F., Turski, W.M., Wernick, P. 1997. Metrics and laws of software evolution. Fourth International Symposium on Software Metrics: 20-32.
[12] Medvidovic, N., Rosenblum, David S., Redmiles, David F., Robbins, Jason E. 2002. Modeling software architectures in the Unified Modeling Language. ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 11: 2-57.
[13] Mens, T., Van Eetvelde, N., Demeyer, S., Janssens, D. 2005. Formalizing refactorings with graph transformations. Journal on Software Maintenance and Evolution: Research and Practice: 2-31.
[14] Van den Broek, P.M. 1991. Algebraic graph rewriting using a single pushout. In Abramski, S., Maibaum, T., eds.: TAPSOFT'91. Volume 493 of Lecture Notes in Computer Science., Spinger Verlag: 90-102.
[15] Wemelinger, M., Fiadeiro, J.L. 2002. A graph transformation approach to software architecture reconfiguration. Sci. Comput. Program. 44(2): 133-155.

Azween Abdullah obtained his bachelors degree in Computer Science in 1985, Master in Software Engineering in 1999 and his Ph.d in computer science in 2003. His work experiences includes eighteen years as a lecturer/senior lecturer in institutions of higher learning and as director of research and academic affairs at two institutions of higher learning, twelve years in commercial companies as Software Engineer, Systems Analyst and as a computer software developer and IT/MIS and educational consultancy and training. He has many years of experience in the application of IT in business, engineering and research and has personally designed and developed a variety of computer software systems including business accounting systems, software for investment analysis, website traffic analysis, determination of hydrodynamic interaction of ships and computation of environmental loads on offshore structures. His area of research specialization includes system survivability, formal specifications and modeling and software engineering.