PRINCIPLES OF MODELS UTILIZATION IN SOFTWARE SYSTEM LIFE CYCLE

Ján KUNŠTÁR, Iveta ADAMUŠČÍNOVÁ, Zdeněk HAVLICE
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic, tel. +421 55 602 2573
E-mail: jan.kunstar@tuke.sk, iveta.adamuscinova@tuke.sk, zdenek.havlice@tuke.sk

ABSTRACT

Modeling is one of the most important factors in the process of computer systems development. It is the process of representing real-world concepts in the computer domains as a blueprint for the purpose of software development. Recent trends in software and system development have also revealed the value of developing systems at higher levels of abstraction. Abstract models streamline and speed up not only development but suitable models can also improve maintenance process to be more effective and safe. This paper briefly analyses SysML, which supports development process of complex systems unlike UML which is strictly focused on software. Main part is oriented to presentation of a new approach to model driven system development supporting SysML concept named System Development Unified Process extended by concept of Model-Driven Maintenance (MDM). MDM is based on new architecture of software systems characterized by conjunction of system models with application's code. MDM supports direct changes of system based on modifications in system's models.

Keywords: *model-driven development, model-driven maintenance, software life cycle, SysML, System Development Unified Process, UML*

1. INTRODUCTION

Nowadays, models present one of the most important considerations of system and software development. Model-based design supports exploratory design and analysis by allowing designers to effectively represent and investigate their knowledge about the system during the decomposition and definition process. Models are used to represent formally the structure, function, and behavior of a system [8]. Additionally, experiments can be performed on models to eliminate poor design alternatives and to ensure that a preferred alternative meets stakeholders' objectives. Models also facilitate designer collaboration by providing a common formalism for communicating information about the system. This modeling concept stays at the core of Model Driven Architecture (MDA). However, one of the most important factors of modeling, in order to support the MDA development process, is the choice of modeling language. To support model-based design and to overcome some limitations related to Unified Modeling Language (UML) strict software focus, the Object Management Group (OMG) has developed the Systems Modeling Language (OMG SysML™) [13]. SysML is general-purpose systems modeling language that allows system designers to create and manage models of physical systems using well-defined, visual constructs. The knowledge captured by a SysML model is intended to support the specification, analysis, design, verification and validation of complex systems.

In this paper, a methodology for model-based system development using the SysML is presented with emphasis on maintenance phase of proposed life cycle.

Presented paper is organized as follows. First, essential information about MDA and SysML are briefly analyzed in Section 2. Then, Section 3 introduces proposal of System Development Unified Process (SDUP). Section 4 introduces proposal of Model-Driven Maintenance. Finally, Section 5 presents the conclusions.

2. MODEL-DRIVEN SYSTEM DEVELOPMENT

Model Driven Architecture, defined and supported by the Object Management Group [11], defines an approach to IT system specifications that separates the system functionalities from the implementation details on a particular technological platform. The MDA [12] is a framework for model driven software development defined by the OMG which has elevated the software development to the next step. Using MDA, it is possible to have an architecture that will be language, vendor and middleware neutral. In other words this concept corresponds to cross platform interoperability, portability, platform independence and productivity.

This approach places the emphasis on models, provides a higher level of abstraction during development and enables significant decoupling between Platform Independent Models (PIMs) and Platform Specific Models (PSMs).

One of the key standards that make up the MDA is UML [5]. UML has, since its adoption in 1997, proved immensely popular with software engineers, but its software focus has discouraged many system engineers from adopting it earnest. Those who did adopt UML developed strategies to cope with its shortcomings. The OMG customization of UML for systems engineering in form of new modeling language called SysML is intended to support modeling of a broad range of systems, which may include hardware, software, data, personnel, procedures, and facilities.

2.1. SysML

OMG SysML is a visual modeling language for systems engineering that extends UML 2 in order to analyze, specify, design and verify complex systems, intended to enhance systems quality, improve the ability to exchange systems engineering information amongst tools and help bridge the semantic gap between systems,

software and other engineering disciplines [13]. OMG SysML reuses a subset of UML 2 concepts and diagrams and augments them with some new diagrams and constructs appropriate for systems modeling (Fig. 1).

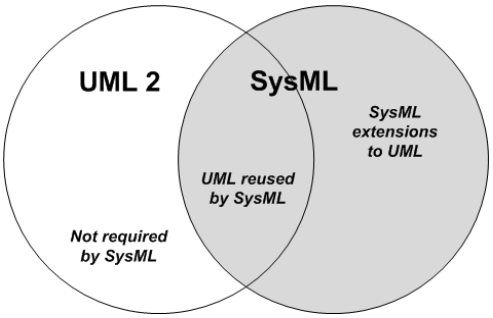

Fig. 1 UML 2/SysML Relationships

The benefits of using SysML in system development process are following [5, 13]:

- SysML semantics are better suited for systems engineering. SysML reduces UML's software-centric restrictions and adds two new diagram types for requirements engineering and performance analysis.
- SysML allocation tables support various kinds of allocations. These tables support requirement, functional and structural allocation, thereby facilitating automated verification and validation and gap analysis.
- SysML improves communication by providing a formal language for sharing system information. Based on UML, SysML ensures the flow-down from systems engineering to software engineering is more accurate.
- SysML's requirement modeling support provides the ability to assess the impact of changing requirements to a system's architecture.
- SysML is a precise language, including support for constraints and parametric analysis which allows models to be analyzed and simulated.

SysML is an open standard and supports XMI and ISO 10303-303 (AP233) allowing for information interchange to other systems engineering tools [11, 13].

OMG SysML includes diagrams that can be used to specify system requirements, behavior, structure and parametric relationships. These are known as the four pillars of OMG SysML [13]:

I. Structure.

The block is the basic unit of structure in SysML and can be used to represent hardware, software, facilities, personnel, or any other system element. The system structure is represented by block definition and internal block diagrams.

II. Behavior.

The behavior diagrams include the use case diagram, activity diagram, sequence diagram, and state machine diagram. The extensions made to standard UML activity diagrams include the support for representation of time-discrete and continuous systems, control of the size and behavior of buffers on incoming data flows and compatibility with widely used EFFBD notation that will facilitate and improve interaction between SysML and traditional software engineering tools and facilitate the migration to SysML [14].

III. Requirements.

The requirement diagram is a new SysML diagram type that captures requirements hierarchies and the derivation, satisfaction, verification and refinement relationships. The requirement diagram provides a bridge between typical requirements management tools and the system models [3]. Hence requirements become an integral part of the product architecture [13].

IV. Parametrics.

The parametric diagram is a new SysML diagram type that describes the constraints among the system's properties associated with blocks. This diagram is used to integrate behavior and structure models with engineering analysis models such as performance, reliability, and mass property models. Parametrics also enable integration of specification and design models with engineering analysis models [13].

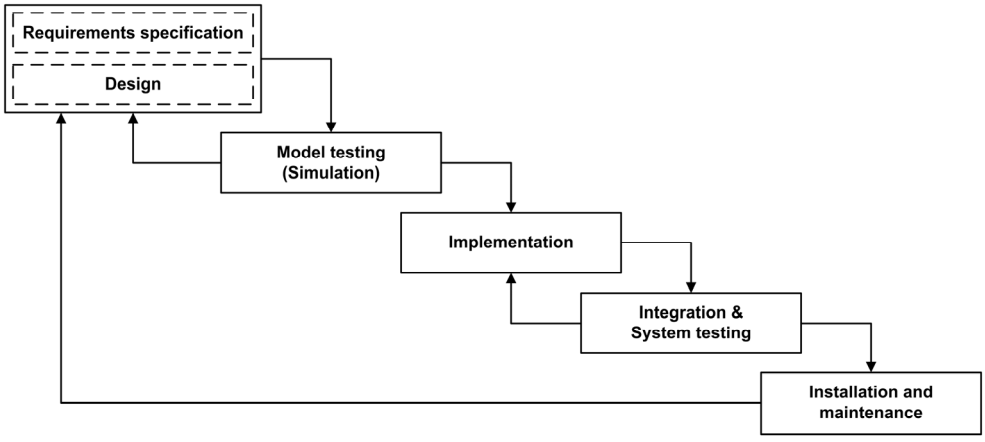

Fig. 2 System development unified process

3. SYSTEM DEVELOPMENT UNIFIED PROCESS

Presented model of system development life cycle includes all phases typical for the most of common life cycle models, e.g. waterfall model.

However, within this model, the modifications regarding the MDA development approach using SysML were required. The model emphasizes the maintenance phase and its impact on the whole system development process (Section 4).

In general, the presented model (Fig. 2) may be considered as having five distinct phases, described below:

Integrated phase that includes phases of requirements specification and design of the system. By means of using SysML as modeling language, it is possible to integrate these two previously distinct phases into one using the parametric, requirement and design models [3, 13, 14]. This step involves gathering and defining the system's requirements that are directly related to design models with a high level of abstraction that is independent of any implementation technology (*platform independent models*).

Model testing. This phase consists of using the models created in previous step to be methodically verified to ensure that they are error-free and fully meet the specified requirements. This testing can be processed in form of simulation using the properties of SysML parametrics [7].

Implementation. In this step, the platform independent models are transformed into system's *platform specific models* that are linked to specific technological platforms (e.g. programming language, operating system or database) [12]. These models are afterwards transformed into implementation artifacts as executable code and database schemas.

Integration and System testing. In this stage, both individual system components and the integrated whole are methodically tested and evaluated regarding to technological platforms and quality and reliability of system's performance.

Installation and Maintenance. This step occurs once the system has been tested and certified as fit for use, and involves preparing the system for installation and use at the customer site. A maintenance part involves making modifications to the system or individual component to alter attributed or improve performance [1]. These modifications arise either due to change of requirements, or defects uncovered during system's testing. The main difference compared to standard system maintenance is that no change in system can be processed without accordant modification in design/requirement models (Section 4).

4. MODEL-DRIVEN MAINTENANCE

Model-Driven Maintenance is the maintenance approach based on models of software system. It uses knowledge from essential models of the system not only for faster implementation of required changes, but also for uncovering the impacts of these changes and for controlling of consistency between application's code and essential models.

Mentioned consistency is crucial for preservation of maintainability of the system which presents nowadays a common problem of software systems.

4.1. Main problems of software maintenance

Maintenance is the last phase of software life cycle (Fig. 2). Unlike development, maintenance has to work within the parameters and constraints of an existing system. Therefore knowing and understanding of software system is crucial phase during maintenance process at which models can help a lot.

Software maintenance task is the most expensive part of the software system life cycle (the surveys indicate that it consumes 60% to 90% of the total life cycle costs [6, 15]). Survey also shows that around 75% of the maintenance effort is spent on the enhancements, and error correction consumes about 21% (Fig. 3).

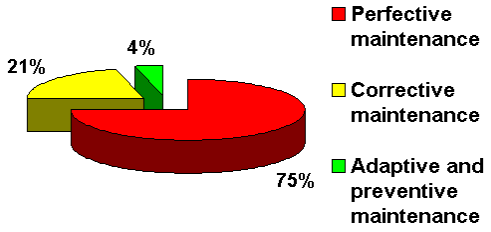

Fig. 3 Costs of maintenance activities

This proves that the maintenance cost is not only a consequence of the poor design, but mostly a consequence of the changing customer or environmental requirements and the manner in which the system was constructed [1].

Program comprehension, impact analysis and regression testing are the most challenging problems of software maintenance in the present [4].

An inconsistent state of the software artifacts significantly contributes to all three mentioned problems. Each software system consists of artifacts (e.g. source code, documentation, makefile, models of system) which describe only a limited part of the software and the actual system is their composite. When a software system is changed, each artifact affected by this change has to be modified for preservation of software maintainability. If one artifact is changed during the maintenance and the other one is omitted, it leads to inconsistent state of software artifacts – the result is that all artifacts do not describe the same system (i.e. system in its actual state).

4.2. Model-driven maintenance process

Model Driven Maintenance process presents one useful aspect of knowledge-based software life cycle oriented to better usability of all analysis, design and implementation models within the maintenance of systems [6]. To find the way how to streamline and speed up maintenance activities together with preserving maintainability of software system, which is crucial for future system's modifications during the software

evolution, is important for improving current state in software maintenance.

MDM is based on utilization of knowledge from the system models and dependencies among them for improving the maintenance process. Inspiration for MDM is the Model driven Architecture (Section 2). However, Figure 4 shows significant differences between these two approaches. MDA concentrates on development of software system using UML as programming language and the direction of progress is from models to application's code (Fig. 4). If the change of the system needs to be performed according the consistency rules of SDUP (Section 3), it is important to get back to system models, which means that the use of reverse engineering is inevitable. Reverse engineering process for MDA presents the so-called Architecture-Driven Modernization (ADM) [10].

In MDM, models of system are the basis for whole maintenance process and therefore there is a requirement to preserve essential models (type and count of these models depend on software type) together with the code of application. These essentials models are taken from project database (PDB) (a place of models preservation at majority of present software systems) and joined to conjunctive preservation in the installation phase.

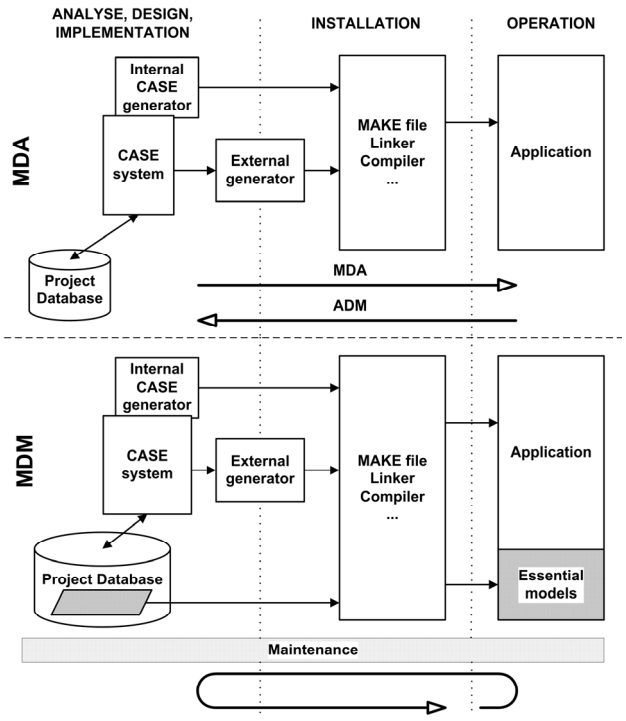

Fig. 4 Differences between MDA and MDM

A conjunctive preservation of program code (machine code) and models (extended source files) has several advantages:

- faster access to essential knowledge
- modification of both artifacts (code and models) during maintenance
- easier consistency control after system modification

Knowledge retrieved from essential models, which in our approach presents one of the application's elements, allows us to go cyclically through the phases of life cycle during the maintenance process without the need of frequent project database browsing.

4.3. Model-Driven Maintenance Life Cycle

A life cycle of MDM starts with the operation of software system. As system is used, requirements for correction of its errors or requirements (user defined or as a consequence of environment change) for its changes are detected. According to the first Lehman law of the software evolution [9]: "A program that is used must be continually adapted, else it becomes progressively less satisfactory". The last phase of MDM life cycle consists of modification of the system itself (Fig. 5). Consequently, it is possible to assume that required modification is an inescapable consequence of the nature of software and the changing environment in which it's used.

As the first step, the requirements need to be well specified as it is very important to avoid the misunderstandings between users of the system and maintenance programmer. In here, the active user participation presents a very important aspect. Within this first step (i.e. requirements specification), the models represent a very natural and useful way to avoid any potential misunderstandings, although unfortunately most users don't understand the complex diagrams (for example in SysML notation) preferred by many traditional modelers. The solution might be the adoption of inclusive models which use simple tools and simple techniques that users can easily learn and therefore use to help capture and analyze requirements for certain system [2].

When the requirements are well specified by means of the inclusive models, maintenance programmer can modify essential models which are part of the application. The extensive use of CASE systems for visualization and applying of changes may be considered a very suitable choice in this phase as the changes of models can be processed without modification of operating software system. In this way, the maintenance programmer is able to discover impacts of required changes before they are really implemented into the system's code. The dependencies among components of system models and among models themselves, present knowledge which allow discovering the chain reaction of all required changes as a consequence of the modifications requested by user. It is particularly useful because if a programmer is aware of all required changes he can implement them all in one single step, without impacts to unchanged parts of the system. In the end, when all required modifications are implemented into the code and into the models of application, the consistency control needs to be performed. This control basically consists in a simple principle - if all changes within the models were processed also within the code, it's possible to deduce that they both describe the same system – so they are in consistent state.

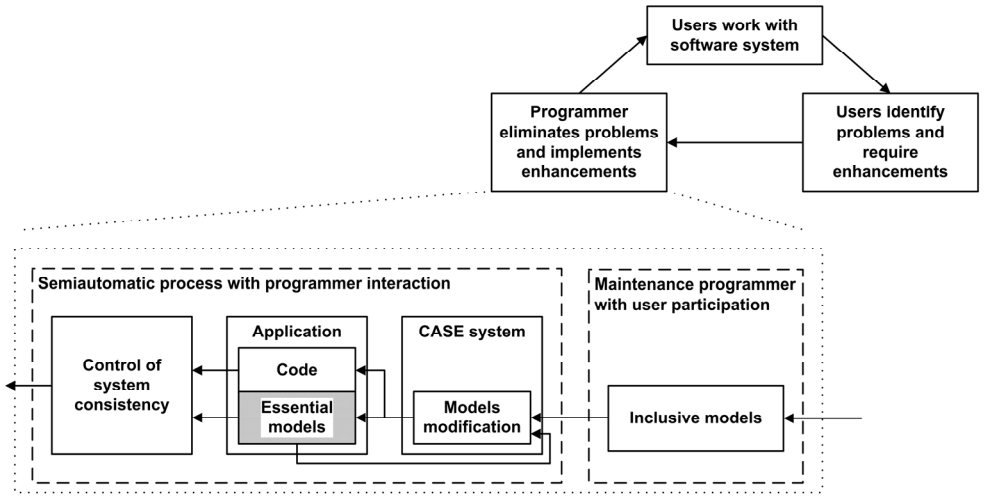

Fig. 5 Model-driven maintenance life cycle

5. CONCLUSIONS

This paper presents the System Development Unified Process, which supports the concepts of Model-Driven Development and Model-Driven Maintenance using the advantages provided by SysML. This approach based on the conjunctive preservation of program code and models, supports consistency between the essential models and code, as no change in code can be processed without accordant modification of system's models.

ACKNOWLEDGMENTS

This work was supported by VEGA Grant No. 1/0350/08 Knowledge-Based Software Life Cycle and Architectures.

REFERENCES

[1] ALLEN C., Software maintenance – an overview, *British Computer Society, Programming & Software Articles*, en-GB 3rd, February 2006.

[2] AMBLER, S. W. - JEFFRIES, R.: Agile Modeling: Effective Practices for Extreme Programming and the Unified Process. John Wiley & Sons, New York, 2002.

[3] BJORKANDER, M.: UML and SysML Software Modeling & Requirements Management, 2007, http://www.modprod.liu.se/Talk8-Bjorkander-UML-SysML.pdf

[4] CANFORA, G. - CIMITILE, A.: Software Maintenance. Handbook of Software Engineering and Knowledge Engineering, volume 1. World Scientific, 2001, ISBN: 981-02-4973-X.

[5] EmbeddedPlus SysML Toolkit tutorial, 2008, http://www.embeddedplus.com/sysml.php

[6] HAVLICE, Z. et al.: Knowledge-based software life cycle and architectures. *Computer Science and Technology Research Survey*. Kosice, elfa, 2007, ISBN 978-80-8086-071-4.

[7] JOHNSON, T. - PAREDIS, Ch.: Integrating Models and Simulations of Continuous Dynamics into SysML, 2008, http://www.omgsysml.org/

[8] KLEPPE, A. - WARMER, J. - BAST, W.: MDA Explained: The Model Driven Architecture™: Practice and Promise, 192 pp, Addison Wesley, 2003, ISBN 0-321-19442-X.

[9] LEHMAN M. M.: Lifecycles and the Laws of Software Evolution, *Proceedings of the IEEE, Special Issue on Software Engineering*, 19:1060-1076, 1980.

[10] NEWCOMB, P.: Architecture Driven Modernization, *Proceedings of the 12th Working Conference on RE. IEEE Computer Society*, 2005, ISBN 0-7695-2474-5.

[11] Object Management Group, http://www.omg.org

[12] OMG Model Driven Architecture Specification, 2007, http://www.omg.org/mda

[13] OMG Systems Modeling Language (OMG SysML™) v 1.0, OMG Available Specification, 2007, http://www.omgsysml.org/

[14] WEILKIENS, T.: Systems engineering with SysML/UML: modeling, analysis, design. 322 pp, Morgan Kaufmann Publishers, 2007, ISBN: 978-0-12-374274-2.

[15] YANG, H. – WARD, M.: Successful evolution of software systems. 300 pp, Artech House Publishers, 2003, ISBN 1580533493.

Received Jun 9, 2009, accepted August 18, 2009

BIOGRAPHIES

Ján Kunštár was born in Brezno, Slovakia, in 1982. He received the Ing. (MSc) degree in informatics from the Faculty of Electrical Engineering and Informatics, Technical university of Košice, Slovakia, in 2006. He is

currently working on his PhD. degree at the Department of Computers and Informatics FEEI, Technical university of Košice. His scientific research is focusing on enhancement of software maintenance process through the modification of system's architecture and by taking advantage of knowledge acquired from software systems' abstract models.

Iveta Adamuščínová was born in 1983. In 2007 she graduated (MSc.) with honors at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. Currently, she is a PhD. student at the same department. Her scientific research is mainly focused on knowledge-based systems, integration of knowledge into software systems' architectures and model-driven software development.

Zdeněk Havlice was born on 14. 02.1958. In 1982 he graduated (MSc.) with honors at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. He defended his PhD. in the field of visual programming and user interface design in 1991; his thesis title was: "Design of User Interface for Dialogue Systems". Since 1999 he is working as an associated professor at the Department of Computers and Informatics. His scientific research is focusing on the area of special languages, compilers, CASE systems, software methodologies, methods and tools.