STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 1, 2012

TOWARDS MDE IMPROVEMENTS FROM INTEGRATED FORMAL VERIFICATIONS

ANNA MEDVE

Abstract. This paper presents a methodology to improve the scenario-based modelling process from model checking and static analyzers during behavioural modelling in MDE process. Our method consists on combining modelling techniques with formal verification techniques to obtain an incremental iterative analysis process in an earlier phase of modelling. This helps not to fix architectural decisions earlier, and to guard and verify some choice for variability. These are based on generic properties of scenario modelling languages and verification tools, presented in the paper.This methodology contributes for future works, to obtain modelling increments from a separate verification engineering service process based in our previous results.

1. Introduction and Motivation

The paper presents a methodology to improve scenarios models by evaluating results of outputs and feedbacks from model checking and static analysis performed with incremental verification process.

As Bezivin stated in 2008 at Dagstuhl seminar [1], for the industrialization of software engineering the individual verification technology (such as model checking, static analysis, or theorem proving) was insufficient and that integrated tool chains and workbenches for model-driven engineering (MDE) were needed. We claim that individual verification workbenches can be very useful for supporting architectural decisions in earlier phases, when the abstraction level is high and deadlock may occurs from incomplete domain attributes. In this later case static analysis combined with model checking can raise the level of genericity in architectural modelling.

Received by the editors: December 10, 2011.

1998 *CR Categories and Descriptors.* D.2 [**SOFTWARE ENGINEERING**]: D.2 Software/Program Verification – *Model checking*; I.6 [**SIMULATION AND MODELING**]: I.6.5 Model Development – *Modeling methodologies*.

Key words and phrases. MDE process, model improvements by verification, scenario-based modelling, formal verification.

We belief that a systematic modelling process for integrating incremental verification should provide model improvements in earlier phases of MDE. For building an MDE process and integrating them with formal verifiers it is necesary to choose verification workbenches and to perform various verification types, or find verification engineering services. As a direct continuation of previous works [2, 15, 16], which introduced verification engineer roles and tasks using Verimag IFx-Omega toolset [12], this work proposes to obtain model improvements based on verification tasks and services.

In this paper we make the following novel contributions towards MDE improvements presented in Section 3 and 4:

(1) We introduce a methodology for integrating incremental formal verification in modelling process. We formulate steps for static analysis and model checking-based scenario validation (in Section 4).
(2) We show concepts and language features from formal scenario-level approach which can be effective to define increments for checking architectural decisions and build a scalable communication architecture (Section 3).
(3) We highlight the effectiveness of our proposition based on research results and trends in scenario-based design.

The rest of paper is structured as follows: Section 2 contains the theoretical background. Section 3 and 4 contains the proposed contribution. Section 5 presents related works and discussion. The closing section contains the few conclusions and future work.

2. Background

2.1. **Model-based Development and Model-driven Engineering.** Model-based development (MDD) is a scalable process which is built on the global model of software and is made up of heterogeneous components [18], which provide several advantages within software paradigms:

- it has techniques on a high level of abstraction;
- it places the model and the consecutive model transformations in the center of the development process;
- it replaces trials with validation and replaces real prototypes with virtual prototypes;
- it has a strong relation among the methods, tools and model management activities during the life cycle;
- it has the basis of support and guidance of design and validation and the basis of the derived low-level descriptions close to implementation.

The advantages and the heterogeneity to be attained require the integration of the means of validation by methodological support in order to increase efficiency, robustness and flexibility in heterogeneous systems.

The Model-Driven Engineering (MDE) standard of OMG specifies that the model represents the system in an expert's observation and the model conforms to its meta model [18]. Regarding the MDE paradigm and our model refinements aims, we supplement the notion of an MDE model and process with conceptual and technical approaches of a given domain.

In MDE process the primary document is the Architecture Model, which is not mandatory Model-Driven Architecture (MDA) if the rapid application development (RAD) is used. The documents of an MDE process are the Requirements Model, Architecture Model and the Implementation Model. The abstraction of the problem lies in the Requirements Model, which we develop into the Implementation Model with the help of systematic model transformations. The application is created from the Implementation Model by code-generation.

The Architecture Model has a central role because it converges with the requirements of the traditional development. This means to describe the architectural decisions in an explicit way in the requirement cluster of the architectural characteristics. The model relationships and roles in MDE have highlighted the architecturally significant requirements, which involve to place the domain-specific knowledges at the basis of architectural decisions.

2.2. **Formal Verification Methods.** Formal verification is a broad term for model checking, static analysis and model based testing of development artifacts formulated with formal methods.

An overview of large family of model checkers is presented at formal method Wiki portal [9, 19].

Model checking is an automated technique that given a finite state model of a system and a property stated in some appropriate logical formalism systematically checks the validity of this property. Model checking is a general approach. Case studies have shown that the incorporation of model checking in the design process does not delay this process more than using simulation and testing and for several case studies the use of model checking has led to shorter development times[6, 10].

Model checking phase consist for checking and reporting. The verification tools operate as a simulator, following one possible execution path through the system and presenting at the model checker GUI the resulting execution trace or a counterexample if faults occur. The methods can be applied to all or only the most critical portions of the system usefully for detecting both basic and other type of errors. The most important is that the procedure is

completely automatic. Typically, the user provides a high level representation of the model and the specification to be checked. The model checking algorithm will either terminate with the answer true, indicating that the model satisfies the specification, or gives a counterexample execution that shows why the formula is not satisfied. The counterexamples are particularly important in finding subtle errors in complex transition systems.

Static analysis

Static code analysis is a general term for a set of techniques used to aid in the verification of computer software without actually executing the programs. The analysis varies greatly depending on the tool employed and analysis guidelines applied in tools [9]. Static analysis is a powerful concept and can significantly aid in development of higher quality software. Specific static analysis tools are style-checking tools, semantic analysis tools, deep-flow static analysis tools, which extend compilation and abstract interpretation capabilities with generic and specific guidelines for checking. A static analysis tool runs automatically and reports all defects it identifies, some of which may be insignificant and require little or no work to correct, whilst others could be critical and need urgent correction. These defects therefore require strong management to ensure that the full benefit is obtained from using the tool in the first place.

2.3. **Attributes for the Selection of Model Checking Tools.** Clarke et al. in [6] argue on orientation toward error detection, as having methods and tools which should support generating counterexamples as a means of debugging and finding errors, rather than for certifying correctness.

We investigated the most important requirements for a verification toolset. A model checking tool contains several input translation engines, the internal or external computation engine for model checking execution, tools to extract the results as outputs and feedback to the input . We enumerate the *desired set of properties for a verification toolset* [20],[10]:

(1) it supports heterogeneity and it integrates model checking with other verification and validation techniques;
(2) it offers combination of features for modelling and validation (i.e. it provides language level access to descriptions and it implements static analysis and optimization techniques);
(3) it is open to modelling languages and validation tools;
(4) it has mechanisms for restricting non-determinism and controlling execution;
(5) it adopts asynchronous execution paradigm to be apt to validate separation of concerns and non-atomic interactions
(6) possibility of QoS predictability;

(7) and last but not least, it is open for all kinds of already implemented components with an adequate interface.

The IFx toolset [12], which we applied in previous work [15, 16] is automaton based and satisfy the above enumerated requirements.

3. Extending MDE Processes by Integrating Incremental Verification

Unlike a complete integrated tool chains and workbenches for MDE, we follow to integrate tools by MDE process which is not intended as a commercial entity. These verification tools are sets of commonly agreed interfaces and operating methods, that allow specific tools to interoperate with other tools forming a complete working environment. This avoids tool dependencies of model-based software applications mainly in earlier phases were multi-language combination became widely used during platform independent modelling.

MDE processes operates dynamically in earlier phases for obtaining the Architecture Model, which is the primary document of a process. The dynamics has strong relationships with requirements model by modelling architecturally significant requirements and architectural decisions. The modelling process involving formal verification is illustrated in Figure 1. The roles for system modelling and for model checking can be performed separately by experts in the fields. The export or import capabilities of modelling tools support safety and efficiency of model transformations needed between tools. This is expressed in XMI/XML standard supporting terms.

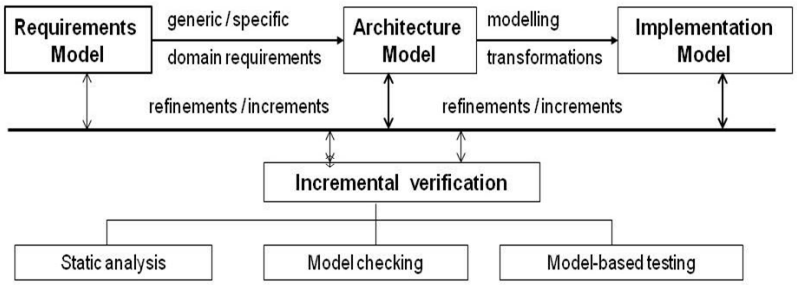

Figure 1. *Main MDE process and documents integrating formal verifications*

Formal verification phase consists of configuring, checking and reporting. The verification tools operate as a simulator, following one possible execution path through the model increment resulting an execution trace and/or a counterexample if faults occur which allows reasoning about model transformations

and to follow incremental verification based on engineering capabilities. Depending on reports content and abstraction level of modelling, combination of automaton based and temporal logic based verification tools may be more fruitful for incremental verification.

Management tasks for integrating incremental verification must involve verification engineering roles and resources, as is reported in [2, 15]. The verification engineering tasks goals are to define and control verification increments in order to discover and eliminate discrepancies between the modelling features. The main process elements are from configuring verification tools for various validation choices; analysis of counterexamples and model checking reports.

3.1. **Explanation of Modelling Process with Integrated Incremental Verification.** We introduce a small example quite complex for showing how a formal scenarios-level approach can be effective to define increments for checking architectural decisions and to build scalable communication architecture. In [14] we introduced the CompConfigur pattern-based method and an example for generic behaviour modelling for the domains which has client-server communication model. This method gives generic domain model in form of a set of class, architecture and scenarios diagrams.

We recall from [14] the example on GMSC (Gateway Mobile Switching Center) call management and we improve it with not covered services. Figure 2 illustrates the model of a simplified GSM network context. The behaviour of a GMSC call management process is restricted to initiate calls received from a local or a remote mobile station (MS, i.e. telephone, intelligent network element), to hande the occurred errors for ensuring the correct performance of connection, and to close the calls.

The static model of GMSC is obtained with CompConfigur architecture pattern language [14] as it shows in Figure 3 and 4. In the case, when the generic architecture pattern not covers every roles and events from a specific domain, it is need to be completed on architectural level for checking their consistency further during the analysis process of detailed design. This case it occurs in recalled example from [14] and it is discussed in this paper, i.e. the architecture model obtained in a way from architecture pattern language is improved by scenario modelling, and model checking.

In the next, by using this example we explain the incremental nature of scenarios and we show the use of scenarios language features, which engage modelling of scenario fragments.

Let the function location of mobile stations that need to be introduced in model. The location service is provided from Basic Station Controllers(BSC)

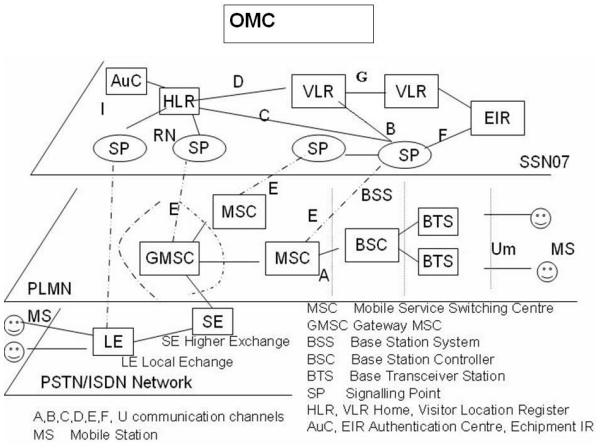

FIGURE 2. *The context of a simplified GSM network*

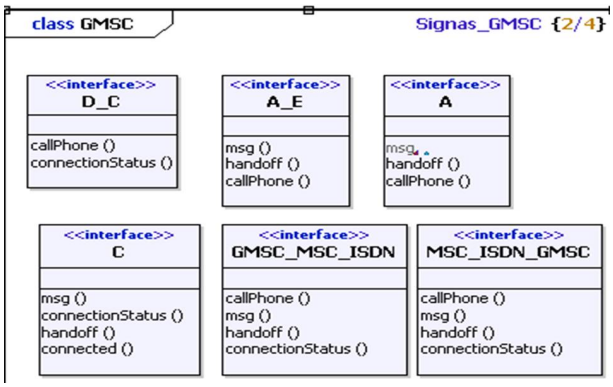

FIGURE 3. *The GMSC class model obtained from applying architectural pattern language*

for Mobile Stations. For this network elements their static model can be obtained from Wrapper Facade architectural pattern. Location service is activated by switching center for user demand. Figure 5 and 6 show two different modelling technique-based scenario models for Mobile Station Location Center (MSLC).

Figure 5 illustrates a bird eye view of the MSL scenario model of the MS location procedure. It serves to follow the message flow to read and find events of MSL function. The composition is difficultly to read, reuse or modify and not supports architectural modelling.

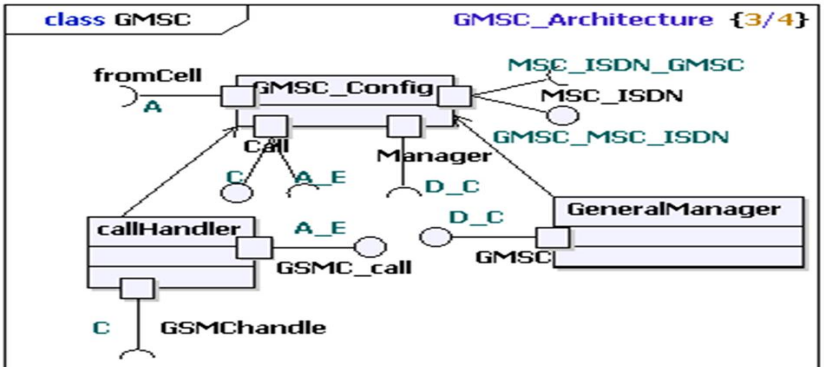

FIGURE 4. *The GMSC architecture model obtained from applying architectural pattern language*

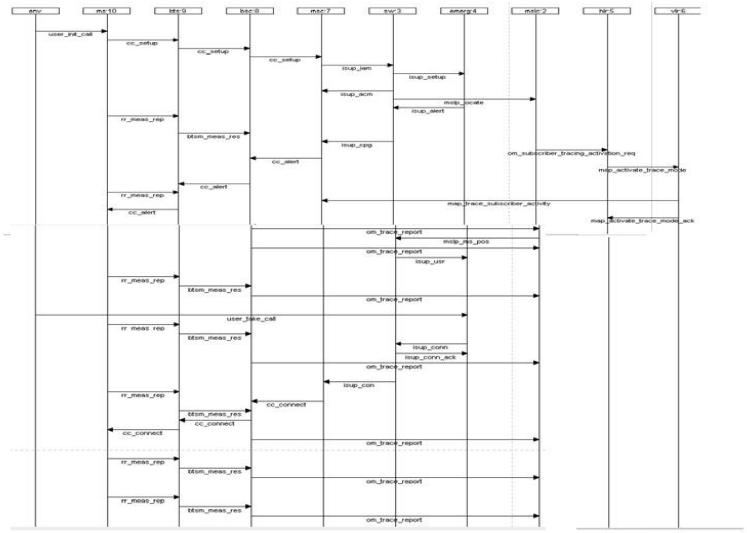

FIGURE 5. *The MSL scenario model in cascade style of message flows.GSM actors from left to rights: ENV, MS, BTS, BSC, SW, EMERG, MSL, HRL,VRL*

Figure 6 shows the MSL scenario model built on referenced subscenarios and messages that prepare the *call* of subscenarios. Referenced scenarios act as remote calls from discovered functionality or event. These form an entity for checking or structuring architectural levels.

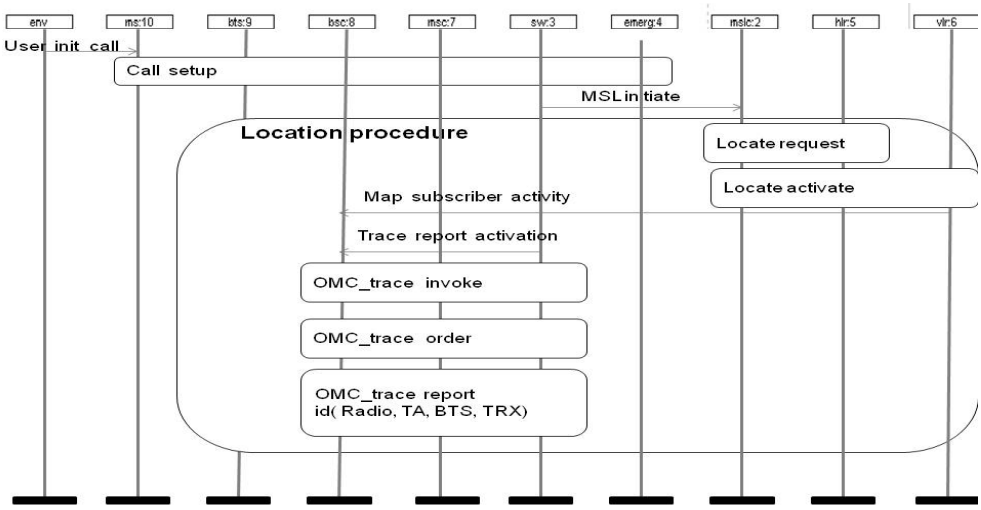

FIGURE 6. *The MSL scenario model in incremental modeling style of scenario fragments and their triggering messages. GSM actors from left to rights: ENV, MS, BTS, BSC, SW, EMERG, MSL, HRL,VRL*

Scenarios fragments allows to follow systematic iterations for incrementing the models and checking for deadline and inconsistencies.

Largely implemented scenario language properties which have results in verification increments are those inter-scenario constraints for searching of pair wise roles in scenarios guided from: scenario reduction from the Inactive Entity; Collaboration-based reduction by reference of one interaction to other interactions within the same (or other) collaboration (remote call); Role decomposition in an interaction into an interaction of its component parts such as *create*; Data composition from observations; - Control- based separations such as *call, kill*; Inline operators for scenario relationships such as *alt, opt, loop, par, corregion, exp, comp, decomp*, as Message Sequence Charts (MSC)[13] language features as well.

Performing model improvements addressing domain properties, such as QoS characteristics, intermediate state transitions to handle error states with fault tolerance for locally corrected, globally nothing bad events, modelling as negative and positive scenarios.

First we perform a scenario-based modelling process to obtain a requirements model from the user views. Next, we augment it with generic and specific domain requirements to obtain an earlier architecture model. This results in

state-based implementation models by integrating in process iteration steps of model checking and model refinements.

The above listed process starts with the modelling interactions between environment and system parts, generic in their nature for the domain. At this modelling phase the major part of the system is abstracted into communication subsystem which assures the safety and performance of the system. To avoid earlier architectural decisions we perform requirements engineering based on generic domain properties and architecturally significant requirements. We can obtain it from architectural pattern language. It results in interaction diagrams for the architecture model.

We introduce an iteration of incremental formal verification. For this an input is needed in the verification workbench. The architecture model resulted from scenario modelling gives the input for verification. Depending on system modelling tools support for XMI transformations we perform a translation process or we translate manually the scenario-based model from earlier architecture modelling.

If the choosed verifier has properties discussed in subsection 2.3 then the input for verifier will be obtained from an XMI or XML translation of the architecture model.

To perform verification engineering an expert modeler or a verification engineering service is needed in process.

In the case of an automaton based model verifier the scenario translation will give the input. In the case of a temporal logic based model verifier the annotated scenario will give a formal basis to construct the input model for the verifier and the input specification as temporal logic expressions.

In the case of manual translation of scenario model into automata's model, which consist of scenario-based model transcription based on conditional annotations for each input signal of a scenario in order to form conditions which behave as states of the emitter-receiver actors in a scenario communication. The receiver will perform a state transition conform to the scenario. It consists of performing one or more actions, decisions, emitting one or more output signals and going in the next state or remaining in the same state.

Some configuration is specified for the static analysis process in order to obtain model transformations or counterexamples. The verification possibilities varies from the abstraction level of the system under verification and the modalities in static analysis followed by deadlock and live lock checking. These are presented in the following section in the form of a set of steps in the verification process and of refinements of the initial system model.

3.2. **Verification Increments for Model Improvements.** *Model improvements based on verification increment:*

The modelling and checking of safety and liveness properties are essentials for absence of deadlock. The deadlock is the general property, where the program stop and makes no further progress.A safety property asserts that nothing bad happens. A liveness property asserts that something good eventually happens. The verification of these properties in earlier modelling phases using small portions of the usage models to obtain increments will improves the quality of architectural modelling.

Static analysis services from verification tools give feedback on unreachable states, unusable variables, or dead code sequences. These result in a refactored model that supports the analysis process in creation of model improvements. The visualization of removed unreachable states and transitions supports observation-control based analysis for fault localization in the design model.

Verification of liveness properties provide useful informations for performing analysis related on architectural decisions, i.e. procedural or functional aspect of the unused variable make explicit the designer biases on functional orthogonality of distributed tasks and variability options resulting in dependency and not of a scalable architecture.

Synthesized model improvements could be obtained from a simultaneously analysis of verification results and corresponding scenario language features localized in scenario model as is indicated in Section 3.1.

In order to obtain granularity of models and to make decisions if the model is complete we observe the results of static analysis for dead variable, i.e the removed variables and signal parameters with their dependency. Setting variations in the configuration file is the way for obtaining the provisioned test by eliminating subsets of functionality.

In a previous experiment in formal verification [15] with a simplified vending machine system we observed significant state space reduction from combination of static analysis tools services before model checking for automated deadlock check.

4. Methodology for integrating incremental formal verification in modelling process

Strategies for configuring verification engineering resources:

First at all, it needs to fix a set of strategy depending on verification engineering goals, services and capabilities, which are considered in the configuration of the behaviour of static analyzer and model checker used:

- for using the automatic configuration in order to not take into account temporal constraints,

- for using a scheduling model in order to describe certain temporal constraints,
- for reducing possibility of state explosion in order to detect possible deadlocks,
- for obtaining the test of the model, which corresponds to an advanced model checking process by configuring the tool features for verifying the model behaviour with the help of temporal logics and other abstractions given in tools,
- for carrying out model increments verification by configuring the static analyzer tool with the behaviour attributes and predefined guidelines from standards and references.

Steps for integrating incremental verification in scenario modelling:

- **Step1: Establish scenarios models as the input to verification.** Create/improve the high-level scenario-based model in MSC or UML2.0 SD. [13]. Export and convert it for input for a given static analysis tool.
- **Step2-2a: Perform static analysis.** Carry all basic features of static analysis (*Live, Reach, Slicing* (the order counts), and *Predefined Guidelines*) and their variations depending on simulation capabilities of the used tool. Iteration can be build from modes of Step8-2b and Step9-2c.
- **Step3: Analyze the reports from verification tool.** Perform an observation-based control to obtain verification increments, which make feedback directly on analyzer configuration and on requirements model.
- **Step4: Return to improve the scenarios.** Decompose static analysis feedback and handle it in order to quantify quality requirements (time, domain, resource), to reduce the number of scenario imprecision; to produce a model consistent domain knowledge and requirements goals; to produce additional information required for development.
- **Step5: Control inter-scenario constraints.** Define and control constraints on interactions for synthesized increments in order to eliminate discrepancies arising in inter-model communication. Inactive entities give support to revision of architectural decisions on task distribution.
- **Step6: Synthesize increments.** The exploration of results from various configuration modes of static analyzer gives verification increments, which improve modelling features as scenario, as use case, as quality requirements, as fault-tolerant scenario category, as input to the next verification act and transformation.

- **Step7: Feedback to new configurations.** Configure configure the verification tool features with abstract descriptions related to time, domain or resource quantifications.
- **Step8-2b: Perform model checking for deadlock check.** Carry out basic features of deadlock checking from verification tool. Iterations will results from combination of Step2-2a and Step9-2c. Follow Step3.
- **Step9-2c: Perform model checking for model-based testing.** Carry all basic features of static analysis (*Live, Reach, Slicing* (the order counts), and *Predefined Guidelines*) and their variations depending on simulation capabilities of the used tool. Iterations will result from combination of Step2-2a and Step8-2b. Follow Step3.
- **Step10-2d: Perform testing.** CADP, SPIN, for carrying out the testing of model, which corresponds to an advanced model checking process by configuring the tool features for verifying the model behaviour with the help of temporal logics and other abstractions given in tools,

Applying those steps the user can observe and validate the specifications in order to be able to decide whether the specification has hiatuses or the model is faulty. This gives feedback for simple safety features which may occur at every run. These may be basic findings starting from: deadlocks, message losses and time settings, or they can be much more specific depending on the domain attributes.

5. Discussion and Related Work

Recent real-world industrial case study on the modelling and validation has stated in Bozga et al. [3] that for such large examples, push-button verification is not sufficient and some iterative combination of analysis and validation is necessary to cope with complexity. They state principles of a verification methodology with three views : systemic view, requirements classification, observation-control based separation of dates.

In [7] Damas et al. present how negative and positive scenarios are the useful tools in requirements engineering for resulting in scenario-based modelling with formal verification which integrate goal, scenario, and state machine models where portions of one model are synthesized from portions of the other models. Our previous works in goal-oriented requirements engineering take account on this win-win engineering framework. These give consistency from iterated goal and improve earlier architecturally decisions.

Werner-Stark et al. in [21] apply systems' theory to obtain granularity and to control the domain quantifiers with qualitative transformations by building

intelligent diagnosis methods for the cases of transient operating conditions, e.g. when it is controlled by an operating procedures.

The Omega UML profile [12, 17] uses IFx to provide a new profile for real-time and embedded systems modelling and verification which extends the expressivity of the currently existing tools. Omega are applied for UML models in other recent works to handle models and model refinements [11]. In future work we apply them to preserve consistency in model improvement steps and to handle the non-determinism of time progress with incremental verification. Visser et al in [22] introduce an iterative technique in which model checking and static analysis are combined to verify large software systems. In their framework the role of the static analysis is to compute partial order information which the model checker uses to reduce the state space, but it not put the focus on earlier architectural decisions and model based verification.

In [5] Frappier et al. describe, utilize at the same example and compare the results from widely used six model checking tools for deciding which of them is the better for the validation of IS specification in model-driven engineering. They do not find generic solutions and not put the focus on verification tools for searching and deciding on the needed verifier capabilities.Their conclusion hold on our hypothesis on the need for MDE process engineering in each development case for optimizing the process among generic and specific domain properties, and existing tools and methods.

Bucchiarone et al in [4] apply in whole process model checking tools as automaton based and temporal logics based, as well for verify and test during entire process of component based web application development. Their method gives generic aspects for formal verification framework building methods. Hannousse et al. in [8] give a generic method for applying static analysis for cross-cutting aspect verification during the composition process.

6. Conclusions and Further Work

We introduced some results of a general approach on integration of formal verification into a scenario-based MDE methodology. Namely, a process and techniques based on static analysis and model checking and scenario-based processes.

In more detail, during the design process one produces a systematically granulated set of scenarios which are used for defining the state-based model of the target system. This model can be then subject to various formal verification tools appropriate for Model Driven Engineering (MDE) (model checking, petri nets, etc.).

Our method for model improvements with incremental verification lies both to formal verification integrated into the tool chains and the workbenches used

by a design methodology, and both to individual verification toolsets. The essence is to draw hope from available verification tools and advances in verification techniques to integrate verification into model-based or model-driven processes.

As future work we plan to apply research results from [7, 17] to work out steps for consistency preservation. Our approach gives rise to a novel conceptual framework for scenario-based model-driven requirements engineering and earlier validation involving model checking. Our results are far from being complete; further analyses and classifications are required based on action research for verification engineering services and tools.

7. Acknowledgements

The authors wish to thank László Kozma who contributed with ideas and help throughout this work.

We thank the anonymous referees for their valuable comments..

This research work was supported by TÁMOP-4.2.1/B-09/1/KMR-2010-0003.

References

[1] J. Bézivin, R.F. Paige, A. Uwe, B. Rumpe, D. Schmidt, : Model Engineering for Complex Systems, Perspectives Workshop: Model Engineering of Complex Systems (MECS), Dagstuhl Seminar Proceedings 08331, http://drops.dagstuhl.de/opus/volltexte/2008/1603.

[2] Zs. Borsi, L. Kozma, A. Medve, One Verification Problems of the Component-based Software Development, 8th International Conference on Applied Informatics, ICAI'2010, Eger, Hungary, 2010, Vol. 2. pp. 391-399.

[3] M. Bozga, S. Graf, L. Mounier, Il. Ober, Iu. Ober, J. Sifakis, The IF toolset, Formal Methods for the Design of Real-Time Systems, LNCS 3185, 2004, pp.237-267, http://www-if.imag.fr/

[4] N. Bucchiarone, H. Muccini, P. Pellicione, P. Pierini, Model-Checking plus Testing: from Software Architecture Analysis to Code Testing, In Proc. Int. Workshop on Integration of Testing Methodologies, FORTE 2004. LNCS 3236 pp. 351-365.

[5] Marc Frappier, Benıt Fraikin, Romain Chossart, Raphael Chane-Yack-Fa, and Mohammed Ouenzar, Comparison of Model Checking Tools for Information Systems,(Eds.: J.S. Dong and H. Zhu , ICFEM 2010, LNCS 6447, pp. 581–596, 2010

[6] E. M. Clarke, E. A. Emerson, J. Sifakis, Model checking: algorithmic verification and debugging, Communications of the ACM Volume 52 , Issue 11, (2009), pp: 74-84.

[7] C.Damas, B.Lambeau, A.van Lamsweerde, Scenarios, Goals, and State Machines: a Win-Win Partnership for Model Synthesis, Proc. FSE'06: Intl. ACM Symposium on the Foundations of Software Engineering, Portland (OR), November 2006., pp. 197-207.

[8] A. Hannousse, R. Douence, G. Ardourel, Static analysis of aspect interaction and composition in component models, 10th ACM International Conference on Generative programming and component engineering, GPCE 2011, pp. 43-52.

[9] formalmethods.wikia.com.

[10] T. Hoare, J. Misra, Verified software: Theories, tools and experiments: Vision of a Grand Challenge project, LNCS 4171 Springer, 2005, pp 1-18.
[11] J.Hooman, H.Kugler, I.Ober, Y.Yushtein, Supporting UML-based Development of Embedded Systems by Formal Techniques, Int. J. of Software and Systems Modeling, Volume 7, Number 2, 2008, pp. 131-155.
[12] IFx-OMEGA Toolset: http://www.irit.fr/ifx/; http://www-if.imag.fr/ OMEGA European Project (IST-33522), http://www-omega.imag.fr/
[13] ITU-T, Message Sequence Charts (MSC), ITU-T Recommendation Z.120, Genova. 1999.
[14] Medve, A., Kozma, L., Ober, I. :General Modelling Approach Based on the Intensive Use of Architectural and Design Patterns, Ed. H. Weghom, P. Isaias, R. Vasiu, Int. Conf.on Applied Computing, IADIS 2010, Timisoara, Romania, October 10-16, 2010, pp. 251-256.
[15] A.Medve, Gy.Orbán, L.Kozma: Let's go verification engineering, 8th Int. Conference on Applied Informatics, Eger, Hungary, January 27-30, 2010, Vol. 2. pp. 417-427.
[16] A. Medve, L. Kozma, MDE process and model improvement using the Verimag IFx verification tools, 8th Joint Conf. on Mathematics and Computer Science, MaCS 2010, Komárno, Slovakia, July 14-17, 2010, Selected p. Published by NOVADAT Ltd. pp. 323-336.
[17] Il.Ober, S. Graf, Iu. Ober, Validating timed UML models by simulation and verification. International Journal of software Tools for Technology Transfer (STTT), Volume 8, Number 2, April, 2006. Springer Verlag, pp 128-145.
[18] OMG , http://www.omg.org/technology/documents/index.htm.
[19] SPIN website: http://spinroot.com/spin/whatispin.html.
[20] VSTTE: Second IFIP Working Conference on Verified Software: Theories, Tools, and Experiments, Oct 6–9, 2008, Toronto, Canada http://qpq.csl.sri.com/vsr/vstte-08.
[21] A.Werner-Stark, E. Németh, K. Hangos, Knowledge-Based Diagnosis of Process Systems Using Procedure HAZID Information, in Knowledge-Based and Intelligent Information and Engineering Systems, LNCS 6883, 2011, pp. 385-394.
[22] W. Visser, G. Brat, Combining Combining Static Analysis and Model Checking for Software Analysis,16th IEEE In. Conf. on Automated software engineering, ASE 2001, IEEE CS, Washington, DC, USA , pp.262-272.

Department of Electrical Engineering and Information Systems, Faculty of Information Technology, University of Pannonia; PhD Student, Faculty of Informatics, Eötvös Loránd University

E-mail address: medve@almos.uni-pannon.hu