A Domain Specific Modelling Language for Specifying and Visualizing Requirements

Niklas Mellegård, Miroslaw Staron

IT University of Gothenburg,
Chalmers Tekniska Högskola | Göteborgs Universitet
SE-412 96 Gothenburg, Sweden
{niklas.mellegard, miroslaw.staron}@ituniv.se

Requirements can cause substantial problems in large software projects if not handled correctly and efficiently. The problems of missing requirements or incorrect de-scoping of projects are virtually the most prominent ones. Combining graphical representation of requirements and organizing these requirements in several abstraction levels was identified as one of the potential solutions to such issues in our research project conducted with one of major automotive companies in Sweden. The objective of the research reported in this paper is to improve requirements engineering activities by using a graphical modelling language for managing requirements based on Requirement Abstraction Model (RAM). We evaluated our results via a pilot controlled experiment and the results show a statistically significant improvement in the time required to assess the impact of changes by 37% with the same accuracy.

1. INTRODUCTION

Model Driven Engineering [1] is an established software analysis and design paradigm, bringing software engineering even closer to other engineering disciplines [2,3]. Nevertheless, in order to achieve significant improvements after introducing models into development processes, domain-specific modelling notations should be used [4]. Here, the notion of the domain can either be a vertical domain (e.g. automotive or telecom) or a horizontal one (e.g. requirements engineering, architecture). In our work with the industrial partner (Volvo Car Corporation) we noticed that there is a need for improvement in the area of these horizontal domains. In particular to introduce (although not necessarily develop from scratch) a graphical notation for modelling requirements. Despite the numerous advantages of the existing modelling techniques which can be used for modelling requirements (e.g. UML [5], DSLs[1] [6], SysML [7]) engineers still struggle to efficiently link requirements to design models for the purpose of documentation, traceability, or later change impact assessment. In the automotive domain, the textual requirements are common as this domain often combines heterogeneous disciplines like hardware engineering, software engineering, mechanical engineering, each with different tools and techniques. These different tools and techniques usually result in using text as the common ground for communication between the teams. The complexity and volume of the requirement

[1] Domain-Specific Language

specifications in vehicle projects are often problematic for the understanding of specifications. The problems with understanding and incompleteness of the specification [8] might lead to quality problems with the final products or timeliness of car development projects (when the quality has to be improved before the release).

In this paper, we present a notation developed in order to evaluate whether using a graphical way of structuring requirements leads to improved quality of the design models during car development projects. The modelling language is based on the existing framework for structuring requirements – Requirement Abstraction Model (RAM) and is intended to fulfil the following requirements:

- The models should graphically visualize requirements at different abstraction levels and the relationships between them.
- The models should be fully integrated with the existing requirements engineering tools, e.g. RequisitePro [9] from IBM/Rational.
- The models should support both forward and reverse engineering – i.e. modelling the requirements and generating text specifications (forward) and vice-versa (reverse).
- By using the models, business analysts and developers should be able to assess the impact of requirement change in a shorter time, identify contradicting and missing requirements in a shorter time and thus increase the quality of the final software product.
- All requirements should be traceable to the design documents (e.g. UML or Simulink models) to support assessment of changes in the design (as an effect of optimizations) on the requirement specification.

These requirements resulted in developing a new modelling notation as a domain-specific modelling language gRAM. The initial evaluation of this language via a controlled experiment shows that such a notation fulfils the requirements for shortening the time required for assessing the impact of change.

This paper is structured as follows; Section 2 presents work related to this research. Section 3 briefly introduces the requirements specification format used in this evaluation. Section 4 reports the design and result of the evaluation and section 5 concludes the paper.

2. RELATED WORK

The work presented in this paper is part of our ongoing research outlined in [10] within the research project ASIS done in cooperation with Volvo Car Corporation [11]. One part of the project aims at improving the way requirements is specified, and in particular, the extent to which requirement specification can be reused with a minimum of effort. As part of this research, a model for the requirements specification process is developed with the intention of finding areas where Model-Driven Engineering (MDE) approaches may improve efficiency. The research in this

paper contributes to that research; by examining to what extent a graphical model of the requirements affect the process of assessing the impact of a change request, a common way of reusing a requirements specification, to a specified system.

Existing graphical modelling languages which include requirements modelling are the UML [5] (after Objectory [12]) and SysML [7]. In the latter, the requirements are specified as stereotyped objects linked to first-order design entities. The notion of 'use case' is used in UML to capture requirements, which makes them more structured than textual documents, although leaving the format of specifying and linking requirements open for interpretation for the modellers (which usually leads to problems). The Entity-Relationship diagrams were used historically to capture requirements for databases. The modelling notation presented in EAST-DSL [13] (after AUTOSAR [14]) also advocates modelling requirements using abstraction levels, although does not solve this problem in the current version of the modelling language.

Modelling of requirements has also been advocated in the context of MDE/MDA, e.g. in [15-17] and in particular in the context of executable UML [18-20]. In the executable UML the requirements are very closely linked with the conceptual models (domain models) which are used as the first steps in creating working software.

Studies to validate the effectiveness of the RAM approach have been done in e.g. [21-23] and in our paper we intend to extend these studies by investigating change impact assessment and using graphical DSL. In this paper, we evaluate whether adding a graphical representation for RAM-structured requirement specification can lead to further improvements. However, we also consider time as one of the factors, thus focusing on efficiency, not only effectiveness.

3. REQUIREMENTS SPECIFICATION FORMAT

In this section we present the graphical modelling language for structuring requirements and summarize the basis for it – the textual format of Requirement Abstraction Model.

3.1. REQUIREMENTS ABSTRACTION MODEL

The Requirement Abstraction Model (RAM) [21] has the goal of ensuring consistency and traceability among requirements in order to increase the overall quality of requirement specifications. The RAM defines a number of abstraction levels to which each requirement is classified and checklists to ensure that the requirements are assigned their proper level. In their original paper Gorschek and Wohlin [21] suggest, but do not limit their model to, four abstraction levels:

- *Product*: Product level requirements have a goal-like nature, very high-level descriptions of the desired functional and qualitative properties of the product.
- *Feature*: Feature-level requirements describe the features that fulfil the product level goals.

– *Function*: Function level requirements define which functions should be provided by the system in order to provide the features.
– *Component*: Component level requirements describe how something should be solved, i.e. bordering to design information.

RAM ensures traceability between requirements through all levels of abstraction by enforcing that, with the exception of the product level, no requirement may exist without a link to the more abstract requirement. The rationale is that no requirement may exist unless there is a clear and unambiguous reason for its existence motivated by higher-level requirements, and conversely, high-level requirements should be traceable to the lower-level requirements that satisfy them.

3.2. gRAM – DSL FOR MODELLING REQUIREMENTS

gRAM is a Model-Driven Engineering (MDE) approach to requirements specification with the purpose of creating an easy to use environment for direct on-screen manipulation of a requirements structure, from which other documents can be automatically generated. The gRAM is a formalized graphical Domain Specific Language[2] (DSL) complying with the RAM, where validation rules (i.e. static semantics) built into the gRAM ensures that the model and the resulting requirement specification are syntactically correct and well-formed according to the RAM. gRAM defines traceability links according to the RAM with its *Owns/Satisfies* link between requirements at adjacent abstraction levels, and adds the *Depends-on* traceability link, which indicates that there is a dependency between two requirements within the same abstraction level.

The definition of the gRAM follows the approach for language engineering as advocated by Evans et al. [24] and is divided into three parts:

– *Abstract syntax* – a meta-model complying with the RAM, defined using MS Visual Studio 2008 DSL Toolkit [25].
– *Concrete syntax* – a set of graphical shapes for elements. The concrete syntax is defined using the domain designer in MS Visual Studio.
– *Semantics* – The semantic parts of the gRAM are (i) *static semantics* (validation rules) in order to ensure that the diagram complies with the specifications of RAM and (ii) *translational semantics* for information interchange with other tools. The rules for transforming the requirements structure to a structured text document are part of the latter.

The above elements are defined in the following sub-sections.

Syntax

The main component in the abstract syntax, shown as a UML meta-model in Fig. 1, is the concept of abstraction level as defined in RAM. The top node in the abstract

[2] In this context the term “domain” is requirement engineering, not a vertical application domain like automotive or telecom

syntax is the notion of the model (*Model*), which contains abstraction levels (*Abstraction Level*). The abstraction levels contain requirements (*Requirement*). These two containment relationships are denoted by the name "*BelongsTo*".

Each requirement, with the exception of ones at *Product* level, must be linked to a requirement one abstraction level higher – denoted by relationship *Owns*/*OwnedBy*. Any given requirement can also depend on other requirements – as defined by the relationship *DependsOn*/*UsedBy*. Furthermore, requirements at the lowest level of abstraction (*Component* in our case) also have a link to the module in the logical design that implements the requirement.

This abstract syntax defines models that contain requirements grouped in abstraction levels. The constraint that each requirement must link to a requirement at a higher abstraction level is not expressed in the abstract syntax, but is expressed as static semantics – validation rules. This particular design choice is dictated by the need to be able to move requirements between abstraction levels during requirement engineering processes.

The concrete syntax is defined using a set of geometric shapes in MS DSL Toolkit Domain Designer. The concrete syntax is shown as an example gRAM requirements structure presented in the following subsection.

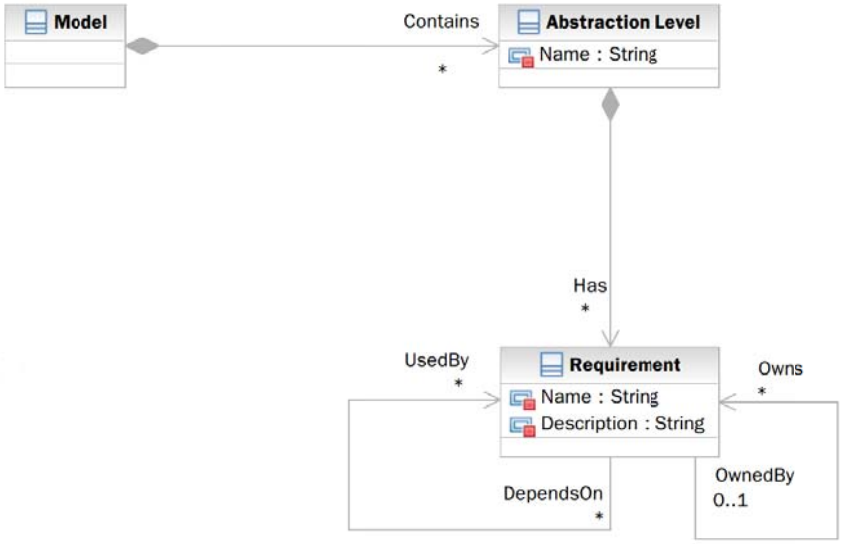

Fig. 1 The gRAM UML meta-model

Semantics

The semantic parts of the gRAM are (i) static semantics, in the form of validation rules in order to ensure that the diagram complies with the specifications of RAM and (ii) translational semantics for information interchange with other tools.

Static semantics

The static semantics are validation rules ensuring that the model complies with the RAM. The validation rules in gRAM is written in a general purpose programming language (C#) which allows for the creation of custom complex rule sets.

Although our proposed DSL as shown here is based on the four abstraction levels suggested in [21], the static semantics does not limit the number of levels.

Translational semantics

The translational semantics of the gRAM allows information exchange with other tools. The version of gRAM proposed in this paper, three such semantic sets were defined; ability to export the requirements structure to a format compatible with IBM/Rational RequisitePro [9] and two different styles of structured text in Microsoft Word format (used to generate the requirement specifications used in the evaluation as described in section 4). The translational semantics allow for manipulating the graphical requirements structure in gRAM, and then automatically generating documents in other formats, thus localizing changes to one artefact (the model).

3.3. EXAMPLE

The purpose of this example is to illustrate how a set of requirements can be designed using gRAM.

Let us consider a roadside speed surveillance camera, which should be able to identify and take a picture of a speeding vehicle, and from that image identify the vehicle model or type, the registration plate number and extract a cropped image of the drivers face. An excerpt of a simplified requirement structure is shown in Fig. 2.

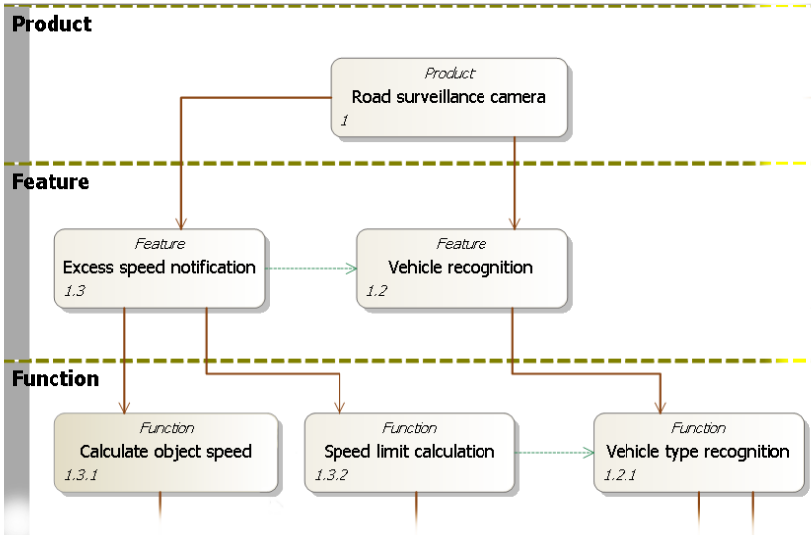

Fig. 2 An example gRAM requirements structure

In the requirements structure, the level a requirement belongs to is automatically assigned to the requirement depending on where it is placed. The tool allows requirements to be dragged and dropped into another abstraction level, with automatic integrity checks. An invalid model cannot be saved and the modeller is asked to correct the problems.

When a valid requirement structure is considered ready for use in an external tool, it is exported to the textual interchange format, which can be used for import into the desired tool. Fig. 3 and Fig. 4 show two versions of a requirement exported to Microsoft Word format (these versions were used in the evaluation reported in the following section).

1.3 Excess speed notification

Abstraction level	Feature
Description	From an moving image source, such as a radar or camera, the the product shall be able to identify and track moving objects in order to estimate their speed.
Satisfies	1
Depends on	1.2

Fig. 4 A generated gRAM requirement (full version)

1.3 Excess speed notification

Description	From an moving image source, such as a radar or camera, the the product shall be able to identify and track moving objects in order to estimate their speed.

Fig. 3 A generated gRAM requirement (short version)

4. EVALUATION

The purpose of this initial evaluation is to examine how the use of a graphical requirements specification model, such as gRAM, affects change impact assessment. The study examined whether there is any statistically significant impact on the efficiency, with respect to speed and accuracy, of assessing change impact on a specified system.

In the following subsections, the experiment design is briefly outlined and then the results are reported.

4.1. Experiment design

The evaluation of the gRAM was conducted through one initial and one replication experiment using the same basic instrumentation. Two groups of subjects were introduced to the requirements specification and logical design of a toy software system. One group was presented with the graphical requirements structure together with a short textual requirements specification, as shown in figure Fig. 2 and Fig. 3, while the other group with a the full textual requirements specification as shown in Fig. 4. Both of these requirements specifications were generated from the same

model to ensure consistency. The subjects were then asked to perform a number of change impact assessment tasks, common to both groups, using the provided material.

The following subsections details the experiment design.

Population and Sample

The population of this experiment is software designers working with implementation of software requirement specifications and systems analysts creating/maintaining these specifications.

In the initial experiment, the participants were 14 first and second year master students (i.e. in their 4th and 5th year of university studies) attending Software Engineering and Management programme and four 3rd year bachelor students (i.e. their 3rd year of education) from the same programme. Most of the participating master students had over one year of industrial experience prior to their studies.

In the replication experiment, 12 bachelor students, attending the first year of the Software Engineering and Management programme, participated.

Instrumentation

In each experiment, the test subjects were introduced to the context of a toy software system by the following scenario:

> *A vehicle manufacturing company has designed and implemented a simulator for a new type of drive-by-wire power steering system. The intention of the simulator is to provide a realistic environment for engineers to experiment with, e.g. different algorithms for solving certain tasks related to the power steering system. The design of the simulator has the goals of providing a realistic physical environment, a realistic software architecture and easy visualization of the vehicle; in particular, the parts related to steering.*

The simulator was implemented in Java prior to the experiment and the requirements were traced to the software components of the simulator (via the requirements at the lowest level of abstraction). Two versions of software requirements specifications were prepared – a textual and a gRAM-based one, both generated from the same gRAM model, by using the translational semantics built into gRAM to ensure consistency. The toy system was inspired by the real-world systems our partners work with, and which could not be used due to confidentiality and the complexity of the systems.

The experiment objects were: (i) written experiment instructions, (ii) a requirements specification, and (iii) logical view of the system. An introductory lecture was given to each group separately, with the only difference in contents being the requirements specification format; where one group was presented with the textual RAM format (an example requirement is shown in Fig. 4), while the other with the gRAM format (an excerpt of a graphical example requirements structure is shown in Fig. 2, and an example requirement is shown in Fig. 3). The written experiment instructions and requirements specification differed between the groups in the same

way. The requirements specification consisted of 56 requirements, of which 29 were at the lowest (component) level of abstraction.

The logical view, showing a high-level class diagram of the implemented power steering simulator, was identical for both groups, and consisted of 10 logical modules and 15 inter-module dependencies. The full set of experiment material is available at [26].

Measurements

The measurements collected for each task include the time taken, the score as a percentage of correct answers, number of false positives and the subjects' perceived confidence. The following variables were derived from the collected variables for each subject:

- *AVG_SCORE* The subject's average score over all tasks (%)
- *TOT_FP* The subject's total number of false positives for all tasks
- *TOT_TIME* The total amount of time the subject spent on the tasks
- *AVG_CONF* The average of the subject's confidence level over all tasks
- *EFF* The efficiency of the change impact assessment process, calculated as *AVG_SCORE* / *TOT_TIME*

The null hypotheses posed are that there *is no difference in mean* values in the derived variables between the two groups of subjects.

Validity Evaluation

The main threats to the validity of the study are external and conclusion validity, as described by Wohlin et al. [27].

External Validity

The main threat to the external validity is the use of student subjects, which may limit the ability to generalize the result to an industrial situation. The study was done mainly to evaluate the impact of the format of the requirements specification, and we do not make any conclusions about its applicability in an industrial situation yet. An industrial evaluation of gRAM is planned for the future in the same way as an industrial evaluation presented in [28].

Conclusion Validity

The statistical power of the conclusions is quite low due to the small sample size. This threat to validity limits the strength of the conclusions drawn from the study. Rather than stating firm conclusion, we limit ourselves to *indications* and *tendencies*.

Testing of the collected data showed that many of the variables did not fit to the normal distribution. Of this reason, non-parametric tests were chosen, which further decreases the statistical power of the result but does avoid the risk of violating assumptions and introducing further threats to the validity of the conclusions.

4.2. RESULTS

Descriptive statistics

Table 1 shows the descriptive statistics for the derived variables in the experiment. The results indicate (with statistical significance as shown in Table 1) that the gRAM group was 37% faster than the text group (TOT_TIME). Although not significant, the results also show that the text group scores 12% better (AVG_SCORE) and produce 22% less false positives (TOT_FP) than the gRAM group, indicating that the textual specification improves the accuracy of change impact assessment. A hypothesis for this is that the graphical notation quickly gives a sense of overview and understanding of the structure of the requirements, which in turn leads to the subject being confident more quickly and hence satisfied with the answer more quickly. This hypothesis was supported by post-experiment interviews with a sample of the test subjects. There is an indication of a difference in the perceived confidence of the answers (TOT_CONF) with slightly higher confidence in the gRAM group, further supporting this hypothesis.

Variable	Mean	Std. Dev.	Sign. (p)
AVG_SCORE_{TEXT}	49.07	12.16	No (0.424)
AVG_SCORE_{gRAM}	40.00	17.53	
TOT_FP_{TEXT}	17.75	11.25	No (0.351)
TOT_FP_{gRAM}	22.40	11.29	
TOT_TIME_{TEXT}	3113.00	253.49	Yes (0.002)
TOT_TIME_{gRAM}	1965.50	741.65	
TOT_CONF_{TEXT}	14.13	1.64	No (0.501)
TOT_CONF_{gRAM}	15.4	4.67	
EFF_{TEXT}	0.01572	0.0035	No (0.183)
EFF_{gRAM}	0.02265	0.01406	

Table 1 Descriptive statistics

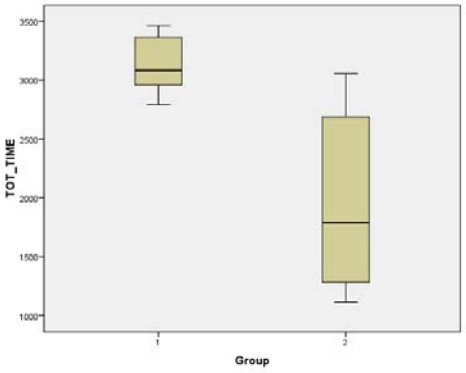

Fig. 6 Total amount of time spent (TOT_TIME)

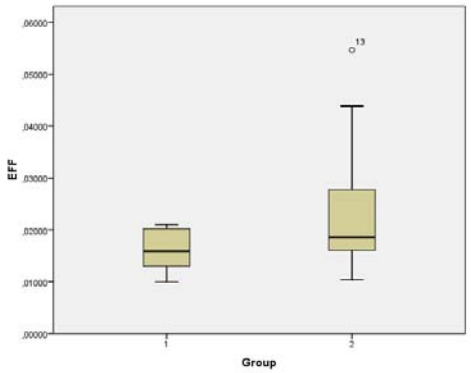

Fig. 5 Efficiency as defined by the EFF-variable

The efficiency (EFF), calculated as score over time, is 44% higher for the gRAM group indicating a great improvement in efficiency, considered as the number of correctly given answers per time unit. The difference in efficiency could however not be shown statistically significant. Fig. 5 and Fig. 6 show the boxplots of the time and efficiency variables, where group 1 was provided with the textual requirements specification and group 2 with the graphical version.

5. CONCLUSION

One of the more prominent problems with realizing the visions of model-driven development is limited traceability between requirements and design models [29]. This lack of traceability can lead to inefficient change impact assessment, a common activity when reusing requirement specifications. In this paper, we presented a Domain Specific Modelling Language for modelling requirements according to the principles of Requirements Abstraction Model (RAM), named gRAM.

Our preliminary conclusions from using the gRAM are that it makes it easier to develop a requirement specification, provides means for automatic verification of the specifications, and provides more flexibility in the process of creating the specification. The visualization of requirements gives a clear overview of the requirements structure and allows for direct manipulation and on-screen feedback of the implications of the manipulations (e.g. automatically changing the requirement properties when moving the requirement between abstraction levels), thus making graphical requirements modelling a more powerful approach than a textual-based one. Using gRAM enables automatic verification of requirements format and structure by validating static semantics as defined by the RAM. Preliminary observations that using gRAM results in more complete and less inconsistent requirement specifications were confirmed during the development of the experiment material reported in this paper, when several inconsistencies in the requirements specification were detected; the inconsistencies had not been spotted when inspecting the text format prior to that. During the development of experiment materials, the gRAM allowed for quick restructuring of requirements in order to try different approaches, allowing for flexibility that is not as easily achieved in a pure textual format.

The goal of information exchange with other established requirement management tools was partly accomplished by the using translational semantics in gRAM, which converts the gRAM requirements model into a textual format understandable by the desired tool (e.g. IBM/Rational Requisite Pro)

We performed an experiment conducted in two sessions to evaluate what effect the use of gRAM has on the efficiency of assessing the impact of a requested change. The conclusions from the evaluation shows that using the gRAM visual requirement structure improves the speed of assessing the impact of a requested change, in our case by 37%.

In our future research, as part of the evaluating this approach a study at a company is planned. The study is expected to discover whether this approach adequately addresses the challenge of producing complete and structurally sound requirements documents in industry.

ACKNOWLEDGMENTS

This research is partially sponsored by VINNOVA under the V-ICT program and the ASIS (Algorithms and Software for Improved Safety) project.

REFERENCES

[1] S. Kent, "Model Driven Engineering," *Integrated Formal Methods*, 2002, pp. 286-298.

[2] R. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap," *2007 Future of Software Engineering*, IEEE Computer Society, 2007, pp. 37-54.

[3] J. Ludewig, "Models in software engineering – an introduction," *Software and Systems Modeling*, vol. 2, Mar. 2003, pp. 5-14.

[4] Miroslaw Staron, "Transitioning from code-centric to model-driven industrial projects – empirical studies in industry and academia," *Model Driven Software Development: Integrating Quality Assurance*, Information Science Reference, 2008, pp. 236-262.

[5] *Object Management Group* [Internet], Available from 'http://www.omg.org/', *(Accessed 2008-09-22)*.

[6] J. Greenfield and K. Short, "Software factories: assembling applications with patterns, models, frameworks and tools," *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, Anaheim, CA, USA: ACM, 2003, pp. 16-27.

[7] *SysML - Open Source Specification Project* [Internet], Available from 'http://www.sysml.org/', *(Accessed 2008-09-22)*.

[8] J. Noppen, P. van den Broek, and M. Aksit, "Imperfect Requirements in Software Development," *Requirements Engineering: Foundation for Software Quality*, Springer Berlin / Heidelberg, 2007, pp. 247-261.

[9] *IBM/Rational RequisitePro - Software* [Internet], Available from 'http://www-01.ibm.com/software/awdtools/reqpro/', *(Accessed 2008-09-22)*.

[10] N. Mellegård and M. Staron , "Methodology for Requirements Engineering in Model-Based Projects for Reactive Automotive Software," Paphos, Cyprus: Springer--Verlag, 2008.

[11] *ASIS - Algorithms and Software for Improved Safety* [Internet], Available from 'http://www.ait.gu.se/english/research_groups/se_management/research_projects/ASIS_Active_Safety_Systems/', *(Accessed 2009-04-7)*.

[12] I. Jacobson, *Object Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley Professional, 1992.

[13] *ATESST Webpage* [Internet], Available from 'http://www.atesst.org', *(Accessed 2008-05-09)*.

[14] *AUTOSAR Webpage* [Internet], Available from 'http://www.autosar.org', *(Accessed 2008-05-09)*.

[15] M.D. Miguel, J. Jourdan, and S. Salicki, "Practical Experiences in the Application of MDA," *Proceedings of the 5th International Conference on The Unified Modeling Language*, Springer-Verlag, 2002, pp. 128-139.

[16] A. Wegmann and O. Preiss, "MDA in enterprise architecture? The living system theory to the rescue," *Enterprise Distributed Object Computing Conference, 2003. Proceedings. Seventh IEEE International*, 2003, pp. 2-13.

[17] T. Meservy and K. Fenstermacher, "Transforming software development: an MDA road map," *Computer*, vol. 38, 2005, pp. 52-58.

[18] S.J. Mellor and M. Balcer, *Executable UML: A foundation for model-driven architecture*, Addison Wesley, 2002.

[19] L. Starr, *Executable UML How to Build Class Models*, Prentice Hall PTR, 2001.

[20] S.J. Mellor, *Executable and Translatable UML* [Internet], Available from 'http://www.embedded.com/9900932', *(Accessed 2009-04-08)*.

[21] T. Gorschek and C. Wohlin, "Requirements abstraction model," *Requir. Eng.*, vol. 11, 2006, pp. 79-101.

[22] T. Gorschek, P. Garre, S. Larsson, and C. Wohlin, "Industry evaluation of the Requirements Abstraction Model," *Requirements Engineering*, vol. 12, Jul. 2007, pp. 163-190.

[23] N. Mohammad, Y. Vandewoude, Y. Berbers, and R. Feldt, "Suitability of Requirements Abstraction Model (RAM) Requirements for High-Level System Testing," *International Journal of Computer and Information Science and Engineering*, vol. 2.

[24] T. Clark, A. Evans, P. Sammut, and J. Willans, *Applied metamodelling: A foundation for language driven development*, 2004.

[25] Microsoft, Redmond WA, *Microsoft Visual Studio 2008 Software Development Kit* [Internet], Available from 'http://msdn.microsoft.com/en-us/library/bb166441.aspx', *(Accessed 2008-09-22)*.

[26] *gRAM Experiment Material* [Internet], Available from 'http://www.ituniv.se/~miroslaw/ram-dsl_experiment/', *(Accessed 2009-04-7)*.

[27] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslèn, *Experimentation in Software Engineering: An Introduction*, Boston MA: Kluwer Academic Publisher, 2000.

[28] M. Staron, L. Kuzniarz, and C. Wohlin, "Empirical assessment of using stereotypes to improve comprehension of UML models: A set of experiments," *Journal of Systems and Software*, vol. 79, 2006, p. 727-742.

[29] M. Staron, "Adopting MDD in Industry - A Case Study at Two Companies," *ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems*, O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, eds., Genova, Italy: Springer-Verlag, 2006, pp. 57--72.