PROJECT MONITORING AND CONTROL IN MODEL-DRIVEN AND COMPONENT-BASED DEVELOPMENT OF EMBEDDED SYSTEMS

The CARMA Principle and Preliminary Results

Rikard Land, Jan Carlson, Stig Larsson, Ivica Crnkovic
Mälardalen University, School of Innovation, Design and Engineering, Västerås, Sweden
rikard.land@mdh.se, jan.carlson@mdh.se, stig.larsson@mdh.se, ivica.crnkovic@mdh.se

Abstract: This position paper describes how the combination of the Model-Driven Development (MDD) and Component-Based Software Engineering (CBSE) paradigms can support project monitoring and control, and project risk reduction. The core principle for this is articulated and named CARMA, and our research agenda and preliminary results are described. Through interviews, industry input, process simulation, tool implementation and pilot projects, and describing an extension of CMMI, we are exploring the CARMA principle in order to provide guidelines for MDD/CBSE projects.

Keywords: Model-Driven Development, Component-Based Software Engineering, Project Monitoring and Control, Risk management, CMMI, Empirical Studies.

1 INTRODUCTION

In this paper, we describe the preliminary results of an evaluation of the combination of two increasingly maturing approaches: Model-driven development (MDD) and Component-Based Software Engineering (CBSE). Current research efforts to combine these are mostly centered on technology, but there is a more or less implicit promise to reduce risk in development projects by adopting these two paradigms – especially when combined.

The assumed benefits are usually cast in technical terminology: the software will be correct by construction, component properties can be composed into system properties, or models at different levels are ensured to be consistent (Håkansson *et al*, 2006; Selic, 2003; Stahl and Völter, 2006). Only implicitly are the benefits understood as e.g. reduced costs and risk (Feiler *et al*, 2009). However, organizations need to change their culture and way of working compared to previous generations of software development paradigms (Selic, 2003). Such changes may include:

- New ways of formulating requirements
- Different approaches to verification (how/when)
- New activities, re-ordered activities, significantly different effort (relative and absolute) than usual
- New methods for project monitoring and control

As far as we know, the MDD/CBSE paradigms have not been thoroughly evaluated from this point of view (see section 2 for related work): will the required effort and commitment pay off? The purpose of this paper is to describe our initial results in evaluating the MDD/CBSE combination from the perspectives of risk management and project monitoring and control.

The goal of evaluating a combination of two paradigms, even from this more specific point of view, is extremely ambitious and needs to be made more concrete in order to be actionable. The next sections will describe: in more detail the particular technology chosen and related work (Section 2); the formulation of a principle capturing the essence of risk management and project monitoring and control in this context (Section 3); our research agenda, including the possible research methods to evaluate this principle, and preliminary results (section 4).

The main threat to validity of the evaluation is that we cannot do industrial case studies by the very nature of the topic. Our investigation should rather be seen as a feasibility study, where we collect insights by means of implementations, interviews, industrial experience, process simulation, tool implementation and student projects, and the formulation of a CMMI extension.

2 TECHNOLOGY AND RELATED WORK

This section describes the fundaments of the fields of Model-Driven Development (MDD) and Component-Based Software Engineering (CBSE), and its relation of the work presented in this paper.

2.1 Model-Driven Development

The principle behind Model-Driven Development (MDD) is to bridge the gap between various development artifacts such as requirements, architectural descriptions, lower-level designs, and implementation level through a series of more or less automatic translations (Selic, 2003; Stahl and Völter, 2006). MDD intends to make the development process more efficient (through automatic or semi-automatic translations), and enable earlier verification (of the models). The final software will thus to a large extent be correct by construction (Selic, 2003). OMG's Model-Driven Architecture, MDA (http://www.omg.org/mda), is one important instantiation of this principle, where the main objective is to achieve platform independence.

However, the MDD field focuses on languages that can capture as much as possible, because the next step in the process should ideally be generated automatically from a detailed model. The verification of models can only occur when a significant time of the project has passed. The concept of virtual integration in the SAVI program (Feiler *et al*, 2009) is similar to the CARMA principle we formulate, but we further clarify the essence of the principle and describe how project planning and milestones are integrated into the MDD/CBSE paradigm, and we explicitly incorporate high-level models and estimates which can be provided very early. This is how risk reduction is performed in industry today and which we think estimates will be an unavoidable part of project management and planning also in the future.

The literature on processes for Model-Driven Development (MDD) focuses mostly on the division into platform development and application development (Stahl and Völter, 2006; Kleppe, Warmer, Bast, 2003), and the new roles required for this (Aagedal and Solheim, 2004; Krahn, Rumpe, and Völkel, 2006; Guta, Szasz, and Schreiner, 2008). Also, while MDD relies on forward engineering in order to produce correct software, the combination MDD/CBSE in general also permits using pre-existing components produced in many different ways, including wrapped legacy code.

2.2 Component-Based Software Engineering

In Component-Based Software Engineering (CBSE), the software is designed and constructed as components with clear boundaries and explicit interfaces (Szyperski, 2002). This paradigm is successful in e.g. the desktop domain, and has also found its way into the embedded systems domain, which is our focus (Hänninen *et al*, 2008; Larsson, Wall, Wallnau, 2005; van Ommering, van der Linden, and Kramer, 2006). From a process perspective, this means the processes of component development and system development are treated separately, but interact (Crnković, Chaudron, and Larsson, 2005). Component development could be a result of system top-down decomposition, and result in either internal development or in hiring a subcontractor. A system may also be built from pre-existing components, such as Off-the-Shelf (OTS) components (components developed for the marketplace), or as part of a product line initiative (Clements and Northrop, 2001).

We have had to choose among the many component technologies in order to be specific enough, and identify the characteristics that support the project monitoring and control, as we envision it, to the largest extent. Although some literature, component models and component technologies describe the "component" concept as a deployable entity (Szyperski, 2002), others fundamentally assume there is a concept of *component identity* throughout the process, from early design to run-time. (For embedded software, it is even common that the component boundaries are optimized away during the deployment stage, when code is compiled and linked into one single binary image). To monitor the development (with support from automatic tools) as will be described in section 3, it is essential to adopt this second viewpoint. Also, there must be compositional reasoning theories and tools available for various component attributes, as well as an attribute framework making it possible to trace component attributes (such as timing properties, memory consumption) throughout the development process. Also, for embedded software, development of hardware models is an essential part of the development. This is true for the *ProCom* component model (Bureš *et al*, 2008; Sentilles *et al*, 2008) and associated research at the Progress Centre for Predictable Embedded Software Systems; there are other component models to which our evaluation will be applicable, but as a choice during our investigation they do not support all the desired characteristics to the same extent as ProCom, in particular the attribute framework: AADL (As-2 Embedded Computing Systems Committee, 2009), Autosar (www.autosar.org), SysML (www.sysml .org) and OMG's MARTE (www.omgmarte.org). Also, we have good access to ProCom, the Progress development environment, and the Progress researchers, which makes it suitable to choose this track.

2.3 Other Related Work

The principle presented in this paper inherits the basic ideas from the concepts of daily builds, continuous integration, continuous verification, and test-driven development (Beck, 1999; Duvall, Matyas, and Glover, 2007; Kruchten, 2004), and adapts them to fit the combined MDD/CBSE paradigm.

3 SUBJECT OF EVALUATION

3.1 Motivating Example

Figure 1 depicts an electronic stability control system of a car (figure: Bureš *et al*, 2008; example previously used in Land *et al*, 2009). Our envisioned way to run a MDD/CBSE-oriented project is:

- The different components may either be already existing, or to be developed. The existing ones may need modification. (In the figure, for example the *Stability Control System* may require new development, while the *Anti-lock Braking System* will be reused from a previous system with minor modifications, and for the *Wheels speed* component, there may be three potential COTS components available, etc.)
- There are certain properties the system must fulfill in order to be successful, such as response times and static memory consumption. (In the example, the latency from the *Wheels speed* input to the *Brake valves* output must be less than, say, 10 ms, and the software needs to fit in, say, 64 kb memory.) Clearly, the functionality is also a property that needs to be fulfilled, e.g. according to a requirements specification, a use case model, and/or state chart models describing the behavior.
- If these properties cannot be fulfilled, project management wants to be informed as early as possible, in order to identify mitigation solutions (e.g. acquire more powerful hardware, allocate human resources to optimize the source code, relax the requirements).
- The properties of interest are (in principle) derivable from knowledge of individual components' properties, their interconnections, and their allocation to hardware, and the characteristics of the hardware. For example, the response time depends on (at least): which components are invoked from input to output, the computation time needed by each component (which depends on hardware), and data transfer between components (which may be significant if this involves several communications over a network). Also behavioral diagrams can in principle be composed (Håkansson *et al*, 2008). (Another model, not shown here, is needed to describe the allocation of software components to hardware.) We assume that there exist reliable such composition theories for the properties of interest.
- Later in the development, it may be possible to generate the values for the properties of interest from implementations. Earlier in the development, it may be possible to estimate the values of these properties from half-finished implementations, or less refined models, adding a certain margin. Very early in the development, it is possible to provide values for these properties through expert estimates, or as allocated budgets to components based on the requirements on the system.
- Each attribute type (e.g. "memory consumption", "behavior") may thus be associated with many different values for a single component instance, which have been created differently (estimates, test results, static analysis results, model checking proofs), and on different versions of the component (Sentilles *et al*, 2009).
- It becomes possible to formulate milestones (e.g. project gates) in terms of expected values of the attributes. For example, if the requirement on static memory consumption is 32 kb for a component, we may define the goal for an early milestone to be 40 kb, generated from a model in a language known to give pessimistic values). A later milestone may be defined for this property as 24 kb, based on static analysis for a point in time where an incomplete implementation should be achieved, with a known set of (planned) completed features. (If these features are not

implemented, this should raise a flag in another milestone criterion, for attribute "functionality".)

The key observation is that all of these types of values from very different sources are valuable at different points during the project from a project monitoring and control viewpoint, and that they can be treated in a uniform manner independent of their source, with support for more or less automatic generation and composition of the values.

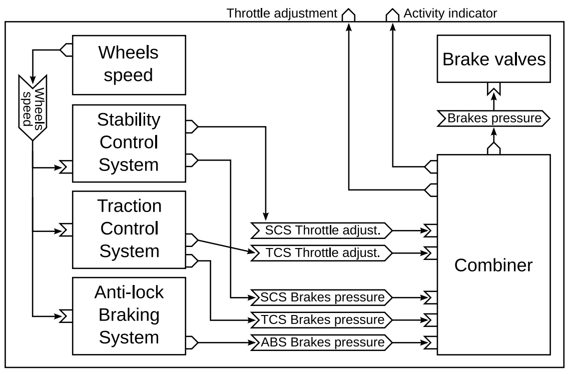

Figure 1: Component design of an electronic stability control (ESC) subsystem of a car.

3.2 The CARMA Principle

The principle we envision is implicit in (the combination of) MDD and CBSE, but has not before been clearly articulated, can be formulated as:

- **Components.** Choose a component technology which supports compositional reasoning of component properties. As early as possible, define the components of your system (i.e. the architectural structure).
- **Attributes.** Keep track of the properties of (components of) your system through component attributes. Use a tool that supports management of these properties, including automatic composition.
- **Requirements.** Refine your high-level system requirements into product requirements, and specify these in terms of the attributes which are analyzable with (tools supporting the) composition theories.
- **Milestones.** Formulate milestones (e.g. project gates) in terms of tuples: <expected value in relation to product requirement; method to generate this value>.
- **Analysis.** Perform verification analysis at the defined milestones. In addition, the individual developers, architects, project manager, etc., may perform analyses of interest at any time; this resembles debugging in direct connection to implementation, which is informally done (i.e. not mandated by a formal process) but an invaluable tool for the individual developer before passing the code (or, in MDD, the model) on to verification as part of the formal process.

We call this principle the CARMA principle (Components, Attributes, Requirements, Milestones, Analysis).

The CARMA principle captures what we believe is a major opportunity in practice if adopting the MDD/CBSE paradigm. There is risk reduction inherent in the "correct by design/construction" paradigm, but it is important to leverage on this at a project management level, including the time dimension, the possibility of changed requirements, which may be due to external events as well as to internal events in the project. We are performing several complementary studies of this principle, each aiming at providing different types of insights. The studies explore the characteristics of MDD/CBSE projects implementing the CARMA principle, as is explained in detail in the next section.

4 PRELIMINARY RESULTS

This section outlines four evaluation methods which are all underway in our research agenda. For each we describe the research method shortly, the evaluation point of view, and preliminary results.

Our main basis for both interviews and extensions of existing implementation is the technology development at the Progress Centre for Predictable Embedded Software Systems (http://www.mrtc.mdh.se/progress/).

4.1 Interviews and Documentation

Research question: How do the researchers developing modeling languages and methods, analysis methods, synthesis to executable, etc. envision the benefits of their methods? Is the approach in large feasible for embedded systems projects?

Research method: We have performed interviews with researchers of various MDD/CBSE modeling/ analysis/construction methods, and tool builders. As a concrete artifact discussed during the interviews, a process simulation model (see section 4.2) has been iterated with these researchers. We have also studied industrial requirements specifications with the objective of identifying how closely it matches the proposed approach.

Preliminary results: The interviews and ongoing collaboration has led to the formulation of the CARMA principle as well as the construction of a simulation model (see section 4.2). The study of industrial requirements specifications have led to the following observations:

- Some requirements are specified in enough detail to allow specific pass/fail criteria to be specified, and are relatively easy to map to the CARMA principle. In particular:
  - Many product requirements are specified in terms of execution steps, sometimes using some kind of dynamic diagram. Example: "During startup, register X shall first be read to determine the cause of the last shutdown/reset. If the cause is… then do…"
  - Some product requirements describe timing behavior of some execution steps. "Example: The first phase of startup shall take less than X ms; the second step Y ms; …"
  - Some product requirements, but not many, are hardware specifications. Example: "The processor shall be of type X"; "the software image shall fit in X bytes of memory".
- Requirements on safety-related functions are formulated to be unambiguous and verifiable, and with the highest level of detail (including e.g. timing and resource usage as described in the bullets above). This is due to the potentially catastrophic effects of a specification error.

4.2 Process Simulation

Research method: We are simulating a queuing network model (Kobayashi, 1978) of a development process where the CARMA principle is adopted. We vary input parameters such as requirements volatility, the likelihood of detecting problems in analysis and verification, the amount and points in time verification is performed (e.g. milestones throughout the project, and/or only or mainly at the end of the project), the actions taken in case a problem is found (e.g. try to optimize, re-architect the system, drop or relax requirements, etc.). This model is iterated with the interviewees as indicated in section 4.1.

Research question: If adopting the MDD/CBSE paradigm and the CARMA principle, what factors affect the project outcome the most?

Preliminary results: The simulation results so far indicate that with frequent milestone verifications, the same amount of effort is spent on verification as when verification is performed at the end. However, the simulation results indicate several drawbacks with verification occurring only at the end: 1) more verifiers (i.e., people) are needed at the same time, 2) problems are found late, which cause a feedback of error correction and re-verification (it is easy to translate this into a sense of urgency and "fire-fighting" in the development organization), and 3) the project time is somewhat prolonged (but not very much). Also, with more volatile requirements (i.e. changed or added throughout the project) the total effort is increased. These results seem intuitively seem to be applicable more generally, and we take this as a sign of credibility of the simulation model. We hope that the simulation results, once fully analyzed, will provide concrete guidelines on how to plan and dimension MDD/CBSE development projects in different circumstances.

4.3 Tool Implementation

Research method: We are implementing an extension of the Progress Integrated Development Environment, which implements the desired attribute framework for components (Sentilles *et al*, 2009). In this extension, requirements on attribute values are distinguished from actual attribute values, and there will be a general mechanism to compare these, as well as display summaries and visualizations of milestone verifications, etc. We will then use this tool in student projects.

Research question: Through the construction of the tool extension, we will hopefully realize details earlier overlooked. By using the tool in student projects, we are able to collect insights. We can nevertheless observe the amount of perceived overhead the approach introduces, and suggestions for e.g. automation and user interface improvements.

4.4 Presentation of the principle as a CMMI extension

Research method: We are systematically extending a well-known process model, CMMI (Chrissis, Konrad, and Shrum, 2007), to clarify and explain the CARMA principle.

Research question: This is dissemination rather than evaluation, similar to CMMI extensions for safety-critical systems (Defence Materiel Organisation, Australian Department of Defence, 2007) and an extension for the medical domain (McCaffery, Burton, and Richardson, 2009).

Preliminary results: An initial version has been published (Land *et al*, 2009), and we are currently extending the guidelines to cover not only the software components and associated models are covered, but also data (e.g. databases) and hardware nodes and networks, which are extremely important for accurate analysis of e.g. timing.

5 CONCLUSIONS

Our findings so far indicates that the MDD/CBSE combination can be used in development projects and potentially reduce costs, time, and especially risk. With input from the research fields of MDD and CBSE as well as industry, the CARMA principle has been formulated and is shown to be reasonably realistic. When there are mature tools available, the results may be developed into guidelines for application. Further studies will also need to go beyond ProCom.

ACKNOWLEDGEMENTS

This work was partially supported by the Swedish Foundation for Strategic Research (SSF) via the strategic research centre PROGRESS.

REFERENCES

Aagedal, J. Ø., Solheim, I., 2004. "New Roles in Model-Driven Development", *European Workshop on MDA*.

As-2 Embedded Computing Systems Committee, 2009. *Architecture Analysis & Design Language (AADL)*, Standard Document Number AS5506.

Beck, K., 1999. *EXtreme Programming EXplained: Embrace Change*. Addison Wesley.

Bureš, T., Carlson, J., Crnković, I., Sentilles, S., and Vulgarakis, A., 2008. *ProCom - the Progress Component Model Reference Manual, version 1.0*, ISRN MDH-MRTC-230/2008-1-SE.

Chrissis, M. B., Konrad, M., and Shrum, S., 2007. *CMMI Second Edition : Guidelines for Process Integration and Product Improvement*, Addison Wesley.

Clements, P., Northrop, L., 2001. *Software Product Lines: Practices and Patterns*. Addison-Wesley.

Crnković, I., Chaudron, M., and Larsson, S., 2005, "Component-based Development Process and Component Lifecycle". *Journal of Computing and Information Technology* 13(4).

Defence Materiel Organisation, Australian Department of Defence, 2007. *+SAFE, V1.2 : A Safety Extension to CMMI-DEV, V1.2*, SEI technical note CMU/SEI-2007-TN-006.

Duvall, P., Matyas, S., and Glover, A., 2007, *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Professional.

Feiler, P., Hansson, J., de Niz, D., Wrage, L., 2009. *System Architecture Virtual Integration: An Industrial Case Study*, technical report CMU/SEI-2009-TR-017, Software Engineering Institute.

Guta, G., Szasz, B., and Schreiner, W., 2008. *A Lightweight Model Driven Development Process based on XML Technology*. Draft Technical report 08-01 in RISC Report Series, University of Linz, Austria.

Håkansson, J., Carlsson, J., Monot, A., Pettersson, P., 2008. "Component-Based Design and Analysis of Embedded Systems with UPPAAL Port", *6th International Symposium on Automated Technology for Verification and Analysis*, Springer.

Hänninen, K., Mäki-Turja, J., Sandberg, S., Lundbäck, J., Lindberg, M., Nolin, M., and Lundbäck, K.-L., 2008. "Framework for Real-Time Analysis in Rubus-ICE", in *13th IEEE International Conference on Emerging Technologies and Factory Automation*, IEEE.

Kobayashi, H., 1978. *Modeling and Analysis: An introduction to System Performance Evaluation Methodology*, Addison-Wesley Publishing Company.

Kleppe, A., Warmer, J., Bast, W., 2003. *MDA Explained : The Model Driven Architecture: Practice and Promise*, Pearson Education.

Krahn, H., Rumpe, B., and Völkel, S., 2006."Roles in Software Development using Domain Specific Modelling Languages", in *6th OOPSLA Workshop on Domain-Specific Modeling*.

Kruchten, P., 2004. *The Rational Unified Process : An Introduction*. Addison-Wesley, 3rd edition.

Land, R., Carlson, J., Larsson, S., and Crnković, I., 2009. "Towards Guidelines for a Development Process for Component-Based Embedded Systems", in *International Conference on Computational Science and Applications (ICCSA)*, Springer.

Larsson, M., Wall, A., and Wallnau, K., 2005. *Predictable Assembly: The Crystal Ball to Software*. ABB Review.

McCaffery, F., Burton, J., Richardson, I., 2009. "Improving software Risk Management in a Medical Device Company", in *ICSE Companion*.

Selic, Bran, 2003. "The Pragmatics of Model-Driven Development", *IEEE Software* 20(5), IEEE.

Sentilles, S., Vulgarakis, A., Bureš, T., Carlson, J., and Crnković, I., 2008. "A Component Model for Control-Intensive Distributed Embedded Systems". In *Proceedings of the 11th International Symposium on Component Based Software Engineering*, Berlin.

Sentilles, S., Stepan, P., Carlson, J., Crnkovic, I., 2009. "Integration of Extra-Functional Properties in Component Models", in *12th International Symposium on Component Based Software Engineering*, Springer.

Stahl, T., Völter, M., 2006. *Model-Driven Software Development : Technology, Engineering, Management*. John Wiley & Sons.

Szyperski C., 2002. *Component Software - Beyond Object-Oriented Programming*, Addison-Wesley, 2nd edition.

van Ommering, R., van der Linden, F., Kramer, J.,and Magee, J., 2000. "The Koala Component Model for Consumer Electronics Software". *IEEE Computer* 33(3), IEEE.