Software Maintenance Research in the PROGRESS Project for Predictable Embedded Software Systems

Johan Kraft, Holger M. Kienle, Thomas Nolte, Ivica Crnkovic, and Hans Hansson
Mälardalen University, Västerås, Sweden
http://www.mrtc.mdh.se/progress/

***Abstract*—PROGRESS is a project and strategic research centre at Mälardalen University in Sweden that is funded for 2006–2010 by the Swedish Foundation for Strategic Research (SSF). PROGRESS research targets embedded software in the vehicular, automation, and telecom domains, focusing on the areas of component technology, verification and analysis for predictability, predictable execution, as well as reuse and maintenance of legacy embedded software. We first describe the funding, organization and research areas of PROGRESS, and then give several examples of PROGRESS research that addresses maintenance of legacy embedded software with the goal to improve program comprehension, quality assurance, and debugging. Specifically, we describe research in tracing and trace visualization, impact analysis of temporal behavior, slicing, and system-specific static analyses.**

I. PROGRESS Motivation

The PROGRESS Centre for Predictable Embedded Software Systems (PROGRESS) at Mälardalen University (MDH) in Sweden is a research project funded for five years by the Swedish Foundation for Strategic Research that provides an umbrella for industry-oriented research in the domain of embedded and real-time software systems.

Under PROGRESS several independent but closely related research groups collaborate to advance the state-of-the-art in the design, analysis, validation, execution, and maintenance of embedded systems. As articulated in the grant documents, PROGRESS' vision is "to be a world-leading research center known for industrial strength research in component-based development of predictable embedded software, and for being a preferred research partner for industry."

PROGRESS is motivated by the increasing challenges that embedded and real-time systems have to meet, and by the observation that for mechatronic and cyber-physical systems the software is a key factor for industrial competitiveness. PROGRESS strives to make software more predictable by enabling to reason about (quality) attributes that are important for embedded systems. PROGRESS targets embedded systems in industry, which typically are special-purpose systems developed for control of a physical process with the help of sensors and actuators. An embedded system is often a mechatronic system, which means that its construction requires a combination of mechanical, electronic, control, and computer engineering.

Among others, PROGRESS has collaborations with industry in the vehicular domain (Bombardier, CrossControl, Scania, and Volvo), automation and power technologies (ABB), and telecommunications (Ericsson). Embedded systems developers in industry have to juggle several challenges. First, embedded systems are becoming more complex and feature-rich, and the growth rate of embedded software is accelerating [1]. Developers have to accommodate this trend without sacrificing key quality attributes. Second, correctness of an embedded real-time system also depends on *timeliness*, i.e., that the latency between input and output does not exceed a specific limit (the deadline). Third, quality control has to be a concern during the entire life cycle and needs to be supported by the software process. Especially for safety there is increasing pressure from the legislature to follow (safety) standards and to document standards compliance [2]. Lastly, as in other software domains, important concerns are maintenance of systems and migration of legacy system towards new platforms and technologies.

After giving more detail about PROGRESS' funding, structure and research areas in Section II, we discuss in Section III how research in PROGRESS addresses the legacy and maintenance challenges of embedded systems. Section IV concludes the paper.

II. PROGRESS Research Centre

PROGRESS is funded by the Swedish Foundation for Strategic Research (SSF).[1] SSF supports scientific, technological and medical research—both fundamental as well as applied research—with the objective to promote Sweden's future competitiveness [3].

Among other forms of support, SSF funds *strategic research centres*, where each center unites independent but co-located research groups at a university that "collaborate to solve an important research problem under the strong, uniting leadership of a center director."[2] As of Spring 2008, SSF funds about 20 strategic research centres, one of which is PROGRESS—the only one with a software focus. PROGRESS has been funded for 5 years (2006 to 2010) for a total of SEK 49 million (about EUR 5 million).[3]

The organizational structure of PROGRESS is depicted in Figure 1. At the bottom PROGRESS is structured into several *research directions*[4] that cover a broad range of embedded

[1] http://www.stratresearch.se/en/

[2] http://www.stratresearch.se/en/About-SSF/

[3] To put this in perspective, the total funding provided by SSF in 2009 was SEK 500 million.

[4] A researcher direction identifies a research area or topic and the researchers working on it. A research direction typically crosscuts research groups.

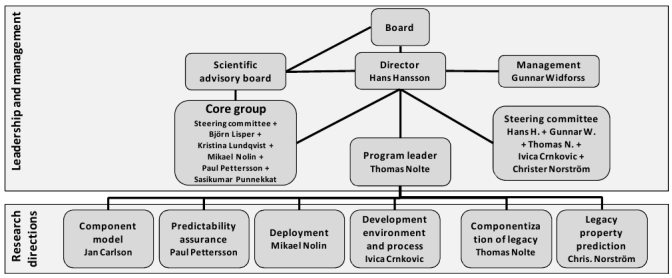

Fig. 1. Organization of PROGRESS

systems research:

- Predictability assurance develops theories, techniques and tools for reasoning about attributes that are important for embedded systems such as timeliness, resource consumption and reliability.
- Deployment provides mappings and (model-driven) transformations from the design to executable code on particular target platforms and within the constraints of predictability assurance.
- Component models play a central role because they are the means to describe the system design, including its properties and constraints related to predictability and deployment. PROGRESS has developed the ProCom component models [4] [5].
- A tool infrastructure based on Eclipse, now called PrIDE (http://www.idt.mdh.se/pride/), makes it possible to create models interactively and to integrate research results and (basic) tools [6].
- Componentization of legacy assists in the migration and wrapping of legacy code to component-based models and architectures supported by PROGRESS as well as platform (virtualization) techniques for predictable execution of legacy code.
- Legacy property prediction provides techniques to analyze legacy code with the goal to obtain useful properties that can assist in the maintenance, predictability assurance and componentization of existing code.

In Section III we elaborate on the research directions that are relevant in the context of software maintenance and reengineering.

The research directions are coordinated and steered by a management structure (cf. top part of Figure 1). The center director is the scientific and administrative leader and assisted by the program leader. The director, program leader, and research coordinator form a steering committee responsible for the operation of the center. For strategic discussions, the steering committee is extended into the core-group by participation of leading researchers from all involved research groups. Perhaps most noteworthy is the concept of a scientific advisory board that consists of three externals that are international experts in the field. The scientific advisory board monitors and evaluates the scientific development and plans of the center, and provides guidance, suggestions, and constructive criticism to the management and board. Its competent advice has been of great help in formulating plans and sharpening the focus of PROGRESS.

While PROGRESS is a national initiative that funds a single university, individual research directions have additional sources of funding such as FP7, Swedish Research Council (VR), The Knowledge Foundation (KKS) and The Swedish Governmental Agency for Innovation Systems (VINNOVA) and are collaborating with other European research institutions. Thus, the SSF PROGRESS funding can be seen as a stable backbone that enables to tackle longer-term, overarching research endeavors.

After almost five years of PROGRESS, it is apparent that, without such an initiative, integration of research within MDH would have taken place to a much lesser extent, resulting in a more "scattered" research effort with less synergy, cross-pollination, and integration.

III. PROGRESS Research in Software Maintenance

A key research goal for PROGRESS is to provide techniques and tools for the predictable development of embedded systems using a component-based and model-driven approach [7]. Generally, predictability concerns the possibility to establish absence or presence of certain properties. PROGRESS focuses on predictability in terms of timing, reliability and resource usage. Within component-based development, predictability analyses guide the design and selection of components and the deployment to a certain platform. During the analysis and design, component-based models are constructed. At this stage evaluation of predictability properties can be conduced based on the information available in the (abstract) models. For example, components in the model can be augmented with properties that allow a prediction of timing behavior of a component as well as of the composition of components.

While PROGRESS emphasizes forward engineering, it recognizes that our industrial partners have a significant amount of legacy code. Reusing legacy real-time software in new system generations is part of PROGRESS. Typical scenarios of reuse are (1) applying PROGRESS forward-engineering techniques when adding functionality to an existing platform, and (2) migration of selected parts of a legacy system towards components that adhere to the PROGRESS component models.

PROGRESS also aims to provide support for maintenance of large embedded system that have real-time constrains and safety concerns. The code bases are often written in C or a restricted subset of C++ and are hundreds of thousands lines of code. For example, one system under study is a control system for industrial robots from ABB Robotics. It has about 2500 KLOC, written mostly in C. The software system architecture was conceived in the early 1990s and has evolved considerably since then. Thousands of changes have been performed over this time, by hundreds of software developers, of which many are no longer at the company. Some parts of the system have hard real-time constraints, while others have strong focus

on performance, as the robot's maximum production speed is mainly limited by the computational performance of the controller.

In the following we highlight several research topics and results that show the range of PROGRESS' research in software maintenance. The ABB Robotics code has been used for case studies in all of the projects described below. Most projects use more than one industrial code base to demonstrate applicability in an industrial setting.

A. System-Specific Analyses

In this most recent project, we are exploring system-specific analyses as a means to improve upon quality assurance of embedded software [8]. Industry routinely uses coding standards such as MISRA C/C++ and associated static checkers for embedded systems, but this approach often produces many false positives (i.e., warnings of non-conforming code that can be ignored but are tedious to filter out). In contrast to generic analyses, system-specific analyses are custom-tailored to a specific system because such analyses can take advantage of domain/system knowledge that is not available to more general analyses. As a result, system-specific analyses can be designed such that they are as unintrusive as possible (e.g., adding low overhead to the existing development process and exhibiting comparably fewer false positives) and that they target (recurring) quality concerns that the system currently exhibits. Thus system-specific analyses promise to show immediate pay-offs (e.g, improving the software's quality and reducing faults).

To study the feasibility of this approach we are using the ABB Robotics code as a case study. We have been able to identify and describe unique coding patterns and conventions in the ABB code that would be suitable targets for system-specific analyses [8]. For example, an analysis could assure that certain conventions about message formats and communication patterns among tasks are actually met. Another example is an analysis that could assure that no modifications to the priorities of tasks are introduced, because this may result in subtle failures that are often difficult to track down.

We have started to implement these analyses with the help of Understand Analyst (http://www.scitools.com/), a commercial, maintenance-oriented IDE that supports program understanding and reverse engineering activities. Understand builds a database model of the source code, describing what symbols exist and where they are used. Our analyses leverage Understand's Perl API to access the database.

B. Dynamic Analysis and Visualization of Tasks

The Tracealyzer (cf. Figure 2) is a trace analysis and visualization tool for embedded systems focusing on high-level runtime behavior, such as scheduling, resource usage and OS calls [9] [10]. It consists of a trace recorder, which generates a memory-efficient binary trace file and a visualizer. The visualizer displays task traces using a novel visualization technique that focuses on the task preemption nesting and only shows active tasks at a given point in time.

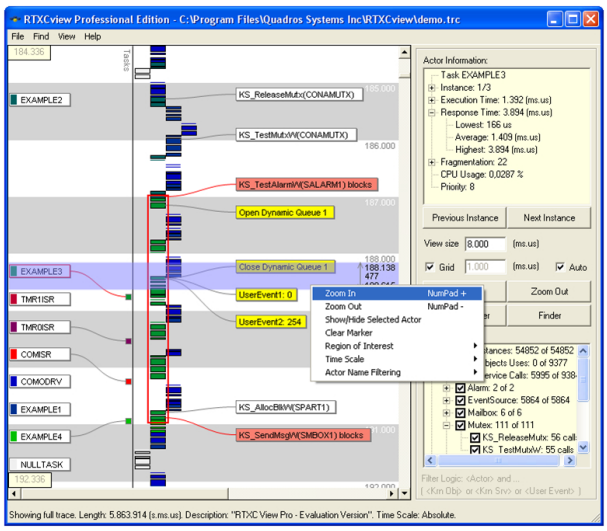

Fig. 2. The Tracealyzer tool [10]

The first version of the Tracealyzer was developed in 2004 in collaboration with ABB Robotics. The Tracealyzer is today used systematically at ABB Robotics for troubleshooting and for performance measurements, and the monitoring is active by default in all of its robots since 2005, also in production mode.

The Tracealyzer is since 2009 a commercial product in the spin-off company Percepio AB (http://www.percepio.se/), in a close collaboration with the OS provider Quadros Systems.

C. Program Slicing for Large Software Systems

A significant PROGRESS result is *Katana*, a new method for *program slicing*, developed between 2008–2009 [9]. Program slicing is an established concept (first proposed by Weiser [11]) that is useful in many areas related to software maintenance (e.g., program comprehension, debugging, software visualization, and testing), but so far has had only minor industrial impact. This is probably due to the fact that available tools do not scale sufficiently for industrial code bases. The runtime and memory usage grows too fast with the size of the program.

The dominating approach to program slicing since the mid 1990s is based on the *system dependence graph* (SDG) [12]. Constructing such graphs is however very time consuming for large programs, as a detailed analysis is required over the whole code base. The Katana method uses a different approach, which scales to large industrial software systems. It is based on a lightweight program model, a *symbol database*, which essentially is an index over the source code and can be constructed using a lexical scan.

In an evaluation presented in [9], a leading commercial tool with slicing capabilities (GrammaTech's CodeSurfer) was unable to analyze 183 KLOC of industrial C code given 92 hours. In contrast, Katana's lightweight program model was constructed in only 15 seconds and subsequent slicing queries took from 2 seconds up to 3 minutes, in the latter case for a

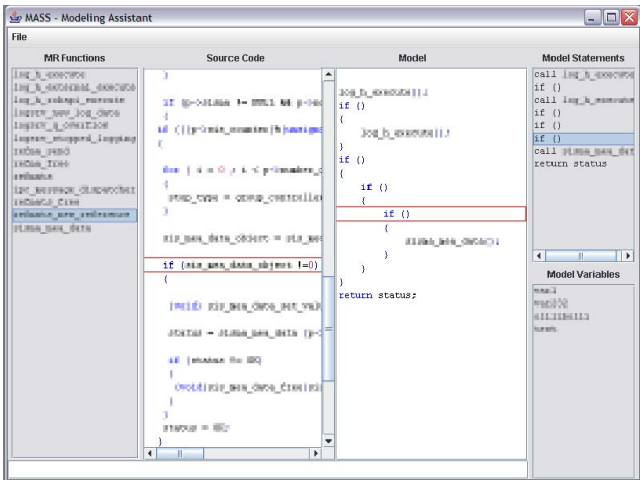

Fig. 3. The MASS tool (industrial sample code obfuscated) [13]

large slice containing 2000 program statements. The Katana prototype is currently restricted to C code and implemented on top of Understand Analyst.

D. Impact Analysis of Temporal Behavior

As already mentioned, timing-related properties are an important concern for embedded and real-time systems. When evolving such systems it is highly desirable to predict the impact that a change has on the system's temporal behavior such as response time and utilization of limited logical resources.

For legacy systems, which often do not have formal analysis models, PROGRESS tackles this problem with simulation-based impact analysis. Using a combination of static and dynamic analysis, a discrete-event simulation model is extracted from the legacy system [9] [13].

Instead of making changes directly to the source code, changes can be applied to the model instead, resulting in a "hypothetical" model that captures a proposed change. By using the RTSSim simulation framework developed for this purpose [9], the (hypothetical) model can then provide estimates on the timing properties. The timing information for the model is derived from Tracealyzer runtime recordings. Since results are based on measurements and simulation, the result is an approximation, a best-effort approach, and not formally sound.

An earlier approach to model extraction resulted in an interactive modeling tool known as MASS, implemented in Java (cf. Figure 3) [13]. In this semi-automatic tool, the source code is presented on the left side while the corresponding model skeleton that needs to be manually completed is given on the right. An automatic solution was later realized in the Model eXtraction Tool for C (MXTC) [9]. This is essentially a statement-level code filter, based on the Katana slicing method introduced above.

IV. Conclusions

We have described the PROGRESS project and center and its research in software maintenance of industrial embedded systems. The described projects span dynamic as well as static approaches to obtain facts from embedded systems, using analyses and visualizations to help with debugging, program comprehension, impact analysis and quality assurance.

PROGRESS' research has academic impact as testified by the publications generated up until half-time (2006–2008), which include 146 peer-reviewed conference articles, 23 journal articles or book chapters, 2 books, and 13 dissertations [14]. Besides, a key approach of PROGRESS is close collaboration with industry, resulting in applied research relevant to industrial needs. This has resulted in several projects that are in various stages of being commercialized, among them the Tracealyzer tool.

Acknowledgments

This work is supported by the Swedish Foundation for Strategic Research (www.stratresearch.se) through the PROGRESS Centre for Predictable Embedded Software Systems (www.mrtc.mdh.se/progress/).

References

[1] C. Ebert and C. Jones, "Embedded software: Facts, figures and future," *IEEE Computer*, vol. 42, no. 4, pp. 42–52, Apr. 2009.

[2] M. Åkerholm, R. Land, and C. Strzyz, "Can you afford not to certify your control system?" *iVTinternational*, Nov. 2009, http://www.ivtinternational.com/legislative_focus_nov.php.

[3] SSF, "Research that shapes our future," 2010, http://www.stratresearch.se/Global/publikationer/om%20SSF/Research%20that%20shapes%20our%20future%202010.pdf.

[4] M. Åkerholm, J. Carlson, J. Fredriksson, H. Hansson, J. Håkansson, A. Möller, P. Pettersson, and M. Tivoli, "The SAVE approach to component-based development of vehicular systems," *Journal of Systems and Software*, vol. 80, no. 5, pp. 655–667, May 2007.

[5] S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic, "A component model for control-intensive distributed embedded systems," in *11th International Symposium on Component Based Software Engineering (CBSE 2008)*, M. R. Chaudron and C. Szyperski, Eds., Oct. 2008, pp. 310–317.

[6] S. Sentilles, A. Pettersson, D. Nyström, T. Nolte, P. Pettersson, and I. Crnkovic, "Save-IDE – a tool for design, analysis and implementation of component-based embedded systems," in *Research Demo Track of the 31st International Conference on Software Engineering (ICSE 2009)*, May 2009, pp. 607–610.

[7] H. Hansson, I. Crnkovic, and T. Nolte, "The world according to PROGRESS," 2007, internal document.

[8] H. M. Kienle, J. Kraft, and T. Nolte, "System-specific static code analyses for complex embedded systems," *4th International Workshop on Software Quality and Maintainability (SQM 2010)*, Mar. 2010, http://holgerkienle.wikispaces.com/file/view/KKN-SQM-10.pdf.

[9] J. Kraft, "Enabling timing analysis of complex embedded systems," PhD thesis No. 84, Mälardalen University, Sweden, Aug. 2010, http://mdh.diva-portal.org/smash/get/diva2:312516/FULLTEXT01.

[10] J. Kraft, A. Wall, and H. Kienle, "Trace recording for embedded systems: Lessons learned from five industrial projects," *1st International Conference on Runtime Verification (RV 2010)*, pp. 315–329, 2010.

[11] M. Weiser, "Program Slicing," in *5th International Conference on Software Engineering (ICSE'81)*, Mar. 1981, pp. 439–449.

[12] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural slicing using dependence graphs," in *ACM SIGPLAN 1988 conference on Programming Language design and Implementation (PLDI'88)*, Jun. 1988, pp. 35–46.

[13] J. Andersson, J. Huselius, C. Norström, and A. Wall, "Extracting simulation models from complex embedded real-time systems," in *1st International Conference on Software Engineering Advances (ICSEA 2006)*, Oct. 2006.

[14] H. Hansson, T. Nolte, J. Axelsson, M. Björkman, J. Carlson, I. Crnkovic, B. Lisper, K. Lundqvist, C. Norström, P. Pettersson, S. Punnekkat, and M. Sjödin, "PROGRESS Half-time report: January 2006 – June 2008 (edited version)," 2010, http://www.mrtc.mdh.se/publications/2364.pdf.