

Dynamic memory management for software product family architectures in embedded real-time systems

Christian Del Rosso
Nokia Research Center
P.O Box 407
FIN-00045 NOKIA GROUP (Finland)
christian.del-rosso@nokia.com

Abstract

Dynamic memory management is one part of the software system that influences the performance and the cost of a product the most. In the context of an embedded real-time system, several requirements must be taken into account. The system must be optimized due to the limitation of memory. Real-time deadlines must be respected: the dynamic memory management system must allocate and deallocate blocks in due time. One more challenge is represented when a dynamic memory management system is developed for a product family architecture, which is representative of a set of related products. We present a scenarios-based approach to analyze and evaluate dynamic memory management systems for embedded real-time systems in a software product family architecture. Architectural transformations and improvements against the tradeoffs for the software product family are analyzed.

1. Introduction

A dynamic memory management system has the task to allocate and deallocate memory blocks efficiently and in case of real-time system in due time. The evaluation of dynamic memory management systems for real-time embedded systems is targeted to performance and memory efficiency.

However, a more challenging objective is represented when the scope of the memory optimizations are targeted to a set of related embedded real-time systems which are part of a software product family. A software product family is a set of software-intensive systems having common assets and sharing architectural properties [1].

In this paper we discuss the evaluation of dynamic memory management systems in the context of a software product family architecture for an embedded real-time system.

Members of the same product family can have very different patterns of memory allocations. As result, the products in the same family can require a different architecture for dynamic memory management systems. Therefore, in the discussion section particular emphasis is given to the analysis of dynamic memory management systems in a software product family architecture.

2. Software Performance Method

We have used a scenarios-based approach to evaluate the dynamic memory management system in the software product family.

Scenario selection is the first step in the process and sets the focus of the analysis. Evaluating the dynamic memory management system means finding significant scenarios in terms of memory usage. However, features with strict real-time requirements must be included and analyzed.

Different patterns of data allocations can be found in executing the scenarios. Ramps, peaks and plateaus were the patterns described by Wilson et al. [3]. In addition to heavy memory hitter scenarios, memory optimization must include scenarios that allow the software product family to be analyzed. To evaluate software product family architectures, scenarios must be significant for the set of products in the family unless the evaluation is targeted to special features of one product.

Low-end products have different feature sets, and therefore they have different and sometimes conflicting memory requirements than high-end products. Multimedia features have patterns of data allocations that differ for core features of the mobile phone such as data calls or SMS sending. The software product family features list is the starting point for selecting significant scenarios. However, in the scenario selection process, interviews and brainstorming sessions with developers and architects are fundamental.

The number of scenarios depends on the analysis scope

and the time available for the analysis. A right balance must be found. A limited number of scenarios means an analysis better focused; on the other hand, too a narrow scope prevents a comprehensive and sound analysis.

The data extraction phase follows the scenarios selection process. Data are extracted from representative members of the software product family (containing the features to be analyzed) using trace instrumentation. With trace instrumentation, data of allocations and deallocations can be collected in log text files and fed to the next phase: the simulation and modelling phase.

Using data of memory allocations and deallocations we are able to analyze the data characteristics and requirements of the features selected in the scenarios. Simulations are then used to test and evaluate different allocation strategies.

The last phase is the analysis of the results obtained from the simulations. The analysis must consider the best dynamic memory management systems and the implications for the software product family architecture. The best dynamic memory management system does not exist as an absolute concept. A dynamic memory management system in a product family can mean several dynamic memory management systems for different instantiations of the products or a common dynamic memory management system for all the products in the family.

The product family architecture is affected by the evaluation process and improvements must be analyzed against the impact on the software architecture. The process is iterative and the previous phases can be reiterated more times if needed.

3. Dynamic Memory Management Systems for Software Product Families

Evaluating and analyzing dynamic memory management systems in embedded real-time systems for software product family architectures requires an understanding of several domains. Embedded systems have limited resources and require a different approach than the PC world. The lack of virtual memory support, and the limitation of memory and CPU make a design targeted to resource optimization necessary. Real-time systems add time constraints.

The software product family architecture domain introduces another interesting dimension. A software product family architecture represents the common architecture for a set of related products. The products in the family share a common reference architecture and common assets, called commonalities. Analyzing and improving the performance of the dynamic memory system means analyzing the performance and the requirements of the system for the whole set of products in the family.

The products in the family share a common architecture; however, the range of products in the family may have con-

flicting requirements concerning quality attributes. Low-end products have different memory requirements than high-end products.

Whether the analysis's objective is the whole set of products or only a subset of the products in the software product family, the evaluation will include the whole product family architecture. One size does not necessarily fit all and an in depth analysis of the tradeoffs is fundamental.

A common dynamic memory management system part of the product family has the advantage of reducing the variability in the architecture, thus, increasing the benefit brought by product family architectures.

Otherwise, a variation point to the architecture can be created. The variation point in the product family architecture can include the possibility of choosing between different dynamic memory management systems, one for every subset of the product family, e.g. high-end and low-end. The variation point can be implemented at run-time or at compile-time, and the choice is between a static binding or dynamic binding. This solution has the benefit of optimizing the performance at a finer level of granularity. On the other hand, it is disadvantaged in software maintainability activity. Creating multiple versions of the dynamic memory management system component means maintaining them, and numerous publications report that 80% of the total cost of traditional software is spent on maintenance [2].

The creation of variation points increases the product family variability, decreasing the advantages of such architecture: the commonality of the assets with the consequent benefit of software quality and time-to-market.

4. Conclusion and future work

In this paper we have described an approach for the evaluation and the analysis of the dynamic memory management systems in three domains: embedded, real-time and product family architecture.

In future work, we plan to continue the study of how to improve software performance in the context of software product family architectures. An additional interesting topic is to study other non-functional quality attributes and their impact on software product family architectures.

References

- [1] J. Bosch. *Design and Use of Software Architectures*. Addison Wesley, 2000.
- [2] T. M. Pigoski. *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. John Wiley & Sons, 1996.
- [3] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles. Dynamic storage allocation: A survey and critical review. *Proc. Int. Workshop on Memory Management*, Kinross, Scotland, UK, September 1995.