Increasing Accuracy of Property Predictions for Embedded Real-Time Components

Johan Fredriksson
Mälardalen University, Mälardalen Real-Time Research Centre
Västerås, Sweden, johan.fredriksson@mdh.se

Abstract

Many embedded systems for vehicles and consumer electronics critically depend on efficient, reliable control software, and practical methods for their production. Component-based software engineering for embedded systems is currently gaining ground since variability, reusability, and maintainability are supported. However, existing tools and methods do not guarantee efficient resource usage in these systems. We present methods that increases the accuracy in extra-functional property predictions, by considering context without restricting reusability; thus, enabling less pessimistic extra-functional component properties and, hence, improving resource utilisation.

1 Introduction

As the complexity and the amount of functionality implemented by software in embedded systems increase, so does the cost for software development. Also, since product lines are common within the embedded domain, issues of commonality and reuse are central for reducing cost. Component-Based Development (CBD) has shown to be an efficient and promising approach for software development, enabling well defined software architectures as well as lowering development-costs and time-to-market by increasing reusability [6, 1].

Embedded Real-Time Systems are soon integrated in all everyday appliances. Already systems like cars, TV-sets and DVD players are controlled by embedded systems[7]. The software must conform to often very limited resources in terms of calculation power and memory. Due to the nature of such systems they must also often conform to their physical environment they control, i.e., they must strictly conform to stipulated timing requirements. Therefore, extra-functional requirements (EFRs) are becoming more and more commonly used in software today. Real-time systems and safety-critical systems have been using EFRs for a long time. Today even entertainment systems and small embedded systems have many EFRs that have to be modelled and verified with analysis and testing. To perform analysis on EFRs, the software must be augmented with extra-functional properties (EFPs), describing how the software behaves. Examples of such properties are *execution time*, *memory consumption*, *reliability* etc.

Because of the intrinsically non-linear behaviour of software, it is often hard to make accurate predictions of EFPs. The problem is worsened in component-based development where components are kept free of context to facilitate reuse. EFPs consider all possible configurations in which a component can be used, which lower the accuracy of each specific configuration. To make analysis more accurate, and thereby systems more predictable, it is desired to have higher accuracy of the EFPs. This can be achieved by considering the context of the software.

The extra effort required for identifying, designing, analyzing and maintaining the context-information must not be greater than the benefits acquired from higher accuracy. Effort can be measured in many different ways; one natural way is to measure it in terms of economic benefit.

For many resource constrained systems, especially embedded systems, it is important to not use over dimensioned hardware due to high costs. Correct analysis and predictable systems are equally important.

The contribution of this paper is the discussion and proposal of techniques for increasing the accuracy of extra-functional properties in component-based development for embedded systems, while maintaining reusability of components. Thus within this context we mainly focus on two concerns:

- Methods for determining accuracy of EFPs.
- Designing components for increased accuracy and reuse.

The work in this paper is a continuation of ideas in [4].

The outline of the rest of this paper is as follows; in section 2 we discuss contexts in CBD. Different approaches for identifying contexts are discussed in section 3. In section 4 assumptions and research questions for future work is presented. Section 5 concludes the paper and future work is discussed.

2 Accuracy of Property Prediction

Components are elements of reuse, and should therefore be context independent. Hence, they must conform to a worst-case scenario for all possible contexts. By considering the context of a component it is possible to make the predictions more accurate.

2.1 Contexts and reuse

All software is executed within a context, i.e., CPU, collaborating software, run-time system, etc. By making assumptions about the context, prediction of EFPs is made more accurate.

Some of the main driving forces for component-based software engineering are reuse and third party composition. Both these artefacts significantly speed up development by using already developed and pre-tested components. To facilitate reuse and third party composition components are deployable on different platforms. Hence, we make a distinction between *deployment context* and *usage context*.The concepts and rationale for separating deployment and usage contexts are as follows:

Deployment context: By making assumptions about properties like hardware, composition and run-time system, components become hard to reuse, especially by third party. A component should be oblivious considering the deployment to not have a strong connection to one specific configuration. All components have an infinite number of deployment contexts since it includes, e.g., composition, stimuli from connected components and hardware. A deployment context consists of:

- *Composition*; how components depend on each other.
- *Usage profiles*; part of the composition, more specifically input values and probabilities of input values, depending on connected components see section 2.3
- *Configurational assumptions*; assumptions about properties like hardware, run-time system, scheduling and resource availability.

Usage context: is a parameterizable model of an EFP and each component has *one* usage context for every EFP. At deployment, a usage context is parameterized with values from its current deployment context. The usage context aims at increasing the accuracy of EFPs by considering deployment context, without reducing reusability. A component can be used in any deployment context regardless of usage context, because the usage context makes no assumptions about the deployment.

A usage context can be modelled with different models, from *dependent finite state machines* (DFSMs) [5] to simple lookup tables. The accuracy of the model determines the accuracy of the analysis. A usage context consists of:

- Parameterizable models of every EFP, considering *usage profile*, *composition* and *configuration* (deployment context).

2.2 Usage contexts

The accuracy of EFPs is crucial for the accuracy of the analysis. For instance, when performing real-time analysis, the accuracy of the EFPs worst-case and best-case execution times is determining hardware requirements. In this paper we will exemplify with the EFP *worst-case execution time* (WCET).

Example 1: Consider a case with a simple component, with one single input variable of the type integer. Lets assume whenever the integer value is lower or equal to 10, a part ”A” of a component is executed, and when the input is higher than 10, a part ”B” is executed (figure 2.[2]). Lets assume that part A has an execution time of 5 ms and part B has an execution time of 500 ms. By not considering contexts, the component will be assigned a worst-case execution time of 500ms and a best-case execution time of 5ms. The worst-case execution time, without consider context, will be 500ms.

In a static configuration, considering example 1, where path B will never be executed due to the usage of the component, the actual WCET will be 5ms; the predicted 500ms is thus very inaccurate. Context information is required to get accurate predictions and can be acquired by analyzing the usage context of the component.

In a dynamically scheduled system, considering example 1, where a few timing errors are acceptable, i.e., *soft real-time systems*, probabilities related to the execution of the paths A and B can be used for opportunistic scheduling, and effectively lower the WCET.

One priority requirement in resource constrained embedded systems is to use as little resources as possible while maintaining a predictable system. Most resource predictions assume worst and best-case scenarios; hence it is desired that the predicted properties are as close to the real-case as possible, as depicted in figure 1.

As described in example 1, we can see that a component that is only augmented with EFPs, without considering a usage context, has the possibility of being very inaccurate. We do not aim at lowering the WCET of a component, but we want the BCET and WCET to reflect the actual execution time as accurately as possible.

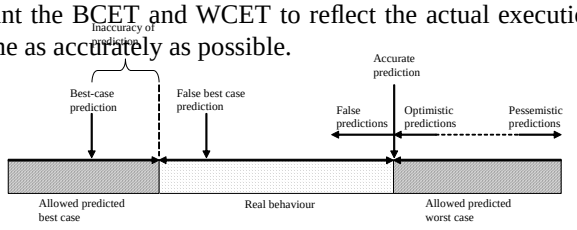

Figure 1. Accuracy of predicted best and worst case execution times

2.3 Probability of contexts

Since usage contexts depend on usage profiles, i.e., inputs, we determine all possible values for each usage context. Consider a very simple case with a usage context only depending on a usage profile with two booleans, b1 and b2. By building a lattice of all possible values, and augmenting each leaf with timing properties we can see interesting patterns. Each leaf is augmented with a WCET. Each node above the leafs are augmented with a WCET equal to the max of each connected leaf (see Fig. 2.[1]).

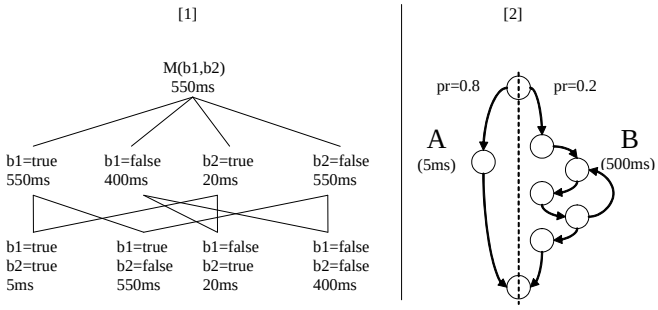

Figure 2. [1] Compositional context and [2] execution paths A and B, with WCET and probabilities

In figure 2.[1] we see that when b2 is true, the WCET is lower than any0 other case. This show that if pre-run time analysis can determine that b2 is true for certain situations, a lower WCET is used in a static schedule or similarly in dynamic scheduling. The lattice can be transformed to a table where lookups are made during run-time for assessing scheduling decisions.

An example of a trade-off can be seen in relation to table 1. There is a trade-off between minimizing the size of a table and keeping as much accuracy as possible. A small table will give us fast lookup and fast analysis, but lower accuracy, and vice versa.

Obviously, with a higher number of inputs, e.g., two 32-bit integers, there will be a state space explosion. However, often embedded systems are designed with a small number of modes in mind, and the inputs that control these modes are an example of suitable inputs to be modelled. The designer have to determine the impact an input has on the EFP, and decide which inputs should be modelled. Hence, it is up to the designer of the model to trade-off between effort and accuracy. Concretely, a usage profile is a naming convention of a variable that can be used for automatic analysis.

A usage profile consists of:

- A type, e.g., {range, enum or boolean}
- A natural ordering, e.g., $\leq, =, \neq$

	b1=false	b1=true	b1=n/a
b2=true	**20**	**5**	**20**
b2=false	400	550	550
b2=n/a	400	550	n/a

Table 1. WCET in usage contexts

- A probability distribution P

We return to *example 1* and augment the paths A and B with probabilities of execution by modelling describing probabilities of inputs.

Each input is modelled with a probability distribution which is derived from the deployment context. The purpose of the distribution is to augment the model (figure 2.[2]) with probabilities that are used for calculating a probability distribution P of a usage profile.

3 Identifying contexts

Optimally, a usage contexts have varying execution-time depending on the usage-profile. In figure 3 we assume a usage context for the EFPs WCET and BCET for a specific deployment context. We assume that component execution times vary with the usage-profile. The dots in figure 3 represent execution times that have been estimated from the component instance in a specific deployment context.

The estimation of an EFP in a specific deployment context can be achieved in many different ways. WCET can, e.g., be statically analyzed from the code or observed by monitoring the system.

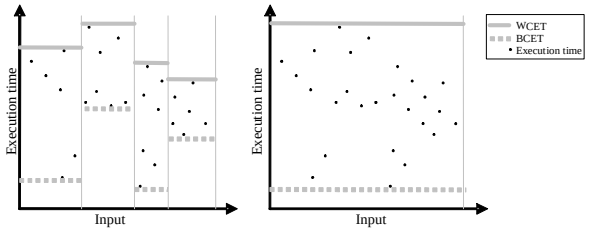

Figure 3. Context aware vs. context free WCET and BCET

There are two fundamentally different approaches to create usage contexts. One approach is to identify contexts by analyzing existing code considering execution paths and estimating probabilities considering usage-profiles. This method requires formal methods and code analyzers.

The other approach is to *create usage contexts* considering a deployment context, i.e., create the model from measured EFP values. Only usage-profiles have to be specified

to be considered. The usage contexts are not directly associated with semantic constructs but will rather be automatically analyzed with respect to the EFP values that have been estimated. Also probability distributions can be estimated from observing the system. Usage contexts can be optimized with respect to given trade-off conditions, e.g., as high accuracy as possible with as few contexts as possible. The accuracy in figure 3 could, e.g., be defined as the difference between BCET and WCET. The optimization would then simply be to minimize the area between the dashed and full lines in figure 3.

4 Constraining the problem

To facilitate future work we make explicit assumptions and try to identify hypotheses to be verified. Our main thesis is informally: "We can increase the accuracy of EFPs with little effort by considering usage contexts.".

Our assumptions are:

More contexts increase the accuracy. With higher granularity of analyzable execution paths the analysis can be made more accurate. A high number of contexts have the benefit of possible higher accuracy in static analysis and opportunistic reclaiming of resources during scheduling with methods like [2].

More contexts increase the effort. a high number of contexts have the drawback of higher effort design and identifying a high number of contexts. The overhead in terms of decisions during run-time for reclaiming resources also increases.

Design for reuse makes predictions more inaccurate In order to reuse components in different products and on different platforms, they must make as few assumptions as possible, which makes prediction of EFPs inaccurate.

Accurate analysis requires accurate EFPs Accurate analysis requires accurate knowledge of the system.

Furthermore, we state the hypotheses:

1. *A few usage contexts* is enough to get *useful accuracy*.
2. *Many usage contexts* leads to *too high effort*.

A few contexts is a finite, low number of contexts; typically greater than 2 and smaller than 100.

Useful accuracy is of course entirely dependent on the application. Although, higher accuracy than the accuracy of one context is always expected. It is not always required to have maximum possible accuracy. Thus, there is a trade-off between a low number of contexts, i.e., low effort, and a high accuracy.

Too high effort must be related to some quantitative property like economics. A too high effort is one that is higher than the benefit.

5 Future Work

In future work we will try to verify our thesis with experiments. The EFP we will try and verify is initially WCET. We have started to implement a framework to generate contexts to verify our ideas.

Our desired results from the experiments are that we would like to see a high (more than linear) increase in accuracy in the first contexts and at the same time see a linear (or lower) increase in effort with increasing number of contexts.

We will integrate the ideas in Save component technology [3]

References

[1] I. Crnkovic and M. Larsson. *Building Reliable Component-Based Software Systems*. ISBN 1-58053-327-2. Artech House, 2002.

[2] J. Fredriksson and K. S. Mikael Å kerholm, Radu Dobrin. Attaining Flexible Real-Time Systems by Bringing Together Component Technologies and Real-Time Systems Theory. In *Proceedings of the 29th Euromicro Conference, Component Based Software Engineering Track Belek, Turkey)*, September 2003.

[3] H. Hansson, M. Åkerholm, I. Crnkovic, and M. Törngren. SaveCCM - a Component Model for Safety-Critical Real-Time Systems. In *Proceedings of 30th Euromicro Conference, Special Session Component Models for Dependable Systems*, September 2004.

[4] A. Möller, I. Peake, M. Nolin, J. Fredriksson, and H. Schmidt. Component-based context-dependent hybrid property prediction. In *ERCIM - Workshop on Dependable Software Intensive Embedded systems*, Porto, Portugal, September 2005. ERCIM.

[5] H. W. Schmidt, B. J. Krmer, I. Poernomo, and R. Reussner. Predictable component architectures using dependent finite state machines. Technical report, Lecture Notes in Computer Science, 2941:310-324, 2004.

[6] C. Szyperski. *Component Software - Beyond Object-Oriented Programming*. ISBN 0-201-74572-0. Addison-Wesley, 1998.

[7] R. van Ommering, F. van der Linden, and J. Kramer. The koala component model for consumer electronics software. In *IEEE Computer*, pages 78–85. IEEE, March 2000.