Predicting Execution Time for Variable Behaviour Embedded Real-Time Components

Johan Fredriksson, Thomas Nolte, Mikael Nolin
Dept. of Computer Science and Electronics
Mälardalen University, Västerås, Sweden

Heinz Schmidt
Dept. of Comp. Science and Software Eng.
Monash University, Victoria, Australia

Abstract

Embedded systems for vehicle control critically depend on efficient and reliable control software, together with 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.

In this paper we present a method, which increases the accuracy of execution time predictions for embedded components without lowering reusability of the components. For assessing the correct timing behaviour, the method classifies run types by their time in addition to their probability.

1 Introduction

Components are elements of reuse, and should therefore be context independent. Hence, they must conform to a worst-case scenario for all possible contexts, leading to possibly inaccurate predictions for each specific usage. However, by considering the context of a component it is possible to make predictions more *accurate*, and thus use less resources. Introducing a context aware execution time representation allows for a more efficient use of resources, which in turn contribute to a lower product cost.

The embedded systems industry is under competitive pressure to continually shorten its time-to-market and increase product differentiation at the same time. Component software and its reuse accelerates this competitive process. As a result however, (i) embedded systems become increasingly software-intensive, (ii) costs shift from hardware to software, and, (iii) individual components integrate increasing functionality over different projects and reuse cycles.

Integrating more functions into a single component gives rise to increasingly varying behaviour, some variants only invoked in a particular deployment context, some only as part of adaptive behaviour triggered by context-awareness or deployment-specific configuration parameters. Properties of the component such as time and reliability are variable and context-dependent and the variance may be large. Software components in embedded systems make it extremely hard to predict system properties and hence guarantee quality of the systems and the services they provide. At the same time, componentization has been the key to structured design processes with predictable properties in many engineering domains. For software, in particular, a context-free characterisation of component properties is inadequate for accurate predictions.

1.1 Related Work

The work in this paper is a continuation of ideas presented in [4] and [10]. In the SWEET `wcet` analysis tool suite [2, 5, 9], abstract interpretation is used to predict `wcet` with specific input intervals. However, this approach requires source code, and reusability is lowered because analysis is required for every new usage. Hybrid methods may overcome some deficiencies by combining static and dynamic methods. In, e.g., [8] methods are used to generate benchmarks to determine `wcet` of basic blocks of code. In [1] the program is divided into different parts each associated with a probability distribution for execution times.

2 Contexts and reuse

Some of the main driving forces for component-based software engineering are reuse and third party composition. Both speed up development significantly by using already developed and pre-tested components. To facilitate reuse, and third party composition components are deployable on different configurations. Often only a subset of the total functionality of a component is used in different reuse configurations. To be able to use the component in any of the different supported configurations, without reanalyzing the component, it must be predicted for the worst possible scenario. This of course may lead to inaccurate predictions and poor utilization of resources.

We introduce a context aware execution-time representation to achieve more accurate predictions. To support this representation we classify component behaviour and interfaces.

Components consists of *behaviours*, *provided interface* and *required interface*. The provided interface is associated with a *usage profile* that describes the configuration. In [11] the fourth author has shown how usage-profile probabilities compose over component architectures and give rise to probabilistic predictions of reliability. There the reliability annotations are associated with required interfaces and the usage-profile probabilities with provided interfaces. In the current paper we take a similar approach, except that the requirement annotations are related to `wcet` and that these times are computed from observations rather than from formal execution models. The required interface is associated with a *control context* that describes the data and control flow based on usage probabilities.

The behaviour is restricted by the usage profile, and the control context is restricted by the behaviour. We can model the control context as a function of the usage profile. Thus we get a compositional model where the behaviour of a component is restricted by a function of the usage profile of the preceding components.

The behaviour of the component is directly related to the execution time of the component, thus we specify the extra-functional properties *worst-case execution time* (`wcet`) and *best-case execution time* (`bcet`) for each component.

3 Usage profile

In software reliability theory (see e.g. Musa [7]) usage profiles are probability distributions for so-called run types. Probabilities are calculated using long program runs and large numbers of them. To guarantee statistical properties (for example relative independence of input order), these are divided up into short runs, for example cycles in periodic real-time systems, transaction in transaction processing systems, and if necessary sampled. Run types are then defined as sets of similar runs based on input classes or other context parameters.

In the "real" physical world, distinct run types exist and are often engineered into systems, for example, as *modes of operation*. We hypothesize that run types are significant discriminators of `wcet` and can be utilised for more accurate `wcet` modelling. Thus we define a *usage profile* as $U = \langle I_0, ..., I_{n-1} \rangle$, where the $I_i (0 \leq i < n)$ are input variables, each with a small domain D_i of values of a given type, and a probability distribution $P_i : D_i \rightarrow [0,1]$ for the occurrence of these values in the input. We assume that these variables (and hence their distributions) are chosen to be statistically independent and either have small domains naturally or model discretized partitions of

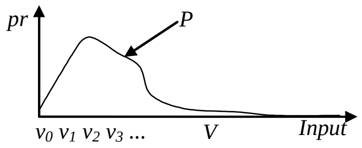

Figure 1. Input variable I

real input variables. (See Fig. 1 for an illustration of these concepts). The input domain M is then defined as $M := D_0 \times \ldots \times D_{n-1}$. The probability distributions $P_i (0 \leq i < n)$ extend uniquely to a probability distribution $P : M \rightarrow [0,1]$ on the input domain, defined by $P(x_0, \ldots, x_{n-1}) = P_0(x_0) \times \cdots \times P_{n-1}(x_{n-1})$.

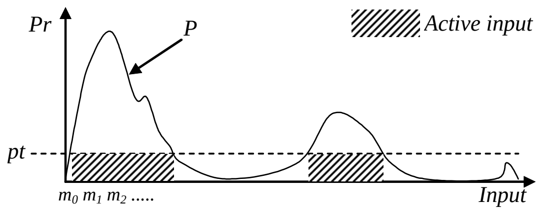

Figure 2. Usage profile

Furthermore we assume $0 \leq pt < 1$ is a given probability threshold to ignore low probability inputs (and consequently later their times). This will permit predictions of the form "with 0.99 probability `wcet`$< 500ms$." Inputs over the threshold are called *active* and the ratio of *active inputs* over *all inputs* is called the usage-profile *utilization*. See also Fig. 2.

4 Context-aware prediction

Components are reused in different products and different contexts. A different usage profile can substantially change the behaviour of a component. To predict the execution time of a complex component with high accuracy, today components must be reanalyzed for every new usage profile – a costly activity. Also, it is not certain that source code is available for components as they may be delivered by sub contractors. In this case analyses become even more costly.

Our method overcomes the problem by analyzing the execution times and their probability as a function of the component input. We assume that execution time varies with different inputs and their associated run types. The input domain is divided into discrete segments that we call "bins", as shown in Fig. 3.

Similar work includes [6], which uses abstract interpretation and eliminates dead code not used in the given context. This however is not possible for black-box components since source code is required.

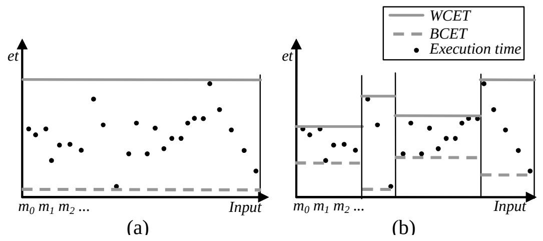

Figure 3. One bin versus four bins

Given a specific usage profile for a component, it is straightforward to assess which bins are active, i.e., contain an *active input*. The purpose is to remove the "bins", i.e., the execution times, that are not active in a specific usage. Thus the difference between `wcet` and `bcet` for the entire component can be lowered heuristically.

For each input value, there exist a corresponding `wcet` and `bcet` for the component, corresponding to the highest and lowest execution time possible through all inputs in the bin as shown in Fig. 3. Each bin i is associated with a $\texttt{wcet}^i$ and $\texttt{bcet}^i$, corresponding to the highest and lowest execution time within the bin.

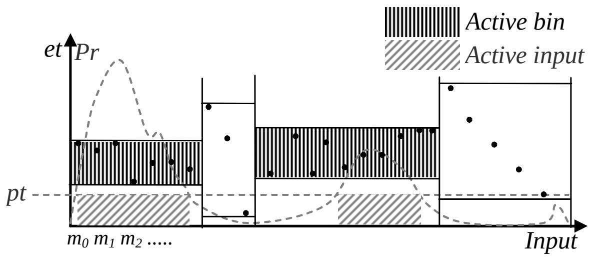

Figure 4. Usage profile applied to the bins

A usage profile is applied, and the bins that contain an active input are considered for the `wcet` and `bcet` for the entire component, as shown in Fig. 4, where Fig. 2 is overlaid on top of Fig. 3.

5 Prediction framework

The prediction framework is a set of models for calculating the accuracy of the prediction. Each bin can represent an area based on the number of inputs in the bin and the difference between the $\texttt{wcet}^i$ and $\texttt{bcet}^i$.

After dividing the input domain into bins, the bins that are inactive in a specific usage profile are removed. By removing bins, the number of candidates for the component `wcet` and `bcet` is lowered. Consequently the accuracy of the execution time may be increased.

An important factor for the success of the method is how the input domain is segmented into bins. It is desirable to have a behaviour with minimal variation in `wcet`. The finest-resolution binning would associate a single bin with each single value and thus bring variation to 0: however at very high modelling and computational cost. Instead, we strive to simplify the model such that no exhaustive analysis is required for reuse of components. Therefore we associate both variation and the number of bins with a penalty and aim to minimise both at the same time. The rational of the penalizing bin creation is that it is harder to reuse and analyze a component with a high number of bins, than a component with a low number of bins. Thus, we want as high accuracy as possible with as few bins as possible.

The implementation is based on genetic algorithms (GA) [3], where each gene represents an input range, i.e., the size of a bin. Each chromosome represents the entire system with all inputs assigned to bins. Each system produced by the GA is evaluated by the framework, and is given a fitness value dependent on the total area of the bins.

The fitness function that guides the genetic algorithm is defined to minimize the sum of the areas of all bins while promoting few bins above many bins. For each simulation the method divides the component into bins and compares the areas and incrementally finds new divisions with lower areas. When the GA has found a good division of bins, a number of random usage profile are assigned to the component to evaluate the performance of the method.

6 Ongoing Evaluation

Currently we are working on further formalizing the method and the framework. We are also evaluating the performance of the method in the implemented framework by simulating random components and usage profiles. Some initial findings of the evaluation are briefly discussed later in the remainder of this section.

The framework is evaluated with respect to performance, relation between number of bins and accuracy, and, performance with respect to probability threshold.

The components are simulated with random timing behaviour, i.e., the execution time is a random function of the input. usage profiles are created with a set of random variables, where each variable has a random type, range and a probability distribution conforming to one of normal, binomial or Poisson distributions. The distributions have been chosen based on the assumption that many physical processes follow these distributions.

Random execution-time generation is chosen based on the following assumptions: real software usually has a behaviour resulting in more distinct patterns than random; if accuracy can be increased even for uniformly random distributed execution times, it is reasonable to believe that the performance gain is even higher for real software following non-random patterns; real software usually has some peaks and valleys that can be isolated to achieve better performance.

Initial results indicate a higher improvement for low utilizations. For random distribution when few inputs are used, i.e., a low usage-profile utilization, high improvements, over 50%, are shown. However, improvements up to 20% are achieved for higher utilizations. Still these results are very preliminary, and further evaluation is required in order to show results with high confidence.

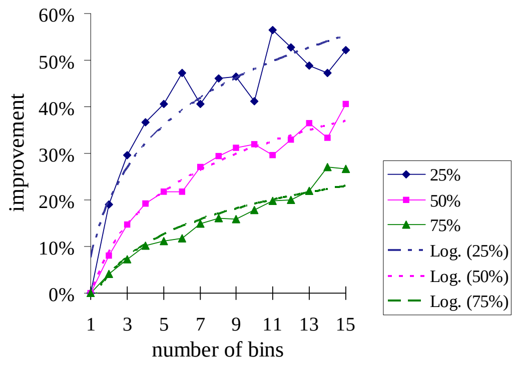

Figure 5. Initial results

Fig. 5 show the improvement of accuracy depending on the number of bins. The y-axis in is the improvement of accuracy between a context-free (one bin) and a context-sensitive (several bins) such that:

$$improvement = \frac{Area_{contextfree} - Area_{contextsensitive}}{Area_{contextfree}}$$

The x-axis is the number of bins the GA has chosen is the best division. The improvement is presented with respect to different usage-profile utilizations and different number of inputs. The fitted curves (dashed) are fitted as polynomials with the least square method.

The curve with highest improvement has a utilization of 25%, the middle curve a utilization of 50%, and finally the lowest improvement is shown for a utilization of 75%.

7 Summary

In this paper, we have shown how usage profiles suitably represent context information to increase the accuracy of timing predictions in reusable components, without decreasing reusability. Our method combines probability theory from reliability engineering with genetic algorithms and promises to show a significant increase in accuracy of `wcet` prediction. We reported initial findings of an ongoing evaluation.

References

[1] G. Bernat, A. Colin, and S. Petters. pWCET, a Tool for Probabilistic WCET Analysis of Real-Time Systems. In *WCET*, pages 21–38, 2003.

[2] A. Ermedahl. *A Modular Tool Architecture for Worst-Case Execution Time Analysis*. PhD thesis, Uppsala University, Dept. of Information Technology, Uppsala University, Sweden, June 2003.

[3] C. M. Fonseca. and P. J. Flemming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary computation*, 3(1), 1995.

[4] J. Fredriksson. Increasing accuracy of property predictions for embedded real-time components. In *18th Euromicro Conference on Real-Time Systems (ECRTS 06), WiP*, July 2006.

[5] J. Gustafsson, A. Ermedahl, and B. Lisper. Towards a flow analysis for embedded system C programs. In *$10^t h$ Intl. Workshop on Object-Oriented Real-Time Dependable Systems*, Sedona, USA, Feb. 2005.

[6] M.-L. Ji, J. Wang, S. Li, and Z.-C. Qi. Automated wcet analysis based on program modes. In *AST'06, Shanghai, China*. ACM, MAY 2006.

[7] A. I. John D. Musa and K. Okumoto. *Software Reliability - Measurement, prediction, application*. McGraw-Hill, New York, 1987.

[8] R. Kirner, P. Puschner, and I. Wenzel. Measurement-Based Worst-Case Execution Time Analysis using Automatic Test-Data Generation. In *In Proceedings of the 4^{th} Euromicro International Workshop on WCET Analysis*, June 2004. Catania, Italy.

[9] Mälardalen University. WCET project homepage, http://www.mrtc.mdh.se/projects/wcet. Mälardalen University, 2006.

[10] 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.

[11] R. H. Reussner, H. W. Schmidt, and I. Poernomo. Reliability prediction for component-based software architectures. *Journal of Systems and Software*, 66(3):241–252, 2003.