Worst-Case Execution Time Clustering for Software Components

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

Abstract

For component-based systems, classical techniques for WCET-estimation produce unacceptable overestimations of the WCET. This is because software components have more general behavior in order to support reuse. Existing tools and methods for component-based software engineering (CBSE) do not yet adequately consider reusable analyses.

We present a method that allows different WCETs to be associated with subsets of the component behavior by clustering WCETs with respect to behavior. The method is intended to be used for facilitating reusable WCET analysis for reusable software components. We illustrate our technique and demonstrate its potential in achieving tight WCET-estimates for components with rich behavior.

1 Introduction

In this paper we present a method that allows reuse of components with rich behavior (and its implied high resource usage) in contexts where not all functionality of the components is needed. For these contexts it is imperative to be able to analytically reduce the estimated resource usage in order to achieve tight predictions of high quality. Thus increasing accuracy of predictions.

The work presented in this paper is intended to be used to facilitate reusable WCET analysis for software components, e.g., in the framework presented in [1].

Components are often reused over product boundaries, i.e., they are part of product lines and it is desirable to use the same component without re-analysis or recompilation. However, different products offer different contexts or usage of components; thus a component used in a, e.g., truck may use different parts of the component compared to the same component used in a caterpillar. Using a context insensitive WCET analysis may be very inaccurate compared to the WCET^{truck} or WCET^{caterpillar} (WCETs of the truck and caterpillar respectively), leading to a poor utilization of the system resources because of large differences between predicted behavior and actual behavior.

Resource constraints and predictability requirements are especially common in many embedded-systems sectors, such as automotive, robotics and other types of computer controlled equipment. Because of the intrinsically non-linear behavior of software, it is often hard to make accurate predictions of extra functional properties (EFPs). The problem is worsened in component-based development where components are kept free of context to facilitate reuse. To make analysis more accurate, and thereby systems more predictable, it is desirable to have high accuracy of the predictions. This can be achieved by considering the context in which the software is used.

The contribution of this paper is a method for increasing the accuracy of WCET by clustering WCETs with respect to usage. We use binary search heuristics to efficiently creating clusters of similar WCETs. We describe and formalize the method, and exemplify with an illustrative example. Finally we use a simple academic case study and create clusters on two components.

The outline of the rest of this paper is as follows; in Section 2 we discuss related works. Usage scenarios are discussed in Section 3. In Section 4 component WCET analysis and the WCET clustering method are presented. In Section 5 we evaluate the method. In Section 6 we discuss the applicability of the method, and finally, Section 7 concludes the paper and future work is discussed.

2 Related work

Static WCET analysis is the only safe method for estimating WCETs for hard real-time systems [2]. However, traditional static WCET analysis does not consider usage. Software components designed for reuse are often more general compared to application specific code, leading to that parts of the component are only used in specific usages; in turn leading to greater variance of execution times. For component-based systems, where reuse is in focus, it is desirable to not being forced to reanalyze components for each usage, at least within the same platform.

One approach to solve similar problems is parametric WCET. This has been proposed by many researchers within

*contact author: johan.fredriksson@mdh.se

the WCET community. There are very few parametric WCET methods developed; although, in a MSc thesis [3, 4] such a method has been developed and tested with the aiT tool [5]. However, the focus of this work is not reusable WCET analysis, and reanalysis is required for different usages.

In [6] each basic block of a program is analyzed with respect to execution times and probability distributions of the execution times are derived. This method is, in comparision to our method, based on measurements. In [7] a framework has been developed that considers the usage of a system; however, neither software components nor reuse is considered. In [8] the source code is divided in modes depending on input, and only modes that are used in a given context is analyzed. In [9] a framework for probabilistic WCET with static analysis is presented. The probabilities are related to the probability of possible values of external and internal variables. All mentioned methods have the drawback of requiring reanalysis for every new usage.

Recent case-studies show that it is important to consider mode- and context-dependent WCET estimates when analyzing real sized industrial software systems [10, 11].

There are several WCET tools that support assertions and conditions to make the WCET tighter, e.g., aiT [5], RapiTime [12], Bound-t [13] and SWEET [14].

3 Usage scenario

In the "real" physical world, distinct modes exist and are often engineered into systems, for example, as *modes of operation*. We hypothesize that modes are significant discriminators of WCET and can be utilized for more accurate WCET modeling.

In [15] usage scenarios are probability distributions for so-called modes. Probabilities are estimated using large number of long program runs. To guarantee statistical properties (for example relative independence of input order), the program runs are divided into short runs, for example cycles in periodic real-time systems, transaction in transaction processing systems, and if necessary sampled. Modes are then defined as sets of similar runs based on input classes or other context parameters.

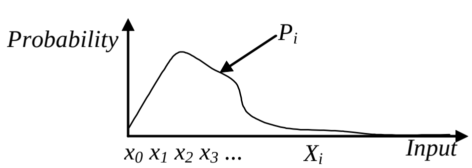

Figure 1. Input variable I

Thus we define a *usage scenario* as $U = \langle X_0, ..., X_{n-1} \rangle$, where the $X_i (0 \leq i < n)$ are input variables, each with bounds on values, a given type, and a probability distribution $P_i : X_i \rightarrow [0, 1]$ 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 discredited partitions of real input variables. (See Figure 1 for an illustration of these concepts). The input domain M is then defined as $M := X_0 \times \cdots \times X_{n-1}$. 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 $P(x_0, \ldots, x_{n-1}) = P_0(x_0) \times \cdots \times P_{n-1}(x_{n-1})$.

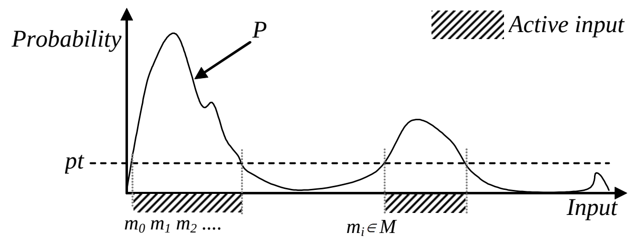

Figure 2. Usage scenario

Furthermore we assume that $0 \leq pt < 1$ is a given probability threshold for ignoring 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-scenario *utilization*. See also Figure 2 for an illustration of the concept.

4 Component WCET analysis

Components are reused in different products and different contexts. A different usage profile can substantially change the behavior of a component. To predict the execution time of a complex component with high accuracy, components must today be reanalyzed for every new usage profile – a very costly activity. Furthermore, it is not certain that the source code is available for components as they may be delivered by sub contractors. In this case analyses become even more costly [16].

Our method overcomes the problem by analyzing the execution times and their probability as a function of the input of the component. We assume that execution time varies with different inputs and their associated modes.

We define an input domain $\mathbf{I}$ for a set of input variables $\{X_0, X_1, \ldots, X_{n-1}\}$ as $\mathbf{I} = X_0 \times X_1 \times \cdots \times X_{n-1}$. Each element q in $\mathbf{I}$ is associated with an execution time $ET(q) \in \mathbf{W}$, where all execution times of the component are represented in the set $\mathbf{W}$. The longest execution time $\max(\mathbf{W}) =$ WCET^{abs} is the absolute WCET. A traditional static WCET tool will only find an estimate WCET$^{est} \geq$ WCET^{abs}; however, we want to find the WCET for a specific usage. Because $\mathbf{I}$ often is very large, we can not perform WCET analysis for every element in $\mathbf{I}$ (every possible usage), instead we perform static WCET analysis with annotations on the input parameters, and perform a number of systematic runs with different bounds on the input parameters. When WCET analysis is performed with restrictions on the input parameters, not *all* input elements are considered, but rather a set of clusters $\{\mathbf{D}_l | \mathbf{D}_l \subseteq \mathbf{I}\}$, such that $\mathbf{D}_0 \oplus \mathbf{D}_1 \oplus \cdots \oplus \mathbf{D}_{n-1} = \mathbf{I}$. Thus, a cluster is a subset of all possible inputs, and a WCET tool can produce a WCET considering only that subset of inputs. Each cluster $\mathbf{D}_l$ is analyzed and associated with two execution times $et_l^{max} = \max(ET(d))_{d \in \mathbf{D}_l}$ and $et_l^{min} = \min(ET(d))_{d \in \mathbf{D}_l}$. The time et_l^{max} is the result of running the WCET tool with the inputs represented in $\mathbf{D}_l$ with respect to WCET. The time et_l^{min} is the result of running the WCET tool with the inputs represented in $\mathbf{D}_l$ with respect to best-case execution time (BCET).

As with all static WCET analyses all execution time estimates are safe over-estimations.

4.1 Clustering WCETs

To handle the size of the input domain $\mathbf{I}$ clusters need to be expressed with bounds or other operators, where each bound is associated with a WCET. It is often unfeasible to make a list of all inputs that are associated with one cluster; furthermore, WCET-tools often uses bounds to restrict the inputs. With the mathematical operators $\{\leq, >\}$ ranges of inputs can be expressed. The clusters $\mathbf{D}_l$ should be chosen in such a way that similar execution times are grouped and can be expressed as restrictions on the inputs. A challenge is to find the right clusters $\mathbf{D}_l$ such that accuracy of execution times become high.

4.2 Finding clusters

When the input domain $\mathbf{I}$ is too large to perform WCET analysis for every single input combination it is necessary to divide $\mathbf{I}$ into clusters of input combinations and analyze each cluster with respect to execution time. As the relation between inputs and WCET is not known a priori, the input space must be searched to find clusters such that all input combinations within the cluster produces similar execution times. In order to find such clusters it is necessary to have a way of evaluating clusters.

Theoretically, each single input combination has only one fixed execution-time. The difference between et_l^{max} and et_l^{min} of a cluster $\mathbf{D}_l$ shows the greatest difference between two execution times within the cluster. This in turn is an indicator of how similar the execution times are in the cluster. The sum of the difference between et_l^{max} and et_l^{min} of all clusters $\sum_l (et_l^{max} - et_l^{min})$ should be minimized to get the highest accuracy. In the extreme, each cluster contains one element; a good solution is a trade-off between acceptable difference and max number of clusters. If the difference between et_l^{max} and et_l^{min} of the cluster is larger than the required accuracy the cluster is not evaluated as a good cluster. Thus, the allowed difference between et_l^{max} and et_l^{min} of the cluster depends on the required accuracy of the cluster.

It is desired to create as few clusters as possible and yet acquire as high accuracy as possible. Clusters are effectively annotations (input restrictions) to a WCET-tool. Hence, we need methods to find annotations for WCET-tools.

To find accurate clusters with the least effort we propose a binary tree search approach, recursively dividing the input space into two clusters until the required accuracy has been found for all branches. Finding the clusters is a blind search problem. The only data initially known is the longest and shortest execution time for the entire search space (the WCET and BCET). This lack of knowledge depends on the nature of most WCET-tools, they provide a WCET and a BCET given a program and annotations; we want a large number of execution times considering different input combinations. The more the input space is divided the more data become available. There are several possible approaches to solve blind search problems, where binary search, simulated annealing and evolutionary search, are a few possible candidates.

Consider a simple example (Figure 3) with a function `foo` having two input variables x and y, where x can take the values $[0..9]$ and y can take the values $[0..4]$. All possible execution times given this simple example are summarized in Table 1. In this small example there are only 50 possible input combinations, and it is trivial to make an exhaustive search to find all combinations that give the same execution time. In a larger example, this is not possible. We have chosen such a simple example to simplify the visualization of the method.

One set of values produce the worst-case execution time WCET. In the example in Figure 3 the WCET is produced by inputs represented by the first row in Table 1. All other input combinations lead to lower execution times. Consider an example where the usage scenario defines $x = \{3..6\}$ and $y = \{3..4\}$, the WCET will never occur. A WCET topology of the example is shown in Figure 4. For the case of a 2-dimensional input domain, the WCET topology is visible in an execution time matrix as shown in Figure 5.

The initial knowledge of the matrix is only the highest and lowest values (Figure 6.a). Since the knowledge of the execution times is limited we need a search method to localize areas with the similar execution times. One approach is to make a binary search for similar WCETs. In

```
// typeA:[0..9], typeB:[0..4]

typeA foo(typeA x, typeB y)
{
    if(x > 2 && y < 3)
        x=bar1(x,y);          // 30ms

    if(x > 2 && y > 2)
        return x-y;           // 1ms

    if(x < 5 && y > 0)
        x=bar2(y,x);          // 40ms

    if(x == y)
        return bar3(0,0);     // 20ms


    return bar4(x+y,x-y);     // 100ms
}
```

Figure 3. Example code

#i	x	y	cond.	et_l^{max}	et_l^{min}
4	$[3,4]$	$[1,2]$		170	170
2	$[3,4]$	$[0]$		130	130
12	$[0,2]$	$[0,4]$	$x \neq y$	140	140
15	$[5,9]$	$[0,2]$		130	130
2	$[1,2]$	$[1,2]$	$x = y$	60	60
1	$[0]$	$[0]$		20	20
14	$[3,9]$	$[3,4]$		1	1

Table 1. Clustered WCETs with respect to the example code shown in Figure 3. #i is the number of input combinations. x and y are the limitations on the inputs. Cond is a logical condition on the inputs and $\mathrm{et}_l^{\max}$ and $\mathrm{et}_l^{\min}$ are the longest and shortest execution times produced by the inputs.

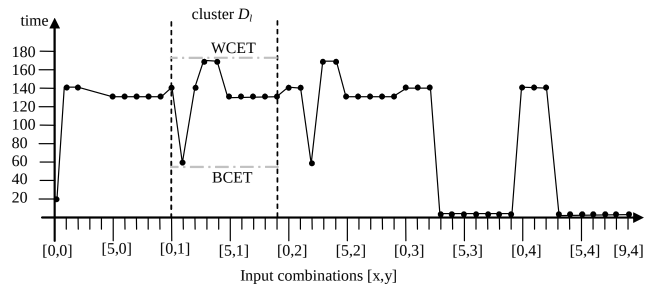

Figure 4. WCET topology with respect to the example code shown in Figure 3.

x \ y	0	1	2	3	4
0	20	140	140	140	140
1	140	60	140	140	140
2	140	140	60	140	140
3	130	170	170	1	1
4	130	170	170	1	1
5	130	130	130	1	1
6	130	130	130	1	1
7	130	130	130	1	1
8	130	130	130	1	1
9	130	130	130	1	1

Figure 5. Matrix of the inputs $\{$x,y$\}$ with corresponding execution times with respect to the example code shown in Figure 3. The dotted line shows the cluster D_l as shown in Figure 4.

Figure 6 binary search is shown, dividing the search space into smaller and smaller clusters until the desired accuracy has been reached. The accuracy is defined as the distance between the highest and lowest values et_l^{max} and et_l^{min} for each cluster. In Figure 6, clusters that have reached their desired accuracy are marked with "*".

If the input space is divided into too few clusters accuracy will be lost; consider the extreme case of only using one cluster (all inputs), then the accuracy will be the same as standard WCET analysis. Due to large input spaces it is often infeasible to make an exhaustive search; therefore, even when the input domain is divided into a relatively large number of clusters it is still important how these are chosen to maximize accuracy. Since the analysis is supposed to be reused, the effort of the analysis itself is of less concern.

5 Evaluation

We have performed a small evaluation with the SWEET WCET-tool [14]. SWEET has an annotation language to give restrictions on input parameters. Hence, it is very suitable for the approach presented in this paper. The annotations are described by the clusters.

Two components from an academic adaptive cruise controller (ACC) have been analyzed, "loggerOutput" and "SpeedControl". Both components have three input variables. We have a performed a guided binary search on both components. The guidance consisted of limitations on the input variables to 8 values for each input; these limitations were chosen based on the source code. The result of the guidance was an input domain of $8^3 = 512$ input combinations on each of the components. It required 12 clusters of

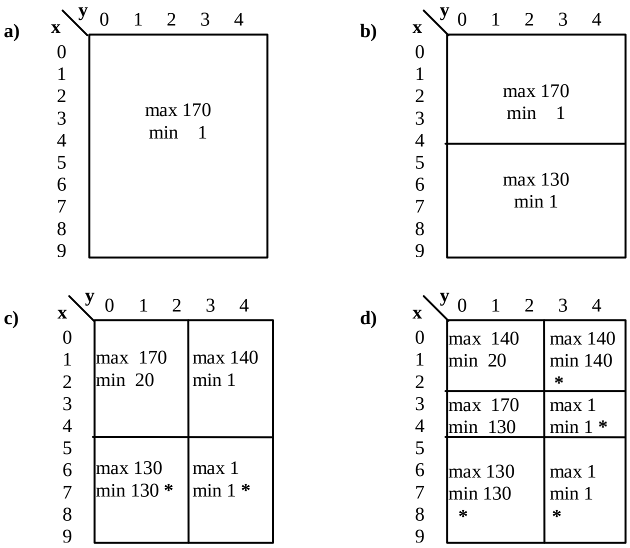

Figure 6. Binary search with respect to the input matrix shown in Figure 5. An '*' indicates that a cell does not need to be further divided. Max indicates the WCET reported by the tool given the annotations, and min the BCET dito.

the input domain of the "LoggerOutput" component to partition the execution times and produce 3 WCET expressions called *contracts*. The execution times were more scattered in the "SpeedLimit" component and it required 25 clusters to isolate all execution times into three contracts, The final contracts derived from the clusters for the "LoggerOutput" and "SpeedLimit" components are shown in Tables 2 and 3.

#	Expression	WCET
1	$i_2 \leq 0 \wedge i_3 \leq 0$	239
2	$(i_2 > 0 \wedge i_3 \leq 0) \vee (i_2 \leq 0 \wedge i_3 > 0)$	433
3	$other$	627

Table 2. LoggerOutput component "contract" from 12 clusters

The derived contracts are used with a usage scenario on the input parameters. Depending on the usage the contracts will give the WCET corresponding to the usage.

We see that for many usages we get substantially lower WCETs for both components. Using a traditional usage independent analysis would produce much to pessemistic WCET for many usage scenarios.

#	Expression	WCET
1	$i_1 \leq 0$	105
2	$i_2 > 0 \wedge ((i_1 = 0 \wedge i_3 < 0) \vee (i_1 > 0 \wedge i_3 \geq 0))$	384
3	$other$	263

Table 3. SpeedLimit component "contract" from 25 clusters

6 Applicability

The method described in this paper is a general clustering method that is well suited for creating contract based WCETs for components. Each cluster can also be augmented with more information, e.g., scheduling parameters and energy consumption. In this way clusters can be created with respect to several parameters and trade-offs between them can be made.

Furthermore, the proposed method is useful for both hard and soft real-time systems. In this paper we have only described the application for hard real-time systems.

The methods as described in this paper indirectly perform an exhaustive WCET analysis because all input combinations are represented. This will result in safe overestimations and the "real" WCET is guaranteed to be included in the analysis.

For soft real-time systems, a number of input combinations (not clusters) can be analyzed with respect to execution-time and clusters can be created through, e.g., the least square method.

The focus of the method is still to create tight and accurate *reusable* WCET estimations through expressing the WCET as usage parameterized contracts.

6.1 Hardware effects

It should be noted that the contracts specified for the clusters only consider input data limits. The timing of the code in the cluster will also be dependant on the hardware upon which the code is executed and where in memory the code is located. Assuming that a simple 4-, 8- or 16-bit CPU is used, which is common in a large segment of the embedded domain, and that the code is forced to reside in and access memory areas with the same timing properties as assumed in the WCET analysis, the WCET estimates derived should also be valid in the new context. However, if a more advanced CPU is used, maybe with a cache or some other performance enhancing features, and/or if the compiler and linker change the code structure, and/or if some other hardware timing properties are changed, the derived component WCET estimates should be used with caution. Thus, in the latter case the contract for a component might also need to include information upon the hardware, compiler and linker configuration. This is something not yet considered in our work.

7 Conclusions and future work

Component-based software engineering (CBSE) is a promising development method to reduce time-to-market, reduce development costs, and to increase software quality. One main characteristic of CBSE that enable these benefits is its facilitation of component reuse, i.e., the same component can be used in different contexts. Unfortunately for resource constrained systems, or systems where high degree of predictability is needed, reusable components with rich behavior increase resource consumption and decrease predictability.

In this paper we have presented a method for clustering WCETs with respect to behavior for reusable software components. The purpose of the method is to associate different WCETs with subsets of the component behavior to achieve tight WCET estimates. The presented method is intended to be used for facilitating reusable WCET analysis for reusable software components as presented in, e.g., [1]. We have illustrated the method and demonstrated its potential in a small case study.

Future work includes case studies on large components to evaluate the feasibility of the approach. Also case studies on industrial code is planned to evaluate the industrial appropriateness of the proposed method. We also plan to investigate augmentation of clusters with additional parameters, e.g., scheduling parameters.

References

[1] Fredriksson, J., Nolte, T., Nolin, M., Schmidt, H.: Contract-based reusable worst-case execution time estimate. Technical report, Mälardalen Real-Time Centre (2007) (submitted to RTCSA'07).

[2] Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The worst-case execution time problem — overview of methods and survey of tools. Accepted for publication in ACM Transactions on Programming Languages and Systems (2007)

[3] Altmeyer, S.: Parametric wcet analysis, parametric framework and parametric path analysis. Master's thesis, Saarland University, Department of Computer Science (2006)

[4] Humbert, C.: Parametric wcet analysis, parameter analysis and parametric loop analysis. Master's thesis, Saarland University, Department of Computer Science (2006)

[5] aiT: (ait execution time analyzer) Absint: http://www.absint.com/ait/.

[6] Bernat, G., Colin, A., Petters, S.: pWCET, a Tool for Probabilistic WCET Analysis of Real-Time Systems. In: WCET. (2003) 21–38

[7] Lee, J.I., Park, S.H., Bang, H.J., Kim, T.H., Cha, S.D.: A hybrid framework of worst-case execution time analysis for real-time embedded system software. In: Aerospace, 2005 IEEE Conference, ieee (2005) 1–10

[8] Ji, M.L., Wang, J., Li, S., Qi, Z.C.: Automated wcet analysis based on program modes. In: AST'06, Shanghai, China, ACM (2006)

[9] David, L., Puaut, I.: Static determination of probabilistic execution times. In: ECRTS '04: Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS'04), Washington, DC, USA, IEEE Computer Society (2004) 223–230

[10] Sehlberg, D., Ermedahl, A., Gustafsson, J., Lisper, B., Wiegratz, S.: Static wcet analysis of real-time task-oriented code in vehicle control systems. In: 2nd International Symposium on Leveraging Applications of Formal Methods (ISOLA'06), Paphos, Cyprus (2006)

[11] Byhlin, S., Ermedahl, A., Gustafsson, J., Lisper, B.: Applying static wcet analysis to automotive communication software. In: 17th Euromicro Conference of Real-Time Systems, (ECRTS05), Mallorca, Spain (2005)

[12] RapiTime: (Rapitime execution time analyzer) Rapita Systems: http://www.rapitasystems.com/.

[13] Bound-t: (Bound-t execution time analyzer) Tidorum Ltd: http://www.tidorum.fi/bound-t/.

[14] SWEET: (Swedish execution time tool) SWEET: http://www.mrtc.mdh.se/projects/wcet/.

[15] John D. Musa, A.I., Okumoto, K.: Software Reliability - Measurement, prediction, application. McGraw-Hill, New York (1987)

[16] Korel, B.: Black-box understanding of cots components. In: IWPC '99: Proceedings of the 7th International Workshop on Program Comprehension, Washington, DC, USA, IEEE Computer Society (1999) 92