Contract-Based Reusable Worst-Case Execution Time Estimate

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

Abstract

We present a contract-based technique to achieve reuse of known worst-case execution times (WCET) in conjunction with reuse of software components. For resource constrained systems, or systems where high degree of predictability is needed, classical techniques for WCET-estimation will result in unacceptable overestimation of the execution-time of reusable software components with rich behavior. Our technique allows different WCETs to be associated with subsets of the component behavior. The appropriate WCET for any usage context of the component is selected by means of component contracts over the input domain. In a case-study we illustrate our technique and demonstrate its potential in achieving tight WCET-estimates for reusable components with rich behavior.

1 Introduction

Component-based software engineering (CBSE) is a promising development method to reduce time-to-market, reduce development cost, and increase software quality. One main characteristic of CBSE that enable these benefits is its facilitation of component reuse. However, for resource constrained systems, or systems where high degree of predictability is needed, reusable components with rich behavior increase resource consumption and decrease predictability.

Resource constraints and predictability requirements are especially common in many embedded-systems sectors, such as automotive, robotics and other types of computer controlled equipment. Hence, to unleash the full potential of CBSE in these domains we need techniques that allow 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 its imperative to be able to analytically reduce the estimated resource usage in order to achieve tight predictions of high quality.

In this paper we reduce the pessimism in the estimate of the WCET of a component. Execution time is one of the most critical resources in many embedded systems, and it is also one of the most difficult to obtain good estimates for. We present a contract-based technique to achieve reuse of known worst-case execution times (WCETs) in conjunction with reuse of software components. Our technique allows different WCETs to be associated with subsets of the component behavior. The appropriate WCET for any usage context of the component is selected by means of predicates and contracts over the input domain.

By using our proposed technique we show that it is possible to make tighter predictions for different usages on software components without reanalyzing the component for every new usage.

1.1 Outline

The remainder of this paper is organized as follows. In Section 2 we discuss related work. In Section 3 we describe the problem and introduce a general component model. Section 4 describe transformation from components to tasks and the relation between transformation and WCET. In Section 5 we describe the WCET method. In Section 6 we validate the work, and finally in Section 7 we conclude the paper and discuss future work.

2 Related work

Timing is important for many systems, especially in the embedded systems domain. Today static WCET analysis is the predominant technology for acquiring worst-case execution times, but both dynamic and hybrid prediction methods are gaining increasing interest. A well known problem is that the WCET often only occur in a special mode or context, if it ever occur.

Today, most companies do not use static WCET tools but instead guess the WCET by performing a set of dynamic WCET measurements and then multiplying the worst observed execution time by some factor to get a safe overestimation. This method is often much more pessimistic than
3 Problem description

To support reuse, components must be developed as general enough in order to be reused in several different contexts and usage scenarios, i.e., components should be context and usage independent. Reuse brings many benefits, but it also brings increased efforts, e.g., more code, greater variance in execution time and higher resource utilization. In desktop systems where resources are abundant and predictability is of less concern, these efforts do not usually imply problems. However, resource constrained embedded real-time systems require both efficient resource usage and high analyzability and predictability. To utilize the benefits of CBSE in the embedded real-time domain the issues brought from static usage independent analysis scales with the number of components in the task, leading to a potentially very high prediction error in the task. This has shown to be a problem especially in real-time systems where jitter is desired to be kept low.

3.1 System model / Context dependent performance analysis

In CBSE applications are built from several software components. In component technologies for desktop systems, e.g., COM, CORBA and .NET, components are usually heavy weight in terms of dynamic behavior. Components for embedded systems are usually much lighter due to the need for efficient analysis and predictability and often have execution semantics like Run-To-Completion [13]. Typical examples of component technologies for embedded real-time systems are Rubus CM [14], Autocomp [15], SaveCCM [16] and PECOS[17].

The execution semantics is decided by the scheduling policy and the operating system. To be able to schedule the components they must be transformed into tasks conforming to the specified rules of the scheduler and properties must be set, e.g., period, priority, deadline etc.

3.1.1 Component characteristics

In this section we describe characteristics for a general component model that is applicable to a large set of embedded component models. The component interaction model used throughout this paper is a pipe-and-filter model. The pipe-and-filter interaction model is commonly used within the embedded systems domain.

Component $c_i$ is described with a tuple $(P_i, R_i, Q_i, wcet_{abs}^{i}, wcet_{use}^{i}, k_i)$, where $P_i$ is the provided interface, which is a set of $\{p_i^0, p_i^1, ..., p_i^{n-1}\}$ input variables and $R_i$ is the required interface, i.e., a set $\{r_i^0, r_i^1, ..., r_i^{n-1}\}$ of output variables. $Q_i$ represents the period. The parameter $wcet_{abs}^{i}$ is the estimated WCET for the component. $wcet_{use}^{i}$ is the usage dependent WCET for the component and $k_i$ is a contract as a function with respect to a usage that returns the estimated WCET of the component $k_i : f(U) \rightarrow wcet_{abs}^{i}$.

3.1.2 Task characteristics

The task model specifies the organization of entities in the component model into tasks. During the transformation from component model to run-time model, properties like schedulability and response-time constraints must be considered in order to ensure the correctness of the final system. Components only interact through explicit interfaces; hence
tasks do not synchronize outside the component model. The task model is for evaluating schedulability and other properties of a system, and is similar to standard task graphs as used in scheduling theory.

**Task** $\tau_i$ is a tuple $\langle Z_i, T_i, C_{i}^{abs}, C_{i}^{use} \rangle$ where $Z_i$ is an ordered set of components. Components within the same task are executed in sequence of the order of $Z$ and with the same priority as the task. $T_i$ is the period or minimum inter arrival time of the task. The parameters $C_{i}^{abs}$ and $C_{i}^{use}$ are the estimated WCET and the usage dependent WCET respectively. The $C_{i}^{abs}$, $C_{i}^{use}$ and period ($T_i$) are deduced from the components in $Z_i$. The $C_{i}^{abs}$ is the sum of all the estimated WCETs for all components allocated to the task and the $C_{i}^{use}$ is the sum of all usage dependent WCETs for all components allocated to the task. Hence, for a task $\tau_i$, the parameters $C_{i}^{abs}$ and $C_{i}^{use}$ are calculated with (1) and (2).

$$
C_{i}^{abs} = \sum_{\forall i (C_i \in Z_n)} (wcet_{i}^{abs}) \tag{1}
$$

$$
C_{i}^{use} = \sum_{\forall i (C_i \in Z_n)} (wcet_{i}^{use}) \tag{2}
$$

The error between the estimated and the usage dependent execution time is the sum of the difference between the estimated and usage dependent WCET of all components allocated to the task, as given by (3).

$$
C_{i}^{error} = \sum_{\forall (C_i \in Z_n)} (wcet_{i}^{abs} - wcet_{i}^{use}) \tag{3}
$$

4 Mapping components to tasks

A problem in current component based embedded software development practices is the mapping of component services to run-time threads (tasks) [18]. Because of the real-time requirements on most embedded systems, it is vital that the mapping considers temporal attributes, such as worst case execution time (WCET), deadline (D) and period time (T). In a system with many small component services, the overhead due to context switches is quite high. Embedded real-time systems consist of periodic and aperiodic events, often with end-to-end timing requirements. Periodic events can often be coordinated and executed by the same task, while preserving temporal constraints. Hence, it is easy to understand that there can be profits from grouping several component services into one task.

Many component-based systems today use one-to-one allocations between design-time components and real-time tasks. Finding allocations that co-allocate several components to one real-time task leads to better memory and CPU usage. However, the one-to-one allocations have the benefit of being highly analyzable, which is often a strong requirement in embedded systems, especially in embedded real-time systems that handle time-critical functions such as engine control and breaking systems. Hence, components need to be allocated to tasks in such a way that temporal requirements are met, and resource usage is minimized.

In [19] we have shown that transformations from components to tasks potentially give high benefits in terms of increased resource efficiency. We have also shown that a tighter WCET estimations (more laxity of the timing constraints) produces a higher number of feasible mappings, and hence a greater chance to find a better mapping compared to one-to-one mappings.

An allocation from components to a task must be evaluated considering schedulability. Both component to task allocation and scheduling are complex problems and different approaches are used. Simulated annealing and genetic algorithms are examples of algorithms frequently used for optimization problems. In [19] we present and evaluate a framework that utilizes genetic algorithms for solving component to task mappings.

A common obstacle to combine predictability with correctly dimensioned resources is the inaccuracy of the system analysis. Real-time analysis is based on worst-case assumptions, and the composition of worst-cases make the system impractically oversized and under utilized[20].

Even though WCET estimations become more accurate and hybrid prediction [21] helps making WCET calculations more precise, there is still no good way to accurately predict the WCET for a reusable component. There is an error $C_{i}^{error}$ corresponding to the difference of the predicted behavior compared to the real behavior. As WCET predictions need to consider the worst possible execution time for all possible behaviors they are inherently inaccurate for reusable components with varying usage and rich behavior.

When each component is mapped to a single task the error of each task is the same as the error of the component. When several components are mapped to one task, the error scales with the number of components, and the error can become quite large. The total system error stays the same but greater errors of individual tasks have a greater impact on properties like input jitter and output jitter, just to mention a few.

5 WCET analysis for reuse

For components that are reused in different systems it is today often not very meaningful to perform WCET analysis. This is because traditional WCET analysis considers only one specific usage of a system, and the usage can vary a lot between different configurations. To support reuse of WCET predictions we need support for WCET analysis of
different usage.

A component that is designed for reuse has to be general and free from context dependencies. By designing the component specifically for one particular context or usage it can be analyzed and predicted with high accuracy, but not always reused. In order for general reusable components to be predicted with higher accuracy we need new methods and frameworks. When the usage is not known at design time of a component, it is necessary to augment the component with information that can be used to accurately predict the worst-case execution time for a specific usage. The WCET can differ a lot between different uses of the same component. It is necessary to perform WCET analysis for every element in a WCET considering only that subset of inputs. Each subset of all possible inputs, and a WCET tool can produce such that

\[ \max(ET(d))_{d \in D} \]

is the execution time associated with input \( d \). A traditional static WCET tool will only find the worst-case execution time for a specific usage. The Reusable WCET analysis can be divided in three steps, namely:

**Component WCET analysis** Analyzing the WCET of the component considering many different general usage scenarios (inputs).

**Clustering WCETs** Clustering inputs that lead to similar execution times.

**Component contracts** Creating a contract that define the clustered inputs.

### 5.1 Component WCET analysis

The input domain \( I_i \) for the input variables \( \{p_i^0, p_i^1, \ldots, p_i^{n-1}\} \in \mathbb{P_i} \) in the provided interface \( \mathbb{P_i} \) of component \( c_i \) is defined as \( I_i = \text{dom}(p_i^0) \times \text{dom}(p_i^1) \times \ldots \times \text{dom}(p_i^{n-1}) \) where \( \text{dom}(p_i^j) \) is the value domain of the \( j^{th} \) input variable in the provided interface of the \( i^{th} \) component. Each element \( q \) in \( I_i \) is associated with an execution time \( ET(q) \in \mathbb{W_i} \), where \( ET(q) \) is the execution time associated with input \( q \) and all execution times are represented in the set \( \mathbb{W_i} \). The longest execution time \( \max(\mathbb{W_i}) = WCET^{abs} \) is the absolute WCET. A traditional static WCET tool will only find the one element that is associated with the highest execution time, i.e., WCET^{abs}; however, we want to find the WCET for a specific usage. Because \( I_i \) is often very large, we can not perform WCET analysis for every element in \( I_i \) (every possible usage), instead we perform static WCET analysis with annotations on the input parameters, and performs a high number of runs with different bounds on all input parameters. When WCET analysis is performed with restrictions on the input parameters, not all input elements are analyzed, but rather a set of clusters \( \{D_i^1, D_i^2, \ldots, D_i^k \} \subseteq I_i \}, \) such that \( D_i^1 \oplus D_i^2 \oplus \ldots \oplus D_i^{k-1} = I_i \). 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 \( D_i^k \) is analyzed and associated with an execution time \( et_i^k = \max(ET(d))_{d \in D_i^k} \). The time \( et_i^k \) is the result of running the WCET tool with the inputs represented in \( D_i^k \).

Consider a simple example (Figure 1) with a function \( \text{foo} \) with two input variables \( x \) and \( y \), where \( \text{dom}(x) = \{0..5\} \) and \( \text{dom}(y) = \{0..100\} \). One set of values produce the WCET^{abs}, e.g., when \( x = 5 \) and \( y = 100 \). All other input combinations leads to lower execution times. Consider an example where the usage scenario defines that \( x = \{0..2\} \) and \( y = \{10..20\} \), then WCET^{abs} will never occur. We only want to consider execution times for the specific usage scenario, without reanalyzing the component. If the combination \( \{x, y\} = \{5, 100\} \) is defined as a cluster, this cluster will not be affected by the usage, and that cluster and its associated execution time can be ignored, effectively making the analysis tighter.

```c
void foo(int x, int y)
{
    if(x==5 & y==100)
        { ... /* wctet = 200 ns */ }
    if(x==3 & y==45)
        { ... /* wctet = 40 ns */ }
    if(...) /* all other wctet < 200 ns */
}
```

**Figure 1. Example code**

In this paper we do not consider hardware effects from, e.g., loading a component at different positions in the memory, this is outside the scope of this paper.

### 5.2 Clustering WCETs

To handle the size of the domain \( I_i \) clusters need to be expressed with bounds or other operators, where each bound is associated with a WCET. It is often unfeasible to make an unordered list of all inputs that are associated with one cluster; furthermore, WCET-tools often use bounds to restrict the inputs. With the mathematical operators \( \{\leq, >\} \) ranges of inputs can be expressed. To use these operators each input \( p_i^j \) need a (natural) ordering. Consider Figure 2 where only one input \( p_i^0 \) is depicted with an ordering \( 0 \leq p_i^0 < 100 \); execution times (et) that are neighboring inputs are clustered. If \( n \) inputs are used the input domain becomes \( n \)-dimensional.

The clusters \( D_i^1 \) should be chosen in such a way that “peaks” and “valleys” in the WCET topology are captured as depicted in Figure 3 (Figure 3 only depicts one input variable). A challenge is to find the right clusters \( D_i^1 \) such that accuracy of execution times is maintained.
be produced in several ways, e.g., markov models [22] or operational profiles [23], usually with the help of a domain expert. Independent of method, the usage scenario is important for accurate predictions; the more accurately the usage scenario can be assessed, the better results can be produced by the reusable analysis.

U is a usage scenario consisting of a set of input bounds \( B : [a, b] \) on an input variable \( p_i^j \) such that \( a \leq p_i^j < b \). Several bounds may target the same input variable. The usage scenario \( U \) consists of bounds on a set of input variables \( p_0^0, ..., p_0^{n-1} \). For usage scenario \( U = \{ [a_0, b_0], [a_1, b_1], ..., [a_{n-1}, b_{n-1}] \} = \{ B_0, B_1, ..., B_{n-1} \} \) such that \( \{ a_0 \leq p_i^j < b_0, a_1 \leq p_i^j < b_1, ..., a_{n-1} \leq p_i^m < b_{n-1} \} \). The usage scenario is also associated with a probability distribution \( P : U \rightarrow [0, 1] \) for the occurrence of the values in \( \{ B_0 \times B_1 \times ... \times B_{n-1} \} \).

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 utilization.

\( A \) is the set of active input combinations that have a probability \( \{ a \in \{ B_0 \times B_1 \times ... \times B_{n-1} \} \land Pr(a) > pt \} \) that is greater than the probability threshold \( pt \).

5.4 Component contracts

Each cluster can be transformed into a predicate with respect to the usage scenario \( U \). The predicate tests if the active inputs \( A \) of the usage scenario is “inside” the clusters, i.e., \( \forall \{ et_i \} \subseteq D_i \). For all clusters \( W = \forall \{ et_i \} \subseteq A \} \) that have at least one element in \( A \), the WCET for the component \( c_i \) with the usage scenario \( U \) is the longest execution time associated with any cluster, i.e., max(\( W \)). Thus the component contract is a function \( f(U) \rightarrow WCET_i^{use} \), where \( f(U) : \max(W) \).

The probability distribution of the usage scenario \( U \) is used for calculating the probability of the occurrence of the inputs in a cluster \( D_i \), i.e., each cluster is associated with a probability for the specific usage scenario \( U \). If the proba-

5.3 Usage scenarios

The reusable execution-time analysis is parameterized with a usage scenario for the system. Usage scenarios can
bility of a cluster is lower than the probability threshold $p_t$ the cluster can be ignored, and that execution time is disregarded in the contract.

5.5 Composable WCETs

Each cluster is associated with a set of possible outputs. Abstract interpretation can be used to make a safe over approximation of limitations on outputs given limitations on inputs by analyzing possible values of the output variables. Each component produces output given the input such that the required interface $R_i$ of component $c_i$ is a function of the input $f(P_i) \rightarrow R_i$. By adding this information to the predicates the approach is composable since one component will automatically give a component usage scenario to the next connected component. SWEET [5] is one tool that can produce restrictions on the output given restrictions of the input.

6 Evaluation

We performed an evaluation according to the proposed WCET techniques. We are using an academic system for evaluating the approach. The system is a simple adaptive cruise control developed with the SaveCCM component model as depicted in Figure 5. The cruise control is built from four components, one switch and one assembly. For a detailed description of SaveCCM and the adaptive cruise controller we refer to [16]. The ACC is seen as a ready to use application that can be used in different products.

The WCET acquired by traditional static WCET analysis is compared to the WCET obtained by the usage contract-based reusable estimate. The tool used for both the usage independent and the usage dependent analysis is SWEET [5] which is developed at Mälardalen University.

As an example we assume that a car company uses the ACC in two different car models, one high end car and one low end car. In the high end car all features are enabled and have a greater range on $ACC\ Max\ Speed$ ($AMS$). The low end car have several features disabled. The $Road\ Signs\ Enabled$ is disabled, and consequently the $Road\ Sign\ Speed$ is also disabled. Furthermore, the distance control is disabled and the $Distance$ is always set to 0.

Both usage scenarios are assuming a uniformly distributed probability distribution over all inputs. The probability threshold $p_t$ is set to 0, meaning that no input combinations are culled.

<table>
<thead>
<tr>
<th>RSE</th>
<th>AMS</th>
<th>RSS</th>
<th>D</th>
<th>CS</th>
<th>AE</th>
<th>BPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>250</td>
<td>0..130</td>
<td>0..2k</td>
<td>0..250</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Table 1. High-end car usage scenario $U_1$

<table>
<thead>
<tr>
<th>RSE</th>
<th>AMS</th>
<th>RSS</th>
<th>D</th>
<th>CS</th>
<th>AE</th>
<th>BPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>130</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Table 2. Low-end car usage scenario $U_2$

6.1 Example usage scenarios

In Tables 1 and 2 we define the usage scenarios $U_1$ and $U_2$ by specifying possible values for the different inputs. The inputs are $Road\ Signs\ Enabled$ ($RSE$), $ACC\ Max\ speed$ ($AMS$), $Road\ Sign\ Speed$ ($RSS$), $Distance$ ($D$), $Current\ Speed$ ($CS$), $ACC\ Enabled$ ($AE$) and $BrakePedal\ Used$ ($BPU$), as depicted in Figure 5.

The high-end car has all features enabled and have a greater range on $ACC\ Max\ Speed$ ($AMS$).

The low end car have several features disabled. The $Road\ Signs\ Enabled$ is disabled, and consequently the $Road\ Sign\ Speed$ is also disabled. Furthermore, the distance control is disabled and the $Distance$ is always set to 0.

6.2 Evaluation results

The results of the analysis with the different usage scenarios applied are presented in Table 3, where $ud(U_1)$ represents the usage dependent analysis with usage scenario $1$,
i.e., the high-end car and \( \text{ud}(U_3) \) represents the usage dependent analysis with usage scenario 2, i.e., the low-end car.

The evaluation, from which all presented measurements are collected, is performed using the SWEET WCET tool suite [5] for the ARM9 machine model and with the options infeasible paths, excluding pairs and min-max node count turned on.

To create the clusters we are using a guided linear division of the input space as we possess the knowledge of the source code and the fact that the components are small. The resulting number of clusters for each component is presented in Table 4. Due to space limitations we do not present the bounds of each input variable of each cluster.

<table>
<thead>
<tr>
<th>Comp.</th>
<th>ui</th>
<th>ud</th>
<th>ud(U_1)</th>
<th>ud(U_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpeedLimit</td>
<td>384</td>
<td>384-105</td>
<td>384</td>
<td>105</td>
</tr>
<tr>
<td>ObjectRecog.</td>
<td>301</td>
<td>301-220</td>
<td>301</td>
<td>220</td>
</tr>
<tr>
<td>BrakeAssist</td>
<td>274</td>
<td>274-88</td>
<td>191</td>
<td>88</td>
</tr>
<tr>
<td>Logger.</td>
<td>627</td>
<td>627-239</td>
<td>433</td>
<td>433</td>
</tr>
<tr>
<td>Calc Speed.</td>
<td>369</td>
<td>369-283</td>
<td>369</td>
<td>291</td>
</tr>
<tr>
<td>Calc Dist.</td>
<td>706</td>
<td>706-188</td>
<td>706</td>
<td>435</td>
</tr>
<tr>
<td>Upd. Speed.</td>
<td>85</td>
<td>85</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>Upd. Dist.</td>
<td>181</td>
<td>181-85</td>
<td>181</td>
<td>85</td>
</tr>
</tbody>
</table>

Table 3. WCETs according to the methods usage independent (ui) and usage dependent (ud) and usage dependent with usage profiles ud(U_1) and ud(U_2)

We see that the two usage scenarios produce different WCETs by activating different clusters. In the first case of usage scenario \( U_1 \) most features are used and the \( \text{ud}(U_1) \) WCET is close to the \( \text{ui} \) WCET. In the second case \( \text{ud}(U_2) \) several features are disabled, and consequently fewer clusters are activated and the WCET is lower.

<table>
<thead>
<tr>
<th>Comp.</th>
<th># clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpeedLimit</td>
<td>3</td>
</tr>
<tr>
<td>ObjectRecog.</td>
<td>3</td>
</tr>
<tr>
<td>BrakeAssist</td>
<td>3</td>
</tr>
<tr>
<td>Logger.</td>
<td>3</td>
</tr>
<tr>
<td>Calc Speed.</td>
<td>4</td>
</tr>
<tr>
<td>Calc Dist.</td>
<td>5</td>
</tr>
<tr>
<td>Upd. Speed.</td>
<td>1</td>
</tr>
<tr>
<td>Upd. Dist.</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4. Number of clusters per component

The WCET presented in Table 5 is a composed WCET of all components in the ACC controller. In this specific example the WCET is reduced by a factor of 40%. It is difficult to say anything about the general improvement of the proposed technique. However, we believe that we have shown its potential. It is also worth mentioning that the WCET produced by our proposed technique is still a safe overestimation. Hence, contract-based usage dependent WCET analysis should be of great interest in many resource constrained industrial applications.

7 Conclusions and future work

Componentization has been successful in facilitating structured design processes with predictable properties in many engineering domains. The embedded software systems industry is competing with decreasing time to market and product differentiation, both leading to an increasing dependence on software required to be flexible enough for rapid reuse, extension and adaptation of system functions. As a result embedded systems become increasingly software-intensive and individual software components integrate an increasing amount of functionality over different projects and reuse cycles.

Integrating more functions into a single component gives rise to increasingly varying behavior. Properties of the component such as time and reliability are variable and usage dependent, and the variance may be large. For software, in particular, a usage independent characterization of component properties is inadequate for accurately predicting the properties of the composite system constructed using these components.

In this paper we propose an effective reusable contract-based WCET estimation technique that provides tighter WCET estimates by clustering input combinations that produce similar WCETs. We have shown the effectiveness of the proposed method in an evaluation of an academic adaptive cruise controller example.

As future work we plan to investigate further constraints on input data and the impact of the proposed method in an industrial case-study.

References

[1] Lindgren, M., Hansson, H., Thane, H.: Using measurements to derive the worst-case execution time. In:


