14th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing, March 28-31, 2011. Newport Beach, CA, USA. Copyright by IEEE

Hierarchical Composition of Parametric WCET in a Component Based Approach

Thomas Leveque, Etienne Borde, Amine Marref, and Jan Carlson
Mälardalen Real-Time Research Centre,
Mälardalen University, Sweden
Email: thomas.leveque@mdh.se, etienne.borde@mdh.se, amine.marref@mdh.se, jan.carlson@mdh.se

***Abstract*—Worst Case Execution Time (WCET) computation is crucial to the overall timing analysis of real-time embedded systems. Facing the ever increasing complexity of such systems, techniques dedicated to WCET analysis can take advantage of Component Based Software Engineering (CBSE) by decomposing a difficult problem into smaller pieces, easier to analyse. To achieve this objective, the corresponding analysis results have to be composed to provide timing guarantees on the whole system. In this paper, we express the WCET of a component as a formula, allowing to represent its different computational modes. We then propose a Model Driven Engineering (MDE) approach that derives parametric WCET for composite components from parametric WCET of their subcomponents. This approach gives more accurate WCET estimates than naïve additive compositional analysis by taking into account usage context of components. However, analysis scalability concerns lead us to consider a trade-off between precision and scalability. This trade-off can be specified in the model. The composition of WCET estimations is automated and produces the parametric WCET expression of the composite component under analysis. This approach has been integrated in PRIDE[1].**

I. Introduction

Designed to control physical systems, functionalities provided by embedded systems have to be performed within predefined deadlines. In order to ensure that such functionalities meet their timing requirements, the well known schedulability analysis theory relies on the notion of worst case execution time (WCET). Existing WCET estimation techniques are mainly based on static analysis and/or timing measurements. Using static analysis, WCET is computed by analysing the software and using the timing properties of the hardware platform it is executed on. Using measurements techniques, the software is executed on the target platform, and different execution times are measured depending on its execution context. Yet, both techniques present important limitations when it comes to their usability [5]. On one hand, the static estimation requires to consider all the different possible execution paths of a system in order to produce an accurate upper bound on its WCET. This analysis becomes all the more difficult since embedded systems become more and more complex. Because of this scalability issue, static WCET estimation produces a safe but pessimistic estimation of the WCET: the analyser uses pessimistic approximations, like for instance not rejecting infeasible execution paths. On the other hand, the measurement based techniques are unable to ensure that there is indeed no execution path of the software that leads to a higher execution time than the reported WCET.

In this paper, we present a model driven engineering (MDE) approach that helps in managing the trade-off between precision and scalability. In order to improve the precision of static WCET estimations, we rely on a parametric estimation of WCET [9]. In this paper, the parametric WCET of a functionality is the WCET associated to value ranges for the input data of this functionality. This static estimation technique combines both the data flow and control flow analysis in order to focus on the WCET estimation of feasible execution paths. We integrate this technique in a hierarchical component-based model in order to improve the scalability of static WCET estimations: assuming we know the parametric estimation of primitive components (components that encapsulate code but are not themselves composed of subcomponents) thanks to techniques such as [2], we produce the parametric WCET estimation of an enclosing composite component. From this result, we also produce an executable binary file that implements a test-case of the obtained parametric WCET. The software designer can thus compare the formal analysis result with the measured execution time in order to validate the parametric WCET.

Besides this MDE approach, we present in this paper a use-case from the automotive domain, as well as the results we obtained when experimenting our approach on this use-case.

The paper is organised as follows: Section II describes the problem we address. Section III gives an overview of our approach, and section IV introduces the use case we used to evaluate this approach. Our contribution is presented in more detail in sections V, VI and VII. Finally, we give in section VIII our experimental results and conclude this paper in section IX.

II. Motivations And Challenges

When it comes to timing analysis of embedded systems, WCET analysis is the only safe method for ensuring that functionalities of such a system will always be executed within a given deadline [13]. In this section, we present existing techniques dedicated to the estimation of the WCET of a software application. Methods for estimating the WCET of a software application are usually divided into two different parts: timing measurement and static analysis.

[1]PRIDE is the integrated development environment for the ProCom component model: http://www.idt.mdh.se/pride

However, existing methods in both parts suffer important limitations when it comes estimating the WCET of a complex application [5]. Using measurement based techniques, complexity of software applications impedes to guarantee that the set of considered test cases will produce the longest execution time of the software application. On the other hand, static analysis suffers scalability problems, leading to over estimated WCET [5], [13]. In order to tackle these issues, parametric WCET evaluation techniques, as well as hybrid methods, have been investigated. Parametric WCET enables to limit the set of feasible execution paths by investigating the dependencies between control flow and data flow [3], [9]. The computed WCET is thus expressed as a formula over the input variables of software system. Hybrid methods consist of combining static analysis and measurement to improve the precision of the estimated WCET [1], [7]. Finally, it is important to consider that timing measurement is a crucial aspect of an industrial WCET estimation process, for which test-case generation can be very helpful [7].

The observations presented above led us to address the following problem: "Considering the increasing complexity of embedded systems, how can we improve WCET estimation techniques in terms of precision and scalability?". To answer this question, we propose a MDE approach that combines parametric WCET techniques with a hierarchical component-based approach. On one hand, compared to [11], parametric evaluation improves the accuracy of the WCET. On the other hand, using a hierarchical component-based model, we divide the estimation problem into smaller and easier to manage pieces. CBSE is traditionally used in software engineering to ease the design of complex software applications by dividing them into independently manageable pieces. Our approach aims at benefiting from this decomposition not only for the design purpose, but also for helping in the analysis. The main problem, considering this approach, is to allow compositionality of the analysis results: considering we analysed all the subcomponents of a composite component, is it possible to deduce the parametric WCET of this composite?

In order to take advantage of both CBSE and parametric WCET estimations, we have to answer the following questions:

A. What are the pre-requisites for the compositionality of WCET estimations?
B. How to automate the composition of WCET estimations?
C. How to parametrize the trade-off between precision and scalability of the WCET analysis?

III. Approach

In this section, we present our approach that aims at answering the questions stated above. Each of the following subsections is dedicated to one of these questions.

A. Compositionality of Parametric WCET Estimations

The evaluation of WCET in a component-based approach has already been studied, considering the problem of reusing the analysis results in different usage contexts [4]. Our approach differs from the above by focusing on the hierarchical compositionality of the WCET estimations. In our approach, we consider that the parametric WCET of primitive components have been obtained using one of the techniques presented in the previous section. Considering the expected simplicity of those primitive components, the scalability and accuracy issues related to those methods should be mitigated.

Compositionality of WCET in a hierarchical development process has been recently explored [10]. Contrasting to our approach, this work discusses compositionality using tasks as the composition units, and without taking advantage of techniques based on parametric WCET estimation. As a consequence, the considered scope of compositionality is different.

In order to compose WCET analysis results, we need to focus on the composition model, thus on the component model. Although considering a different scope, [10] gives prerequisites on the compositionality of WCET analysis:

- The timing of a task should not be affected by the other tasks running in the system.
- The WCET of a composition of tasks should be the sum of the WCETs of those tasks.
- Execution time of tasks should be invariable.

Our approach relies on other requirements, mainly due to the usage of parametric WCET in a general purpose CBSE approach. In the remainder of this subsection, we present in more details the prerequisites of our MDE approach for the evaluation of WCET.

1) Data Flow Pre- and Post-conditions: The resulting output data values of the execution of a component can be expressed as a value range function from ranges of its input data values. The data pre-conditions expressions are logical expressions containing value comparison between input port values and constants. The data post-conditions expressions are logical expressions containing value comparison between output port values, input port values and constants. In addition, when an output data port of a component A is connected to an input data port of a component B, the parametric WCET of the execution of A and B can be computed while taking into account that the output data range of A become the input data range of B.

2) Explicit modelling of Control and Data Flow: The complete control and data flow corresponding to a composite component can entirely be deduced from (i) the semantics of the components interface, and (ii) the specification of the connections between those interfaces.

3) Influence of input data: The different parametric WCETs of a component can be expressed thanks to computational modes (see Definition 1).

Definition 1: The computational modes of a component are the abstract subsets of its execution paths, represented by a set of input data values for which the considered component exhibits identical properties. A parametric WCET of a component is thus a computational mode in which the component executes its functionalities in a time bounded by a single WCET value.

4) Conservative Semantics in a Hierarchical Design: Finally, our approach is fully recursive if we add the requirement that a composite component must have exactly the same semantics as a primitive component. Considering this hypothesis, if one can derive the parametric WCET of a composite from the parametric WCET of its primitive subcomponents, then this composite component can also be used to compute the parametric WCET of the components it might be enclosed in.

B. Automated WCET Analysis

Figure 1 summarizes the general approach we propose in order to improve the scalability of WCET estimation by combining parametric WCET and CBSE. As illustrated in this figure, composite component parametric WCET computation is divided in three main phases.

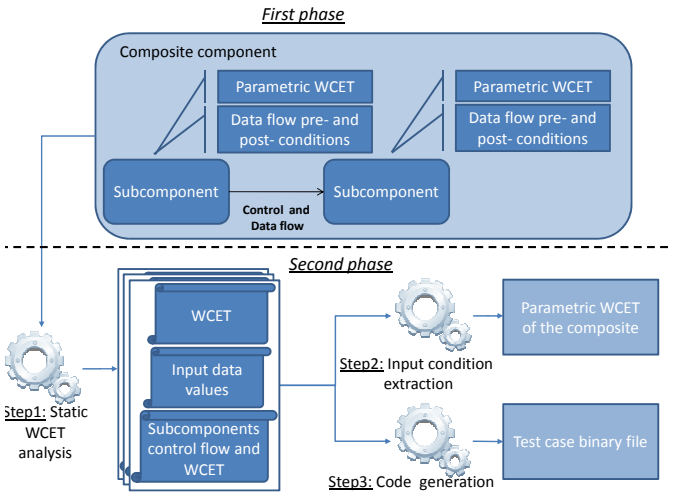

Fig. 1. Compositional WCET Estimation: Process Overview

The first phase (on the top of the figure) consists of enriching an existing component-based specification with (i) the parametric WCET estimation (see III-A3) and (ii) data flow pre- and post-conditions (see III-A1) of subcomponents. In the typical use of our approach, this information is associated to primitive components, and then automatically deduced for composite components by composition. However, it is possible to bootstrap our approach at any abstraction level.

Once subcomponents are assembled in a composite component, the parametric WCET of the composite component is automatically computed thanks to an iterative static analysis. The first step, described in more details in Section V, consists of computing the WCET of the composite component using a constraint solver. The result of this step (bottom left of the figure) consists of (i) a WCET, (ii) an example of input data values that can lead to this WCET, and (iii) the execution flow of the subcomponents and their WCET. By iterating on this step, we discover all the computational modes of the composite component that lead to different values of WCET. This iteration process is described in section VI.

The second step, described in more details in section VI, consists of transforming those computational modes into parametric WCET of the composite component by (i) extracting the input port conditions corresponding to each computational mode, and (ii) computing the WCET corresponding to those input conditions.

Finally, in the third phase (bottom right of the figure) of our approach, a WCET based test-case of the composite component is automatically generated by the component framework. The way this generation process operates is described hereafter (see section VII).

The approach we present in this paper helps in managing the WCET estimation not only by composition means, but also by giving to the software architect the possibility to parametrize the trade-off between precision and scalability. We discuss this aspect of our contribution in the next subsection.

C. Parametrization of the Precision/Scalability Trade-off

In our MDE approach, we offer the possibility to parametrize the model of the system under design with respect to the trade-off between feasibility and precision of the WCET estimation. Our objective is thus to give the software designer the possibility to reduce the number of considered computational modes in order to bound the WCET composition process. This objective raises the following question: what is the relevant metric in order to limit the analysis? We give below some of the metrics we use or plan to use in our approach. To be more precise, we focus here on the static analysis part of our approach.

1) Computational Modes Bounds: As a first implementation of our approach, we propose to limit the number of computed modes for each subcomponent, as well as the number of expected results for a given composite. In order to take advantage of our method, we propose to compute both the most and least time consuming parametric WCETs. Thus, when composing the results obtained on the composite with other components parametric WCET, we offer the possibility to lower the pessimism by computing the smallest parametric WCETs.

2) Timing Bounds: Another possible metric is the time required for computing the parametric WCET. In this case, computations of parametric WCETs are launched (to discover alternatively the most and least time consuming parametric WCETs) until the time bound for analysis is past.

IV. Use Case

In this section, we present our use-case, its design and the corresponding analysis results. In order to evaluate the feasibility of our approach, we evaluated it on a use-case extracted from the automotive domain: an Advanced Cruise Controller (ACC) system. This system extends the regular cruise control functionality (i.e., keeping the speed of the vehicle constant) in the following ways:

- Automatically adjusting the vehicle speed to keep a constant distance to the car in front.

- Adjusting the speed to the current speed limit provided by signposts.
- Providing emergency brake assistance to avoid collisions.

A. ProCom Component Model

We use the ProCom component model [12] which fits requirements presented in section III-A. We enrich the component definition with attributes to define parametric WCET and data flow constraints thanks to the PRIDE attribute extension mechanism. Figure 2 shows the architecture of the ACC system. It is built as a composite which contains five subcomponents. Data input- and output ports are denoted by small rectangles, and triangles denote trigger ports. Connections between data- and trigger ports define transfer of data and control, respectively. Connectors, depicted as rectangles with rounded corners, provide more detailed control over the synchronization and communication between the subcomponents. In particular, the *selection* connector dynamically decides to which output port an incoming triggering should be forwarded, depending of the values on its input data ports.

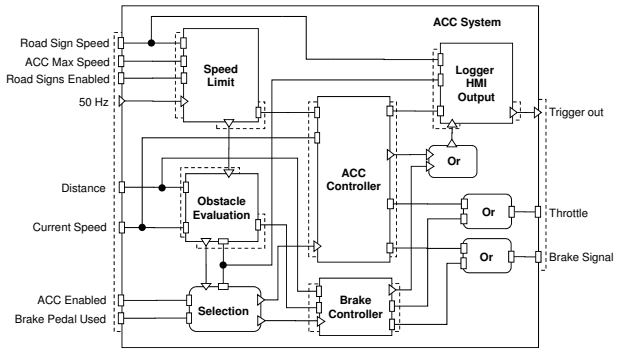

Fig. 2. Advanced Cruise Control system

In order to illustrate the different modeling artefacts we use, we consider the *"Speed Limit"* component used in *"Advanced Cruise Control"* component. The main functionality of this component is to compare the desired speed of the vehicule with a the road's speed limitation.

Figure 3 illustrates the specification of the interfaces of the *Speed Limit* component, using the ProCom graphical notation. This component has one input port group (composed of one input trigger port *"triggerIn"* and three input data ports: *"roadSignEnabled"*, *"speedCommand"*, and *"roadSpeed"*) and one output port group (composed of one output trigger port *"triggerOut"*, and one output data port *"cmdSpeed"*).

As a consequence of the semantic of a ProCom component, the control and data flows of such a component are explicit: once its intput trigger port has been solicited, the input data values are made accessible to the internal structure of the component. The corresponding input trigger port cannot be activated again before all its output trigger ports have been triggered. Besides, an output port must be triggered once and only once each time the related input trigger port has been activated. When an output trigger port is triggered, the data represented by the output data ports of the same port group become available for components that are connected to it.

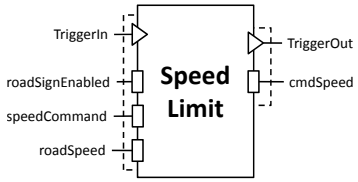

Fig. 3. Model of the Speed Limit Component

To summarize, the execution pattern is the following: trigger input port → computation using input data → production of output data → trigger each output port once and only once. This semantic is the same for a composite component and a primitive one.

V. WCET VALUE COMPUTATION

The approach we propose in this paper comes with a complex method that strongly relies on constraint solving techniques. As we described in section III, the first step of this method aims at discovering (i) the WCET of the composite component under analysis, (ii) the corresponding control flow of subcomponents, and (iii) the corresponding WCET of executed subcomponents. An overview of this step is synthesized in figure 4. As one can see in this figure, the inputs of this step are the different modelling artefacts described earlier:

- the composite component design;
- the parametric WCET of subcomponents;
- the pre- and post- conditions on subcomponents data flow.

The information contained in these modelling artefacts is first transformed, automatically, into different type of constraints. Following the same order as in the previous enumeration, the modelling artefacts are transformed into:

- control and data flow constraints;
- data flow and timing constraints;
- data flow constraints.

Those different constraints are then provided to a constraint solver that returns the expected results: the overall WCET of the composite and the corresponding control flow of subcomponents with their WCET.

In the remainder of this section, we first present a formalization of the problem the constraint solving has to solve, before to explain in more detail how the different modelling inputs are transformed into relevant constraints.

A. The constraint solving objective

From the theoretical basis described in [9], the objective of our constraint solving is to find the path of components in a component graph with the longest-combined execution times of components $C_1..C_n$. We formulate this problem as an implicit-path enumeration technique (IPET) problem [8]. The WCET of the considered composite is found by maximizing

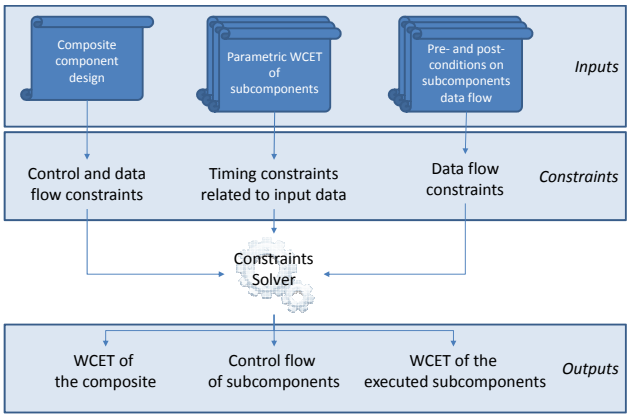

Fig. 4. Static WCET Analysis: process

the sum given in formula (1), where each component C_i has its WCET equal to w_i and is executed x_i times (according to the control flow information).

$$wcet = \sum_{i=1}^{n} x_i * w_i \tag{1}$$

This paper describes the extension made on the original WCET analysis method [9] to improve WCET estimate. The original WCET analysis method assumes a flat component model, and is applied to a task composed of components. In this paper, we consider hierarchical component models. Therefore, we extend the original approach to produce estimates for composite components instead of tasks, and to compute the parametric WCET of the composite component.

B. Mapping the Parametric WCET of Subcomponents

According to Definition 1, parametric WCET is expressed in terms of the component input values. A simplified version of the expression grammar is described in (2).

$$\begin{array}{lcl} \text{condExp} & \rightarrow & \text{ci} \Rightarrow \text{wcet = constant} \\ \text{ci} & \rightarrow & \text{logc } logOp \text{ logc} \\ \text{logc} & \rightarrow & \text{val } compOp \text{ val} \\ \text{logOp} & \rightarrow & \text{and} \mid \text{or} \\ \text{val} & \rightarrow & \text{constant} \mid \text{dataPort} \\ \text{compOp} & \rightarrow & = \mid < \mid > \mid \leq \mid \geq \mid \neq \end{array} \tag{2}$$

Listing 1 represents the different parametric WCET specified for the *Speed Limit* component. For instance, when the boolean input data *"roadSignEnabled"* is false (mode *mode1*), then the WCET of the component equals 1. In mode *mode2* (*"roadSignEnabled"* is true), the WCET of the component equals 3.

```
mode mode1: not roadSignEnabled => wcet = 1
mode mode2: roadSignEnabled => wcet = 3
```

Listing 1. Speed Limit Component: Parametric WCET

In order to take advantage of parametric nature of subcomponent WCET, we need to represent all input ports as variables with corresponding type in the IPET formulas. The translation to a constraint is straightforward. Assuming that parametric WCET of component C_i is described as a set of mutually exclusive conditions $c_i cond_1(I_1, .., I_n)$ to $c_i cond_m(I_1, .., I_n)$ over input ports I_1 to I_n and their respectively related WCET values V_1 to V_m, the constraint is described in Formula (3).

$$\bigwedge_{j \in 1..m} c_i cond_j(I_1, .., I_n) \Rightarrow w_i = V_j \tag{3}$$

C. Mapping the Pre- and Post- Conditions

As described previously, data flow pre-conditions define the domain range of each input port. Conversely, data flow post conditions describe correlation between conditions on inputs and conditions on outputs. We assume that conditions are logical expressions based on comparison of port values (*hypothesis 1*). A simplified version of the pre- and post-conditions grammar is described in (4).

$$\begin{array}{lcl} \text{condExp} & \rightarrow & \text{ci} \Rightarrow \text{co} \\ \text{ci} & \rightarrow & \text{logc } logOp \text{ logc} \\ \text{co} & \rightarrow & \text{logc } logOp \text{ logc} \\ \text{logc} & \rightarrow & \text{val } compOp \text{ val} \\ \text{logOp} & \rightarrow & \text{and} \mid \text{or} \\ \text{val} & \rightarrow & \text{constant} \mid \text{dataPort} \\ \text{compOp} & \rightarrow & = \mid < \mid > \mid \leq \mid \geq \mid \neq \end{array} \tag{4}$$

Listing 2 represents the data flow pre- and post-conditions specified for the *Speed Limit* component. In this listing, the case *m1* corresponds to the following behaviour: when the road sign information is available (*roadSignEnabled* is true) and the desired speed is superior to the road speed limitation (*speedCommand* > *roadSpeed*), then the output command speed is the road speed limitation (*cmdSpeed=roadSpeed*).

```
case m1: roadSignEnabled and speedCommand > roadSpeed
                    => cmdSpeed = roadSpeed
case m2: roadSignEnabled and speedCommand =< roadSpeed
                    => cmdSpeed = speedCommand
case m3: not roadSignEnabled => cmdSpeed = speedCommand
```

Listing 2. Speed Limit Component: Data Flow Pre- and Post-conditions

As input ports, all output ports become variables with corresponding type in the IPET formulas. Assuming that data flow post conditions of component C_i are described as a set of conditions $c_i cond_1(I_1, .., I_n)$ to $c_i cond_m(I_1, .., I_n)$ over input ports I_1 to I_n and their respectively impacts $c_i impact_1(0_1, .., O_k)$ to $c_i impact_m(O_1, .., O_k)$ on output ports $O1$ to Ok , the information is equivalent to Formula (5).

$$\bigwedge_{j \in 1..m} c_i cond_j(I1, .., In) \Rightarrow c_i impact_j(O_1, .., O_k) \tag{5}$$

D. Mapping the Composite Design

Although the theoretical analysis can handle control loops, we take the hypothesis that there are no control loops in the component model (*hypothesis 2*). Many component models specifically targeting real-time systems use a function block architectural style (e.g., Rubus[2] and ProCom [12]) where

[2]http://www.arcticus-systems.com

loops are allowed within the component code but not at the architectural level except in the form of periodic or aperiodic activation. Since the considered control flows exclude loops, x_i (the number of times subcomponent C_i is executed) is not greater than 1.

In a similar way as for components, each control connection is materialized as a variable c_ic_j representing number of times the connection between component C_i and component C_j is executed. According to *hypothesis 2*, c_ic_j cannot be greater than 1. A component is executed when one of the input control connections is executed and must execute one of its output control connection. As an example, assuming that a component C_i is connected to C_1 and C_2 through its input ports and to C_3 and C_4 through output ports, this semantic is translated in Formula (6).

$$(x_i = c_1c_i + c_2c_i = c_ic_3 + c_ic_4) \wedge (c_1c_i + c_2c_i = 1) \wedge (c_ic_3 + c_ic_4 = 1) \quad (6)$$

We extended the original analysis to take into account contextual control flow. We assume that the control flow is completely formally defined (*hypothesis 3*). In particular, conditional execution path must define formally the conditions of their execution. We assume that each connector which is in charge of selection describes its output connection execution conditions as a logical expression over the input data port values. Listing 3 represents the control flow definition thanks to selection expression for the *Selection* connector. For instance, when the boolean input data *"BrakePedalUsed"* is true (mode *brake*), then the *Brake Controller* component is executed. If *"BrakePedalUsed"* is false and *"ACCEnabled"* is true and *"emergencyBrake"* is false (mode *cruiseControl*), then the *ACC Controller* component is executed.

```
mode brake: BrakePedalUsed=true or ACCEnabled=false or
            emergencyBrake=true => Brake;
mode cruiseControl: not (BrakePedalUsed=true or
                    ACCEnabled=false or emergencyBrake=true)
                    => ACC_Controller;
```

Listing 3. Selection Connector: Control Flow Expression

The fact that connection c_ic_j is executed when condition $c_ijcond(p_1, p_2)$ is true will be translated by Formula (7).

$$c_ijcond(p_1, p_2) \Rightarrow c_ic_j = 1 \quad (7)$$

VI. Parametric WCET Computation

The previous section presented how to compute the WCET of a composite component using satisfiability constraint solving techniques. One benefit of using such method is that we also obtain the description of the corresponding control flow of subcomponent, with their respective WCET. In this section, we show how our method takes advantage of this information to compute the parametric WCET expression of the composite component under analysis.

Figure 5 synthesize the complete process of this approach. On the top left part of the figure, the first step consists of getting the overall WCET value of the composite component. This step corresponds to the constraint solving problem described in the previous section. The second step consists of extracting potential WCET values for which we try to extract a parametric expression in the third step. These two steps are further described in the following subsections. Finally, by merging the different results, we obtain the parametric WCET of the composite component.

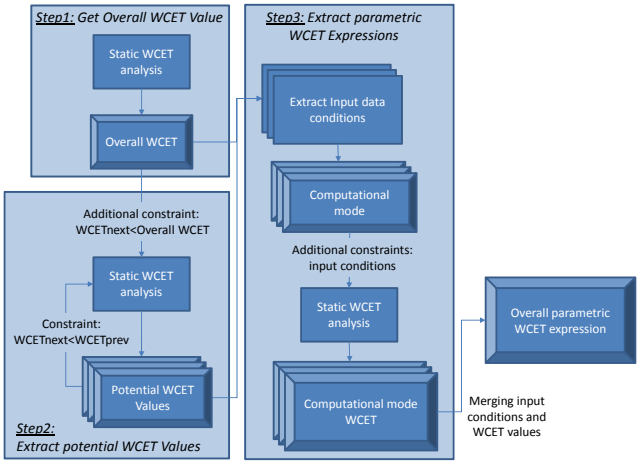

Fig. 5. Parametric WCET Extraction: process

A. Potential WCET Value Computation

First, after finding the composite component WCET value (overall WCET in figure 5), we discover other potential WCET values which could correspond to a computational mode by applying the WCET value computation while adding a constraint specifying that the computed WCET value (WCETnext on figure 5) must be lower than the previous WCET value (see bottom left part on figure 5). When the problem cannot be solved, we have found all the potential WCET values. An interesting alternative is to search only for potential WCET values for which the gap between the two values (previous and next) is greater than a percentage of the previously computed value. Similar to our work, [6] identifies candidate WCET paths for a program with a different purpose however. In the case of [6], multiple longest paths are computed in an attempt to improve the accuracy of the returned WCET whereas in our case we do it because usually it is desired to have information about the k most critical paths instead of just the one most critical path. We use the same strategy as [6] i.e. in order to identify a new potential WCET path, we inject a new constraint in the constraint system, and solve again — this is repeated until a predefined stopping criterion is met.

B. Input Condition Computation

From these results, the second step aims at identifying conditions on input ports that lead to each WCET value (see top right part of figure 5). Each computed WCET value comes with the WCET value of the subcomponents and the control flow of executed subcomponents. The input condition extraction process can be summarized as following:

A. For each subcomponent, compute input conditions that lead to WCET from considered WCET analysis scenario.
B. Iterate from last executed subcomponent to first executed subcomponent by following in reverse direction the control flow. For each subcomponent, perform the following steps:
   a) Propagate input conditions from step A to output conditions on the connected data ports.
   b) Consider computed output conditions CCo from step B.a: from pre- and post conditions ($Ci \Rightarrow Co$), identify input conditions Ci that ensure these output conditions CCo ($Ci \Rightarrow CCo$). This is done by checking with a constraint solver that $Co \Rightarrow CCo$. If not verified, we ignore the rule $Ci \Rightarrow Co$.
   c) Merge input conditions from step A and step B.b. We ignore input conditions from step B.b which are incompatible with ones from step A.

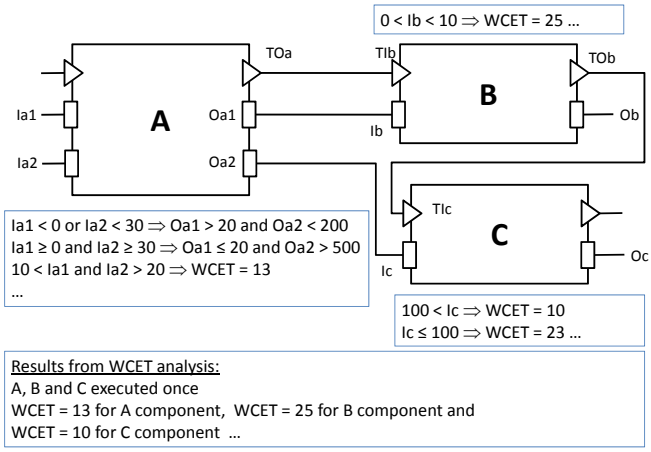

Fig. 6. Example of input conditions extraction

This extraction computation is feasible due to the fact that data flow constraints contain only comparison of port values which allows easy and efficient constraint comparison. To illustrate this algorithm, we introduce the example from figure 6. According to WCET analysis result, that states that WCET of component C is 10, we deduce that Ic must be greater than 100 (step A).

Assuming that the input conditions have been computed for component B and C, we transform these input conditions ((Ic $>$ 100) and (0 $<$ Ib $<$ 10)) to an equivalent condition on connected output ports ((Oa2 $>$ 100) and (0 $<$ Oa1 $<$ 10)) (step B.b). These conditions are compared to possible ouput conditions defined in post-conditions expressions (step B.a). This comparison is performed by using a constraint solver that allow us to define if a condition is always true, sometimes true or never true. In our example, the expression ((Oa1 $=$ 20) and (Oa2 $>$ 500)) always implies ((Oa2 $>$ 100) and (0 $<$ Oa1 $<$ 10)). That is why we consider ((Ia1 $\geq$ 0) and (Ia2 $\geq$ 30)) as mandatory. Finally, we merge these input conditions with ones coming from expected WCET for this conponent ((10 $<$ Ia1) and (Ia2 $>$ 20)). So, we obtain (((10 $<$ Ia1) and (Ia2 $>$ 20)) and ((Ia1 $\geq$ 0) and (Ia2 $\geq$ 30))) which could be reduced in ((10 $<$ Ia1) and (Ia2 $\geq$ 30)). The algorithm stops when all executed subcomponents have been evaluated.

C. WCET Value Computation

Unfortunately, this extraction may suffer of lack of precision when a computational mode cannot be expressed as a condition over the inputs. As an example, assume that a component

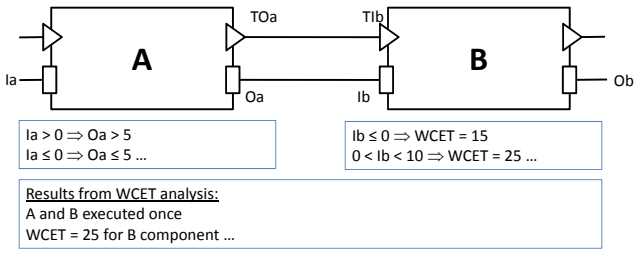

Fig. 7. Example of impossible input conditions extraction

A and a Component *B* are connected through data ports Oa and Ib and trigger ports TOa and TIb (see figure 7). Imagine that results from WCET analysis defines A and B as executed and that considered WCET of component B is 25 ms. Our objective is to find the conditions on component A inputs which lead to have 25 ms as component B WCET. From the WCET expression of component B, we can deduce that Ib value must be between 0 and 10 to lead to this WCET time. Following data connection between Ib and Oa, we can deduce that Oa value must be between 0 and 10. Unfortunately, there is conditions on Ia that ensure that Oa value will be between 0 and 10. As the A component developper has not described the complete functional behaviour, we are not able to deduce the input conditions that ensure a specific value as output.

In this case of condition extraction failure, the remaining conditions that cannot be expressed as condition on input are ignored. It leads to possible underestimate of WCET value as we do not have all constraints. To evaluate the impact of constraint removal, the WCET value computation is launched with computed input conditions (see bottom right part of figure 5). Finally, the parametric expression is obtained by merging all the results.

VII. Test-case Integration

The motivation for generating test cases is twofold:

A. to ensure that the code wrapping the components (usually referenced as glue code) do not lead to bigger WCET than the on produced by the static analysis;
B. to compare the execution time with the results from the static analysis in order to evaluate the precision of the static analysis.

We present hereafter how we instrument the generated code to help in providing such results. Our assumption is that an

existing test-case generation technique provides us with a set of data vectors that enable to cover significantly the range of inputs of a parametric WCET. The idea is to limit the possible number of input by selecting data ranges that correspond to the parametric WCET under analysis and excludes data ranges corresponding to lower WCET.

We then build the test-case of a computational mode of a component. We generate the code enveloping subcomponents the same way it would have been produced for a release of the composite, and launch its initializing its input data ports with the produced data vectors. The execution time of the composite is measured, and stored in a result file formatted as follows: Input values → Execution time → Output values.

The generated test-case (i) checks that the obtained execution time is not superior to the parametric WCET computed by static analysis and (ii) produces the difference between the computed parametric WCET and the measured WCET.

VIII. Evaluation And Experiments

A. Parametric WCET Evaluation

Our example (see figure 2) is a composite component composed of 5 subcomponents and 7 connectors, which is representative of the complexity of a composite component.

The static analysis performed on the model of the ACC system component (see Figure 2) produced 11 different computational modes for which the WCETs go from 18ms to 63ms. In comparison, the method presented in [11] would have given a unique WCET of 88ms. Due to a lack of precision in the description of one component, only 6 of those computational modes could be defined as parametric WCETs. For those 6 parametric WCETs, we have extracted the corresponding condition on the input data port of the composite. On a dual core CPU with 2,79 GHz and a memory space of 3,48 GB of RAM, the overall analysis took about 50 seconds. Listing 4 illustrates the input condition found for a WCET of 22 ms.

```
( distance <10 and CurrentSpeed=<30 and roadSignEnabled=false
and (BrakePedalUsed=true or ACCEnabled=false or
( distance <20 and CurrentSpeed>30))) or
( distance >=10 and (distance>=20 or CurrentSpeed=<30)
and roadSignEnabled=false and (BrakePedalUsed=true or
ACCEnabled=false or (distance>=20 and CurrentSpeed=<30)))
        => wcet=22
```

Listing 4. ACC_System parametric WCET

B. Test-case Generation

Besides this analytical result, we have generated the code corresponding to the ACC_System component. Using the result of the analysis in order to initialize the input data port of the composite, we have measured the corresponding execution time for the six different parametric WCET. The implementation code of primitive components was executed with timing characteristics corresponding to those defined in the model. Thus, in every test performed, the measured WCET was very close to the analysis result. This experiment enabled to test our approach from end-to-end.

IX. Concluding Remarks and Future Work

We have presented a solution to compose parametric WCET computation in a component-based approach. Taking into account component data flow constraints and formal definition of execution path, we produce tight WCET estimates compared to naive additive WCET analysis. We have demonstrated that discovering all contextual WCET can be automated. Finally, our input domain extraction technique allows to discover conditions over composite component inputs of that leads to a specific WCET.

In our future work, we plan to use source code analysis to compute the parametric WCET of primitive component. We also plan to evaluate the composability of WCET precision in order to deduce the precision of the WCET of a composite from the precision of its subcomponents WCET.

References

[1] G. Bernat, A. Colin, and S. M. Petters, "WCET analysis of probabilistic hard real-time systems," in *In Proceedings of the 23rd Real-Time Systems Symposium RTSS 2002*, 2002, pp. 279–288.

[2] S. Bygde, A. Ermedahl, and B. Lisper, "An efficient algorithm for parametric WCET calculation," in *The 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2009*, P. Kellenberger, Ed. IEEE Computer Society, August 2009, pp. 13–21.

[3] J.-F. Deverge and I. Puaut, "Safe measurement-based WCET estimation," in *5th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*, R. Wilhelm, Ed. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2007.

[4] J. Fredriksson, T. Nolte, A. Ermedahl, and M. Nolin, "Clustering worst-case execution times for software components," in *Proceedings of the 7th International Workshop on Worst Case Execution Time Analysis (WCET'07)*, July 2007, pp. 19–25.

[5] J. Gustafsson, "Usability aspects of WCET analysis," in *11th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC'08)*. IEEE, 2008, pp. 346–342.

[6] C. Im and K. Kim, "A hybrid approach in TADE for derivation of execution time bounds of program-segments in distributed real-time embedded computing," in *9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06)*, 2006.

[7] R. Kirner, P. Puschner, I. Wenzel, and T. U. Wien, "Measurement-based worst-case execution time analysis using automatic test-data generation," in *In Proc. IEEE Workshop on Software Tech. for Future Embedded and Ubiquitous Systs. (SEUS'05)*, 2004, pp. 7–10.

[8] Y. S. Li and S. Malik, "Performance Analysis of Embedded Software Using Implicit Path Enumeration," in *Proceedings of the ACM SIGPLAN 1995 workshop on Languages, compilers, and tools for real-time systems (LCTES'95)*. New York, NY, USA: ACM Press, 1995, pp. 88–98.

[9] A. Marref, "Compositional timing analysis," in *SAMOS X – International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*. IEEE, June 2010.

[10] P. Puschner, R. Kirner, and R. G. Pettit, "Towards composable timing for real-time programs," in *STFSSD '09: Proceedings of the 2009 Software Technologies for Future Dependable Distributed Systems*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–5.

[11] M. Santos and B. Lisper, "Evaluation of an additive WCET model for software components," in *WTR 2008 10th Brazilian Workshop on Real-time and Embedded Systems*, May 2008.

[12] S. Sentilles, A. Vulgarakis, T. Bureš, J. Carlson, and I. Crnković, "A component model for control-intensive distributed embedded systems," in *11th International Symposium on Component-Based Software Engineering (CBSE'08)*, ser. Lecture Notes in Computer Science, vol. 5282. Springer Berlin, 2008, pp. 310–317.

[13] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 1–53, 2008.