Taming the Component Timing: A CBD Methodology for Real-time Embedded Systems

Manoj G. Dixit  
India Science Lab  
GM R&D

Pallab Dasgupta  
Indian Institute of Technology, Kharagpur

S. Ramesh  
India Science Lab  
GM R&D

Abstract—The growing trend towards using component based design approach in embedded system development requires addressing newer system engineering challenges. These systems are usually time critical and require timing guarantees from components. The articulation of a desirable response bounds for the components is often ad-hoc and happens late in development.

In this work, we present a formal methods based methodology for an early stage design space exploration. We focus on real-time response of a component as a basis for exploration and allow the developer model it using constant values or parameters. To quantify the parameters, we propose a novel constraint synthesis technique to correlate response times of interacting components. Finally, for system integration, we introduce a new notion of timing layout to specify time-budgeting for each component. The selection of a suitable layout can be made based on system optimization criteria. We have demonstrated our methodology on an automotive Adaptive Cruise Control feature.

I. INTRODUCTION

Component Based Design (CBD) methodology advocates the development of large software systems by composing more basic (often standardized) components. Due to various well known advantages, large multi-disciplinary OEMs, like automotive and aircraft industries, are embracing CBD methodology. For example, consortium effort like AUTOSAR [1] is a recent attempt to enable early and easy adoption of CBD methodology in the automotive domain.

CBD requires fundamental changes in the way embedded software is presently developed. Besides the changes in the infrastructure platform and the software architecture, the methodology of design and verification of individual features or components, which is the focus of this paper, also requires revision. Traditionally, the development of a system level feature starts with its specification, called design intent, which consists of a set of functional requirements. It is then decomposed into a set of component requirements which are then used for developing or purchasing the components meeting the requirements. Software engineering text books [2] discuss in details many of these decomposition methods. The verification step is an important step towards ensuring that the integrated component requirements achieves overall feature design intent.

Many embedded subsystems [3] are time critical. It is imperative that the design intent for such features include timing requirements as well. Decomposition of such timing requirements is not studied very well and is one of the challenges in the adoption of CBD methodology for real-time systems. The main motivation of this work is to extend the standard design intent decomposition methodology to component based frameworks that include real time requirements.

At the feature level, one could have an end-to-end timing requirement which needs to be reduced to many timing requirements at the individual component level while ensuring the design intent coverage. The intent coverage ensures that the overall feature timing requirements are met when you compose the components meeting their respective timing requirements. The reduction of timing requirements requires time budgeting that distributes the end-to-end-time to all the components.

There would be multiple ways of budgeting the time and an early choice might affect the end result. It is important to delay the budgeting decisions so as to take a globally optimal decisions. Also it is desirable to do design space exploration that permits analysing the requirements for multiple choices of timing budgeting. The main aim of this paper is precisely to propose a methodology that supports design exploration with multiple choices of timing budgets and selecting the right one at the appropriate time. For supporting late choice of timing budgets and design exploration, we propose respectively the use of parametric specifications and constraint solver.

Related work: The problem of design intent coverage for functional specifications has been formulated in [4]. Our work provides (parametric) timing extensions to it. The widely used timing analysis methods are related to schedulability [5], [6]. Of late, efforts are also on to model application level timing [7], [8]. But, they focus on design issues whereas we on requirements. In [9] a related constraint synthesis approach is presented. Our work provides an improvement by optimizing the search procedure that reduces the size of the constraint set. Finally, the introduction of parameterised timing for design space exploration is novel for CBD and to the best of our knowledge, we have not yet seen any related work on this.

The rest of the paper is organized as follows, Section II gives overview of the proposed methodology, Sections III, IV and V gives a formal description of the methodology and Section VI concludes with a case study.

II. METHODOLOGY OVERVIEW

Fig. 1 pictorially depicts the proposed methodology. As shown in Fig. 1(a), the first step involves manually identifying a set of components and their requirements by decomposing feature requirements. Since component level timing is unknown, we represent it using parameters.
As shown in Fig. 1(b), the second step has two parts. Part (i) is the verification step of checking the correctness of feature level functional requirements decomposition. This corresponds to the functional intent coverage problem already discussed in depth in [4]. Part (ii), is the most important step towards component level time budgeting. The non trivial aspect of this arises due to the parameters in component specifications. This requires generating an intent coverage constraint, any solution to which defines a budget achieving timing intent coverage.

As shown in Fig. 1(c), the third step aims at obtaining a set of component specification alternatives; each based on a specific timing budget for components. This is achieved by integrating an intent coverage constraint with additional constraints derived from field data and selecting a design optimization criterion to make the final choice realistic from end application point of view.

III. PLTL AND CORNER POINTS

For describing requirements, we provide a library of textual patterns along the lines of [10], that are mapped to a fragment of Parametric Linear Temporal Logic (PLTL) [9]. PLTL extends standard temporal logic operators □, ♦ with quantitative extensions: □≤xΨ (Ψ shall hold continuously for 5 units of time) and parametric extensions: ◦≤xΨ (Ψ shall hold within x (parameter) units of time). The timing intent decomposition problem is naturally formalised based upon the semantics rules for PLTL. For a PLTL property Φ defined over X and a parameter valuation function α : X → N, the pair (Φ, α) represents an instantiated property obtained from Φ, by replacing every occurrence of parameter x ∈ X by α(x). For a PLTL property Φ, generating a set of constraints, such that any parameter valuation defined by its solutions makes Φ valid, forms the central aspect of time budgeting. We now describe a new constraint generation approach for a fragment PLTL□ having parameters appended to □ operator only. The case of its dual PLTL♦ is similar. For describing the constraints we have introduced a new notion of corner points for PLTL.

Let Ψ be a PLTL□ property involving k parameters and let \( V_\Psi = \{ \alpha \mid (\Psi, \alpha) \text{ is valid} \} \) define the solution set. Given two valuations α, β, we say that α dominates β iff \( \forall x \in X, \beta(x) \leq \alpha(x) \). \( V_\Psi \) satisfies the downward closure property with respect to dominates relation ; \( \alpha \in V_\Psi \) and α dominates β then, \( \beta \in V_\Psi \). Since valuations represent points in \( \mathbb{N}^k \), \( V_\Psi \) would correspond to a down-closed region in it.

Fig. 2 shows a downward closed region contained within the points forming a staircase for the case k = 2. The grid points represent valuations from \( V_\Psi \) and for any point α, all the points contained in the rectangle defined by α and origin also lie in it. The points \( \alpha_1, \alpha_2, \alpha_4 \) are defined as corner points because they are not dominated by any other point in the region. It can be easily checked that \( V_\Psi \) can be expressed as a union of rectangles defined by all the corner points or alternatively, as the constraint \( \bigwedge_{j=1}^4 \bigwedge_{i=1}^2 x_i \leq \alpha_j(x_i) \). Thus, constraint computation can be reduced to the enumeration of all corner points and we make use of this reduction.

For a general k, the definition of corner points extends naturally and Theorem 1 ensures plausibility of their enumeration.

Theorem 1: Every downward closed region in \( \mathbb{N}^k \) has finite number of corner points.

IV. TIMING INTENT DECOMPOSITION

Let \( FS = \{ F_1, \ldots, F_m \} \) denote a set of features that are implemented by a set of components \( CS = \{ C_1, \ldots, C_m \} \). Let \( X = \{ x_1, \ldots, x_k \} \) be the set of parameters used in modeling component requirements. Since the focus of the paper is on analyzing timing aspects, we shall consider only timing requirements. Let \( R_{FS}, R_{CS} \) represent the conjunction of all feature and component level PLTL properties respectively. Further, let \( D(X) \) denote a constraint on X. Now we have

Definition 1: \( CS \) achieves the intent of \( FS \) relative to a constraint \( D(X) \) iff for any α satisfying \( D(X) \), the temporal logical property \( (R_{CS}, \alpha) \) logically imply \( R_{FS} \), or more formally \( (R_{CS}, \alpha) \Rightarrow R_{FS} \) is a valid temporal logic property, where \( \Rightarrow \) is an implication operator.

This definition is a natural extension to the intent coverage problem given in [4]. We refer to any \( D(X) \) satisfying the
above definition an intent coverage constraint and \( \alpha \) as a component level timing intent decomposition.

Note that the validity check for the intent coverage can be done using the standard semantics rules of a temporal logic. Thus, the main problem here is computing \( D(X) \). Algorithm 1 describes such a procedure taking \( FS \) and \( CS \) as inputs.

\[
\text{Algorithm 1: Intent Coverage Constraint Generation}
\]

Algorithm 1 iterates (lines 3-6) over all intent decomposition pairs \((f_{ij}, c)\), where, \( f_{ij} \) is the \( j \)-th property of \( F_i \) and \( c \) is the conjunction of component properties derived for \( f_{ij} \) and computes a linear constraint \( D_{ij} \) using Algorithm 2, such that if \( \alpha \) satisfy \( D_{ij} \), then \( (c \Rightarrow f_{ij}, \alpha) \) is a valid formula. It can be easily verified that the final output is the required constraint.

\[
\text{Algorithm 2: Validity Constraint Generation}
\]

Algorithm 2 uses the constraint generation approach from Section III when \( \Psi \) belongs to PLTL\[\square\]. It describes a recursive procedure obtaining a set of corner points dominating the input base valuation \( \beta \) and returning the constraint defined by them. The algorithm consists of two main steps. The search step (line 2) finds a corner point. The split step (line 6), recursively continues further search by removing certain parts of \( V_\Psi \), where no corner point can lie. It also defines new sets of inputs required for further search. Finally, \( D(X) \), is obtained (line 7) as the desired constraint. The general case for \( \Psi \) is handled using heuristics based on this, which we skip here due to want of space.

The search step is described by Algorithm 3 when \( V_\Psi \) is bounded. The general case is similar and is ignored due to space constraints. At the beginning a check is made for the existence of a non-trivial valuation in \( V_\Psi \) dominating \( \beta \) and in case of success, an iterative construction of a finite sequence of pairs \((\Psi_i = \Psi, \beta_i = \beta), \ldots, (\Psi_k + 1, \beta_{k+1})\) is made in such a way that \( \beta_i \leq \beta_{i+1} \) and \( \beta_{k+1} \) is a corner point.

Fig. 3(a), (b), illustrate intermediate steps of this construction, with \( \beta \) as a zero-valuation \( o \) and resulting in \( \alpha_2 \) as corner point for the region shown in Fig. 2. As shown in Fig. 3(a), \( \beta_2 \), is found out as the farthest valuation belonging to \( V_\Psi \) on the diagonal line (slope 1) originating from \( \beta_1 \) (line 9). Note that \( \beta_2 \) lie on boundary of \( V_\Psi \). Using \( \beta_2 \), a check is performed (line 10) to find out which parameters must be fixed so that there is a possibility of getting a valuation dominating \( \beta_2 \) in \( V_\Psi \). In this particular case, choice is \( \{x_1\} \). The property \( \Psi_2 \) is derived from \( \Psi_1 \) by replacing all occurrences of \( x_1 \) with \( \beta_2(x_1) \) (line 12). This makes \( \Psi_2 \) to be a property in one variable. By restricting \( \beta_2 \) to \( \{x_2\} \), we get a base valuation for \( \Psi_2 \). As shown in Fig. 3(b), the next iteration on \( (\Psi_2, \beta_2) \) results in the corner point \( \alpha_2 \).

Next, we illustrate split step using Fig. 3(c). The corner valuation \( \alpha_2 \) splits \( \mathbb{N}^2 \) into \( R_1 \cdots R_4 \) parts. Since \( \alpha_2 \) is a corner point, \( R_1 \) and \( R_2 \) will not contain any other corner point. Hence, Split procedure (Algorithm 2, line 6) recursively divides the search into remaining two parts by using \( \beta_2^l \) and \( \beta_2^l \) as base valuations. Now, we state the main result:

**Theorem 2:** The ValCheck procedure finds out all corner points of \( V_\Psi \) when invoked with an origin \( (\alpha_0) \) as a base
valuation and \{\} as a list. Thus, if \(\{\alpha_1, \ldots, \alpha_l\}\) are corner points, the constraint defined by them is: \(\bigwedge_{j=1}^l \bigwedge_{i=1}^{\alpha_i} x_i \leq \alpha_j(x_i)\)

V. DESIGN SPACE EXPLORATION

Though the intent coverage constraint captures component’s time budgeting, to make a realistic choice of valuation, we strengthen the selection process by (i) augmenting the constraints by field data and (ii) using an objective criteria for design optimization.

### Table II

**Design Space Exploration Parameters**

<table>
<thead>
<tr>
<th>Input Parameter</th>
<th>Possible Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add-on Constraints</td>
<td>(a) Interface</td>
<td>(x = x' + x''), (I \leq x'' \leq h)</td>
</tr>
<tr>
<td></td>
<td>(b) U/L Bound</td>
<td>(l \leq x \leq h)</td>
</tr>
<tr>
<td></td>
<td>(c) Synchronization</td>
<td>(\sum_{i \in X} x_i = \sum_{i \in X} y_i \leq m)</td>
</tr>
<tr>
<td>Objective Criteria</td>
<td>(a) Min/Max</td>
<td>(\min \max_{i=1} x_i \leq x)</td>
</tr>
<tr>
<td></td>
<td>(b) Multi-Parameter</td>
<td>(\max_{i=1} \min_{x_i} x)</td>
</tr>
</tbody>
</table>

Table II row 1 defines add-on constraint patterns. Type (a) incorporates timing constraints for interactions with sensors and actuators. A parameter \(x\) is split into \(x'\) modeling the software response time and \(x''\) modeling the hardware response time. The bounds on latter will be usually known from its data-sheets. Type (b) is a range constraint. Type (c) models a bound on relative time difference between two or more component timing chains. The row 2 shows two objective criteria for modeling system level optimization condition. Type (a) is a standard min/max, whereas type (b) is a multi-parameter.

Given intent coverage constraint (\(D_1(X)\)) and add-on constraints (\(D_2(X)\)) as inputs, the optimization problem for an objective criteria \((O)\) is formulated by the Equation 1

\[
\text{Optimize } O, \text{ subject to } D_1(X) \land D_2(X)
\]

This optimization is solved using a constraint solver like [11], [12]. The solution to this represent an optimal component level timing layout. For each such objective criteria, a separate timing layout can be generated. This enables exploration to arrive at a set of component specification alternatives.

### Table III

**Design Space Exploration: ACC**

<table>
<thead>
<tr>
<th>Optimization Criteria</th>
<th>Add-on Constraints</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\max x + x_1)</td>
<td>(x_1 = 50, x_2 = 75, x_3 = 100)</td>
<td>(x_4 = 50, x_5 = 75, x_6 = 100)</td>
</tr>
<tr>
<td>(25 \leq x'_i \leq 50, 50 \leq x''_i \leq 75)</td>
<td>(j \in {1, 2, 3, 4, 5, 7})</td>
<td>(100 \leq x_1 \leq 150)</td>
</tr>
<tr>
<td>(3 \leq x_j = x_i + x''_i \leq 50, x''_j = 25)</td>
<td>(k \in {3, 6})</td>
<td>(x_1 = 50, x_2 = 25, x_3 = 50, x_4 = 25, x_5 = 50)</td>
</tr>
<tr>
<td>(\max_{i=1} \min_{x_i} x)</td>
<td>(y_1 = x''_1 = 50, y_2 = 25, y_3 = 50)</td>
<td>(y_6 = x''_6 = 50, y_7 = 25, y_8 = 50)</td>
</tr>
</tbody>
</table>

Table III, column 1 shows the add-on constraints defining \(D_2(X)\). The column 2 shows design space exploration result.

VII. CONCLUSION

The approach presented provides a new direction towards a systematical budgeting of component level timing bounds at an early stage. Our initial results have shown a potential towards practicable usability of this methodology.

REFERENCES