15th International ACM SIGSOFT Symposium on Component Based Software Engineering (CBSE'12), Bertinoro, Italy, June 26-28, 201, ACM.

Timing Analysis of Component-based Embedded Systems

Jan Carlson
Mälardalen Real-Time Research Centre
Mälardalen University, Sweden
jan.carlson@mdh.se

ABSTRACT

The recent trend towards applying component-based and model-driven approaches also to the development of safety-critical real-time embedded systems, opens new possibilities for model-level analysis of aspects that traditionally are analysed very late in the development when the system has been fully implemented. For real-time systems, the temporal aspect is as important as the functional to the overall correctness of the system, and thus timing analysis in different forms play a key role in their development. This paper presents the timing analysis of ProCom, a component model specifically targeting distributed real-time embedded systems, focusing in particular on three methods for compositional model-level analysis of worst-case execution time.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification; D.2.13 [**Software Engineering**]: Reusable Software; C.3 [**Special-purpose and Application-based Systems**]: Real-time and embedded systems

Keywords

Component-Based Development, Compositional Real-Time Analysis, Embedded Systems

1. INTRODUCTION

The domain of safety-critical, real-time embedded systems is characterized by on one hand the need for analysis providing accurate and reliable estimations of the system behaviour, including timing, and on the other hand severe resource limitations in terms of, for example, memory and computational resources. As a result of the demand to constantly provide more advanced functionality at the same or lower development costs, system complexity is approaching the limit of what can be effectively handled by traditional development approaches.

Recently, component-based software engineering (CBSE) and model-driven development (MDD) have received much attention also in these domains. These approaches aim to aid system developers by promoting, respectively, the construction of systems from existing components instead of building them from scratch each time, and the use of models as key design artefacts instead of code.

Another consequence of adopting a model- or component-based approach is that some of the analyses that traditionally could only be performed once the system is fully developed can be applied in earlier stages of development, to guide the search for the most suitable design alternatives.

The work presented in this paper is carried out in the context of ProCom, a component model specifically targeting distributed real-time embedded systems, and addresses some of the timing analysis challenges found on different levels of model granularity. We present an overview of the overall timing analysis scheme envisioned for ProCom, and describe in detail three analysis methods for model-level analysis of worst case execution time.

The paper is organized as follows: Section 2 gives a short introduction to ProCom, followed by an overview of the different timing analysis methods in Section 3. Sections 4, 5 and 6 provide more details on the three model-level analysis techniques for worst case execution time. Section 7 surveys related work, and Section 8 concludes the paper.

2. PROCOM

The ProCom component model [8] addresses the particularities of the embedded systems domain, including resource limitations and requirements on safety and timeliness. ProCom is organized in two distinct layers, called *ProSave* and *ProSys*, that differ in terms of architectural style and communication paradigm. Both layers are hierarchical, i.e., supporting *composite* components internally defined by interconnected subcomponents. The way in which the two layers are linked together is that a primitive ProSys component can be modelled as a collection of ProSave subcomponents. At the bottom of the hierarchical nesting, the primitive ProSave components are implemented by C functions.

2.1 ProSave

ProSave components are passive units interacting through explicit data and control flow connections between *data ports* where data of a given type can be written or read, and *trigger ports* that control the activation of components. Data ports always appear in a group together with a single trigger port, and the ports in the same group are read or written together

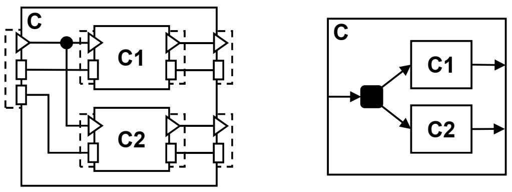

Figure 1: Original and simplified ProSave notation.

in a single atomic action. Together, an input group and its associated output groups are called a *service*.

ProSave components follow a strict read-execute-write cycle where activation is always initiated externally. Initially, all services of a component are in an idle state, just receiving data on its input data ports. When the input trigger port of a service is activated, the data at the input data ports of the service are atomically copied to internal representations which remain unchanged until the end of the service execution. Then, the service functionality is executed, during which each output port group of the service produces data once. When all output port groups have produced data, the service immediately returns to the idle state. Trigger activations arriving while in the executing state are ignored.

In addition to simple connections from output to input ports, *connectors* provide detailed control over the data- and control flow. The connectors influencing the control flow are: *Fork* which forwards control to all output ports; *Join* which forwards control to the output port only when all input ports have been triggered; *Selection* which forwards control to one of the output ports depending on the value of its data ports; and *Or* which directly forwards control from any of its input ports to the output port.

Figure 1 (left) shows a composite ProSave component C consisting of a single service (i.e., one input port group) with two output port groups. Triangles and squares denote trigger and data ports, respectively, and the filled circle represents a control fork connector forwarding the triggering to subcomponents C1 and C2. Since this paper focuses on the control flow aspect of ProSave, we will use a simplified notation shown in Figure 1 (right).

2.2 ProSys

ProSys components are active units communicating by asynchronous messages sent and received through typed message ports. Compared to ProSave, the component semantics at the ProSys level is less restricted. The semantics does not enforce explicit relations between input and output ports, nor does it places any restrictions on the interleaving of execution of functionality and sending or receiving messages.

Figure 2 exemplifies the ProSys notation, depicting a composite component with one input message port, two output message ports and two subcomponents. Note that the control flow of the composite cannot be determined without additional knowledge about the subcomponents, since no restrictions on the control flow of subcomponents are imposed by the ProSys semantics. For example, it is not known if S1 sends messages in response to incoming messages on one of its ports, or as part of an internal activity. In this paper, we will use a simplified version of the notation, where the message channel symbol is replaced by simple arrows between the ports.

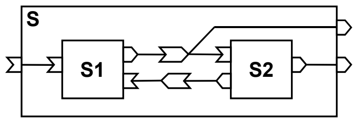

Figure 2: ProSys notation.

3. ANALYSIS OVERVIEW

The objective of real-time analysis is to establish worst case response times for crucial functionalities to ensure that system level end-to-end deadlines are met in all situations. Traditionally, this analysis is performed at task level, assuming knowledge of all tasks in the system in terms of activation patterns (e.g., period) and *worst case execution time* (WCET).

For applications developed in a rich modelling formalism, such as ProCom, the existence of additional information creates opportunities for more advanced timing analysis, compared to the traditional real-time analysis on task level. Analysis on a model level can be performed also in early stages of development, when parts of the system are still undefined. In hierarchical models, compositional analysis can improve analysis scalability. Rather than analysing the entire system at once, analysis of individual part derive timing information that can in turn be used in the analysis of enclosing entities.

The ProCom timing analysis consists of three major types of analysis: code level WCET analysis, model level WCET analysis and response time analysis. They are described below, and the model level WCET analysis mechanisms are further detailed in the next section.

3.1 Code level WCET analysis

The objective of this analysis is to establish WCET information for individual primitive ProSave components, i.e., components with functionality that is (or will be) implemented by user provided code. This type of analysis is not ProCom specific, although explicit knowledge from the component model, such as value range constraints, could be exploited for increased precision.

Timing information can be extracted from code by means of testing or by static analysis deriving a safe approximation of the WCET From a CBSE perspective, both testing and static analysis have higher impact if performed on individual components in a general context, since this allows the timing information to be reused when the component is reused in new systems. On the other hand, analysis or testing in a particular system context can give more precise information.

3.2 Model level WCET analysis

The role of the model level WCET analysis is to cover all hierarchical levels of a ProCom model, deriving WCET information for the top level entities from information about the primitive components at the bottom. In order to permit analysis also in early stages, and to facilitate reuse of analysis results, we propose a fully compositional approach where each composite component is analysed in isolation, based only on WCET information about its constituent subcomponents (previously derived or manually estimated) and not on their internal structure.

As a result of the two layers, ProSave and ProSys, three different analysis mechanisms are needed, one for each type of composite entity: composite ProSave component, primitive ProSys component (i.e., a ProSys component internally modelled as a collection of interconnected ProSave components), and composite ProSys components.

The main difference between the three analysis mechanisms is a result of the different component entities and communication style in the two layers. The ProSave semantics places more restrictions on how a component is allowed to interact with its environment. As a result, the WCET information associated with a ProSave component is fairly simple, essentially a WCET value for each service, while the information needed for a ProSys component is more elaborate as it must capture the WCET and output messages associated with internal activities as well as with received messages. On the other hand, ProSave provides richer constructs for controlling the communication between subcomponents of a composite in terms of data and control flow. Consequently, the analysis of composite ProSys components has fewer constructs to consider than the other two.

Detailed descriptions and examples of the three analysis mechanisms are given in Sections 4, 5 and 6, respectively.

3.3 Response time analysis

Traditionally, this analysis is performed on the system as a whole, since it requires information about all parts of the system to determine the overall impact on a specific task.

ProCom uses a hierarchical scheduling approach by means of *virtual nodes*, intermediate entities between software components and hardware nodes. Components are allocated to virtual nodes, and the virtual nodes are in turn allocated to the nodes of the underlying distributed system. each virtual node has an *execution budget* defining for example how much CPU resources it is entitled to and bounds on how the access may be distributed over time. This means that the response time analysis is separated in two parts:

a) Each virtual node can be analysed in isolation, independently of other parts of the system, to determine if the budget is sufficient to satisfy all timing requirements within the node.

b) On system level, we can determine if a given allocation of virtual nodes over the physical nodes is schedulable, i.e., if the execution budgets of all virtual nodes can be satisfied in all circumstances.

Note that the result from *a)* is valid also if the virtual node is reused (with the same budget) in another system.

The ProCom modelling support in terms of virtual nodes and allocation models, and its impact on timing analysis, is described in [1].

4. PROSAVE ANALYSIS

The analysis at ProSave level is performed on a single composite ProSave component. From WCET information about each subcomponent (mapping each service to an integer representing its WCET), and from the way they are connected, WCET information for the composite is derived.

The explicit relation between input and output port groups in ProSave makes it possible to follow the control flow within the composite. Another useful aspect of the restricted semantics is that a service does not permit being triggered again during execution, and the control flows of different services is not allowed to mix. This means that it is sufficient to analyse a single invocation of each service in isolation.

Algorithm 1 ProSaveWCET(p, S)

```
1: if no connection $p \to p'$ exists then
2:     return $\langle 0, S \rangle$
3: Let $p'$ be the port for which the connection $p \to p'$ exists
4: if $p'$ is an output port of the service under analysis then
5:     return $\langle 0, S \rangle$
6: Let $C$ be the construct (connector or subcomponent service) to which $p'$ belongs
7: $w \leftarrow \mathrm{WCET}(C)$
8: if $C$ is a subcomponent service or control fork then
9:     for each output port $p''$ of $C$ do
10:        $\langle w', S' \rangle \leftarrow$ ProSaveWCET$(p'', S)$
11:        $w \leftarrow w + w'$
12:        $S \leftarrow S'$
13:    return $\langle w, S \rangle$
14: else if $C$ is a control join with output port $p''$ then
15:    Let $P$ be the set of input ports of $C$
16:    if $P \subseteq S \cup \{p'\}$ then
17:        $\langle w', S' \rangle \leftarrow$ ProSaveWCET$(p'', S - P)$
18:        return $\langle w + w', S' \rangle$
19:    else
20:        return $\langle w, S \cup \{p'\} \rangle$
21: else if $C$ is a selection then
22:    $w_{\mathrm{tot}} \leftarrow 0$
23:    $S_{\mathrm{tot}} \leftarrow \emptyset$
24:    for each output port $p''$ of $C$ do
25:        $\langle w', S' \rangle \leftarrow$ ProSaveWCET$(p'', S)$
26:        $w_{\mathrm{tot}} \leftarrow \max(w_{\mathrm{tot}}, w')$
27:        $S_{\mathrm{tot}} \leftarrow S_{\mathrm{tot}} \cup S'$
28:    return $\langle w + w_{\mathrm{tot}}, S_{\mathrm{tot}} \rangle$
29: else if $C$ is a control or with output port $p''$ then
30:    $\langle w', S' \rangle \leftarrow$ ProSaveWCET$(p'', S)$
31:    return $\langle w + w', S' \rangle$
```

The analysis is detailed in Algorithm 1. We denote by $\mathrm{WCET}(C)$ the WCET information associated with a service C of one of the subcomponents, or the WCET overhead associated with the type of connector C (fork, join, etc.). For each service of the component under analysis, let p be the trigger port of the input port group, compute

$$\langle w, S \rangle = \text{ProSaveWCET}(p, \emptyset)$$

and associate the service with w in the produced WCET information for the composite.

A key challenge of the analysis is to handle nested combinations of connectors, in particular nested fork/join and selection/or constructs. The way in which this is handled in the algorithm is by propagation of state information both forward and backward as the control chains are traversed. The state is simply a set of ports denoting the input ports of join connectors that have been reached so far in the analysis. This prevents the analysis from including components after a join connector multiple times.

When analysis reaches a selection, it performs a separate analysis of each output alternative and use the maximum WCET from all branches as the overall result. Forks are handled in a similar way, but with the important difference that the resulting state from the first branch is used as input to the analysis of the second branch, etc. Also, the overall result is the sum of all branches, rather than the maximum.

Example

Figure 3 shows a composite ProSave component A consisting of four subcomponent and four connectors. All components involved consist of only a single service.

The analysis of A follows the triggering path through B,

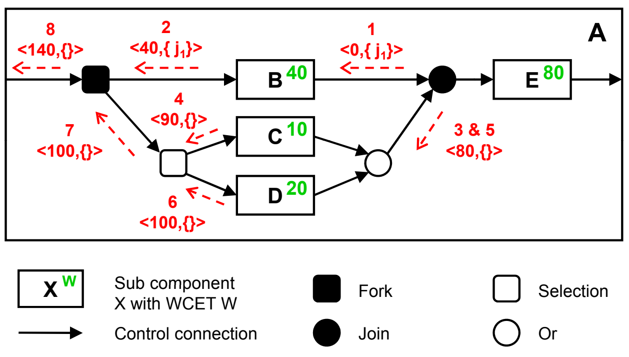

Figure 3: Example of analysis at ProSave level.

stops at port j$_1$ of the join connector. On the way back (steps 1 and 2), the WCET is accumulated and the state {j$_1$} is passed along to be used in the second fork branch. When the join connector is reached after traversing the selector and C, the state indicates that all ports of the join have been triggered, and thus the traversal continues through E and WCET is accumulated on the way back to the selection (3 and 4). The second path from the selection is explored using the same state as for the first one, thus resulting in the same traversal through the join and E (5). Returning to the selection (6), the maximum of the two alternatives is taken (7). Finally, the result of the fork connector is the sum of its branches, which is returned as the result of the whole analysis (8).

5. PROSAVE TO PROSYS ANALYSIS

The analysis bridging the gap between the two levels of the model is applied to primitive ProSys components, i.e., ProSys components that internally are constructed by interconnected ProSave components and connectors. Thus, the analysis algorithm is very similar to the one presented in the previous section. The most important difference is the format of the final analysis result, reflecting the difference between the analysed enclosing entities (a ProSave and a ProSys component, respectively).

Since ProSys components are not passively waiting to be triggered, but instead can contain multiple internal activities as well as activities triggered by incoming messages, the WCET of a ProSys component cannot be represented by a single integer. Instead, we need to represent the WCET associated with the arrival of messages to the different input ports, as well as WCET associated with internal periodic activities. In addition to this, we need information about how often messages are sent from the output ports, in order to perform analysis on enclosing levels of nesting.

Formally, the WCET information associated with a ProSys component C is represented by a tuple $\langle pw, mw\rangle$, where

pw	is a multiset of elements on the form $\langle t, w, o\rangle$;
mw	is a set of elements on the form $\langle p_i, w, o\rangle$;
t	represents a period ($t \in \mathbb{Z}^+$);
p_i	is an an input message port of C;
w	represents a WCET value ($w \in \mathbb{Z}^+$);
o	is a set of elements on the form $\langle p_o, m\rangle$;
p_o	is an an output message port of C; and
m	represents the number of messages ($m \in \mathbb{Z}^+$).

Algorithm 2 ProSaveProSysWCET(p)

```
if no connection $p \to p'$ exists then
    return $\langle 0, \emptyset\rangle$
Let $p'$ be the port for which the connection $p \to p'$ exists
if $p'$ is an output port of the analysed component then
    return $\langle 0, \{\langle p', 1\rangle\}\rangle$
Let $C$ be the construct (connector or subcomponent service) to which $p'$ belongs
$w \leftarrow \mathrm{WCET}(C)$
if $C$ is a subcomponent service or control fork then
    $o \leftarrow \emptyset$
    for each output port $p''$ of $C$ do
        $\langle w', o'\rangle \leftarrow$ ProSaveProSysWCET($p''$)
        $w \leftarrow w + w'$
        $o \leftarrow o + o'$
    return $\langle w, o\rangle$
else if $C$ is a control join or a control or with output port $p''$ then
    $\langle w', o'\rangle \leftarrow$ ProSaveProSysWCET($p''$)
    return $\langle w + w', o'\rangle$
else if $C$ is a selection then
    $w_{\mathrm{tot}} \leftarrow 0$
    $o_{\mathrm{tot}} \leftarrow \emptyset$
    for each output port $p''$ of $C$ do
        $\langle w', o'\rangle \leftarrow$ ProSaveProSysWCET($p''$)
        $w_{\mathrm{tot}} \leftarrow \max(w_{\mathrm{tot}}, w')$
        $o_{\mathrm{tot}} \leftarrow \max(o_{\mathrm{tot}}, o')$
    return $\langle w + w_{\mathrm{tot}}, o_{\mathrm{tot}}\rangle$
```

We also restrict the set mw to not contain two elements $\langle p_1, w_1, o_1\rangle$ and $\langle p_2, w_2, o_2\rangle$ such that $p_1 = p_2$. Thus mw can be viewed as a function from the input message ports of C to $\langle w, o\rangle$ tuples. Similarly, o is restricted to contain at most one occurrence of each port, and thus can be seen as a funtion, with $o(p) = m$ if $\langle p, m\rangle \in o$ and $o(p) = 0$ if p does not occur in any of the tuples in o.

As an example, a ProSys component C with one input port m_1 and output ports m_2 and m_3, could be decorated with the following WCET information:

$$\begin{aligned}\mathrm{WCET}(C) = \langle\{&\langle 200,\ 50,\ \{\langle m_2, 1\rangle\}\rangle\},\\ &\{\langle m_1,\ 20,\ \{\langle m_2, 1\rangle,\ \langle m_3, 2\rangle\}\rangle\}\rangle\end{aligned}$$

representing that there is an activity with period 200 and WCET 50 that sends at most one m_2 message, and that the processing of a m_1 message requires 20 units of WCET and sends at most one m_2 message and two m_3 messages.

Another change compared to the ProSave analysis is the state information needed during the traversal. Instead of keeping track of reached join connector ports, we now only need to propagate as a partial result the number of sent messages from the different output ports, in the format specified by o in the definition of the WCET information above. We define two operators over this information (using the function interpretation of the set, as described above):

$$\begin{aligned}(o_1 + o_2)\ (p) &= o_1(p) + o_2(p)\\ (o_1 \max o_2)\ (p) &= \max(o_1(p), o_2(p))\end{aligned}$$

Algorithm 2 defines the detailed analysis algorithm. For each clock of the analysed component, let t and p be the period and the output trigger port of the clock, compute

$$\langle w, o\rangle = \text{ProSaveProSysWCET}(p)$$

and add $\langle t, w, o\rangle$ to the pw part of the WCET info for the component. Then, for each input message port p, compute

$$\langle w, o\rangle = \text{ProSaveProSysWCET}(p)$$

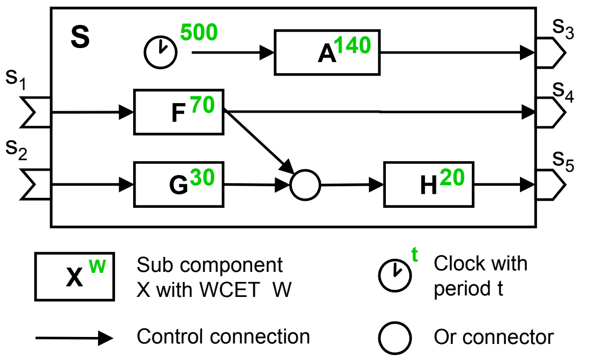

Figure 4: Example of analysis of a primitive ProSys component.

and add $\langle p, w, o\rangle$ to the mw part of the WCET info for the component.

Since the analysed enclosing entity is not a ProSave component service, as in the previous section, we cannot rely on the ProSave semantics of allowing at most a single active invocation at a time, and no mixing of control flow from different services. Consequently, join connectors must be treated differently in the analysis of a ProSys component, since in the worst case, the join ports are reached from different control chains.

As shown in Algorithm 2, join connectors are treated in the same way as or connectors, which results in a safe approximation. In cases where control chains of multiple periodic activities are joined, a tighter analysis would be possible, where the chain after the join is only included in the activity with longest period. However, when chains originating from message ports are involved, a tighter analysis becomes much more complex, and would require a more complex WCET information format.

Example

Figure 4 depicts a ProSys component S, consisting internally of ProSave components A (from the previous example), F, G and H. The analysis algorithm is called three times, for the clock and the two input message ports, respectively. Each result corresponds to a line in the following WCET information, which is the result of the whole analysis of S:

$$\begin{aligned}\textsc{wcet}(\mathsf{S}) = \langle\{&\langle 500,\ 140,\ \{\langle s_3, 1\rangle\}\rangle\},\\ \{&\langle\ s_1,\ \ 90,\ \{\langle s_4, 1\rangle,\ \langle s_5, 1\rangle\}\rangle,\\ &\langle\ s_2,\ \ 50,\ \{\langle s_5, 1\rangle\}\rangle\}\rangle\end{aligned}$$

6. PROSYS ANALYSIS

The third type of model-level WCET analysis is performed on a composite ProSys component. From information about the subcomponents, in the format defined in the previous section, and the interconnection structure, WCET information for the composite is derived.

Algorithm 3 defines the detailed analysis algorithm, and Algorithm 4 shows how this analysis is called multiple times to produce the full WCET information for the analysed composite ProSys component.

The main differences compared to the analysis in the previous section is that on one hand the interconnection structure is much simpler but on the other hand the WCET information of the subcomponents is much more complex. The

Algorithm 3 ProSysWCET(p, n)

```
1: Let $P$ be the set of ports $p'$ that have connections $p \to p'$
2: $w_{\text{tot}} \leftarrow 0$
3: $o_{\text{tot}} \leftarrow \emptyset$
4: for each $p'$ in $P$ do
5:     if $p'$ is an output port of the analysed component then
6:         $o_{\text{tot}} \leftarrow o_{\text{tot}} + \{\langle p', n\rangle\}$
7:     else
8:         Let $C'$ be the subcomponent to which $p'$ belongs
9:         $\langle pw, mw\rangle \leftarrow \textsc{wcet}(C')$
10:        $\langle w, o\rangle \leftarrow mw(p')$
11:        $w_{\text{tot}} \leftarrow w_{\text{tot}} + w * n$
12:        for each $\langle p'', m\rangle$ in $o$ do
13:            $\langle w', o'\rangle \leftarrow \text{ProSysWCET}(p'', n * m)$
14:            $w_{\text{tot}} \leftarrow w_{\text{tot}} + w'$
15:            $o_{\text{tot}} \leftarrow o_{\text{tot}} + o'$
16: return $\langle w_{\text{tot}}, o_{\text{tot}}\rangle$
```

Algorithm 4 mainProSysWCET(C)

```
1: $pw \leftarrow \emptyset$
2: for each subcomponent $C'$ of $C$ do
3:     $\langle pw', mw'\rangle \leftarrow \textsc{wcet}(C')$
4:     for each $\langle t, \langle w, o\rangle\rangle$ in $pw'$ do
5:         $w_{\text{tot}} \leftarrow w$
6:         $o_{\text{tot}} \leftarrow \emptyset$
7:         for each $\langle p, n\rangle$ in $o$ do
8:             $\langle w', o'\rangle \leftarrow \text{ProSysWCET}(p, n)$
9:             $w_{\text{tot}} \leftarrow w_{\text{tot}} + w'$
10:            $o_{\text{tot}} \leftarrow o_{\text{tot}} + o'$
11:        $pw \leftarrow pw \cup \langle t, \langle w_{\text{tot}}, o_{\text{tot}}\rangle\rangle$
12: $mw \leftarrow \emptyset$
13: for each input message port $p$ of $C$ do
14:     $\langle w, o\rangle \leftarrow \text{ProSysWCET}(p, 1)$
15:     $mw \leftarrow mw \cup \langle p, \langle w, o\rangle\rangle$
16: return $\langle pw, mw\rangle$
```

detailed analysis is quite straightforward, but the second argument might require some explanation. This integer represents the number of messages that can arrive to the port that the analysis has reached, and acts as a multiplier applied to the WCET and to the output messages associated with the output ports when we continue the traversal.

Example

Figure 5 shows an example of a composite ProSys component to be analysed. The numbers and the dashed arrows illustrate the WCET information associated with the subcomponents.

Subcomponent S is the component from the previous example (the ports are relocated to simplify the connections), and we assume the following WCET information for the other subcomponent T:

$$\begin{aligned}\textsc{wcet}(\mathsf{T}) = \langle\{&\langle 200,\ 20,\ \{\}\rangle\},\\ \{&\langle\ t_1,\ 40,\ \{\}\rangle,\\ &\langle\ t_2,\ 50,\ \{\langle t_3, 2\rangle\}\rangle\}\rangle\end{aligned}$$

The analysis of U starts by analysing the periodic activities in the subcomponents. The activity in S results in 140 WCET and a s_3 message, which in turn adds 40 to the WCET when following the connection to t_1. The activity in T does not result in any output messages, only 20 WCET. None of the periodic activities result in any output messages from U. Next, the effects of an incoming u_1 message is analysed. Traversing the chains of messages through the subcomponents and collecting WCET along the way results

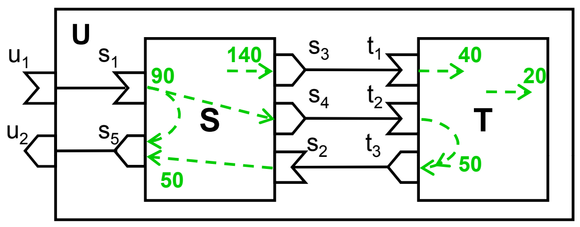

Figure 5: Example of ProSys level analysis.

in 240 WCET and three u_2 messages. Note that the two t_3 messages send in response to t_2 result in the multiplier 2 begin applied to the WCET and the messages in the analysis of s_2. Thus, the resulting WCET information for the analysed component U is:

$$\begin{aligned}\textsc{wcet}(\mathsf{U}) = \langle &\{\langle 500,\ 180,\ \{\}\rangle, \\ &\langle 200,\ \ 20,\ \{\}\rangle\}, \\ &\{\langle\ u_1,\ 240,\ \{\langle u_2, 3\rangle\}\rangle\}\rangle\end{aligned}$$

7. RELATED WORK

Puschner et al. [7] describe compositional WCET analysis in a more general context of hierarchical development processes, and addressing compositionality of WCET information. Contrasting our approach, they primarily consider the composition of tasks into systems, and not composition of sub-task design entities.

A new challenge for WCET analysis is analysis of code generated from high-level models. Kirner et al. [4] adjust the code generation process for Matlab/Simulink to produce code decorated with additional information, such as loop bounds, to permit automated WCET analysis. A similar approach, but targeting AADL, is taken by Gilles and Hugues [3].

Schedulability analysis, i.e., ensuring that all timing requirements in a system are satisfied also in the worst case scenario, is a well researched area [6]. Of particular importance to our work is the recent interest in hierarchical scheduling, and the related schedulability analysis, as a means to perform analysis on individual subsystems in isolation without full knowledge of the entire system [9, 2].

We have previously addressed parts of the model-level WCET analysis for ProCom [5]. That work was restricted to analysis composite ProSave components (the same scope as the analysis in Section 4), but is was done in the context of parametric WCET analysis, where WCET is represented by a formula over the input data ports rather than by a single value.

8. CONCLUSION

We have outlined the timing analysis envisioned for ProCom, ranging from code analysis of individual primitive components, to the response time analysis of individual virtual nodes in isolation. In particular, we have focused on model-level WCET analysis bridging the gap between these two, providing detailed descriptions of the three analysis methods needed to address all levels of the ProCom hierarchy.

Making the analysis fully compositional means improved scalability, a possibility to analyse parts of the system in early stages of development, and also facilitates reuse of an analysis results. However, this compositional approach comes at the cost of reduced precision compared to both monolithic model-level analysis and analysis of generated code. To some extent, this trade-off is inevitable, but the effect can be reduced by applying a flattening transformation to some or all parts of the model before analysis is carried out.

One line of future work is to develop a more flexible chain of analysis methods where the lines between code and model-level analysis are less strict. For example, results from model-level value range propagation analysis (bounding the possible range of component ports based on the connections and abstractions of component functionality) can be used to tighten the timing analysis on code level. We are also investigating how these analysis methods, developed in the context of an academic component model for embedded systems, can be adjusted to state-of-practice industrial languages and tools.

Acknowledgments

This work was supported by the Swedish Foundation for Strategic Research via the strategic research centre PROGRESS, and by the Swedish Research Council project CONTESSE (2010-4276).

9. REFERENCES

[1] J. Carlson, J. Feljan, J. Mäki-Turja, and M. Sjödin. Deployment modelling and synthesis in a component model for distributed embedded systems. In *36th Euromicro Conference on Software Engineering and Advanced Applications*, pages 74–82. IEEE, 2010.

[2] R. Davis and A. Burns. Resource sharing in hierarchical fixed priority pre-emptive systems. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 257–267, December 2006.

[3] O. Gilles and J. Hugues. Applying WCET analysis at architectural level. In *8th International Workshop on Worst-Case Execution Time (WCET) Analysis*, 2008.

[4] R. Kirner, R. Lang, G. Freiberger, and P. Puschner. Fully automatic worst-case execution time analysis for Matlab/Simulink models. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, pages 31–40. IEEE Computer Society, 2002.

[5] T. Leveque, E. Borde, A. Marref, and J. Carlson. Hierarchical composition of parametric WCET in a component based approach. In *14th IEEE International Symposium on Object/Component/ Service-oriented Real-time Distributed Computing*. IEEE, March 2011.

[6] J. W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[7] P. Puschner, R. Kirner, and R. G. Pettit. Towards composable timing for real-time programs. In *1st International Workshop on Software Technologies for Future Dependable Distributed Systems*, pages 1–5. IEEE, 2009.

[8] 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*. Springer, 2008.

[9] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Transactions of Embedded Computer Systems*, 7:30:1–30:39, May 2008.