Timing analysis for a composable mode switch

Yin Hang, Hans Hansson
Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, SWEDEN
Email: young.hang.yin@mdh.se

***Abstract*—Component based software development (CBD) reduces development time and effort by allowing systems to be built from pre-developed reusable components. A classical approach to reduce embedded systems design and run-time complexity is to partition the behavior into a set of major system modes. In supporting system modes in CBD, a key issue is seamless composition of multi-mode components into systems.**

In addressing this issue, we have developed a mode switch logic and algorithm for component-based multi-mode systems. In this paper we introduce timing analysis for our composable mode switch.

***Index Terms*—component-based, mode switch, timing analysis**

I. Introduction

Partitioning system behaviors into different operational modes is a frequently used approach to reduce complexity of system design and verification, as well as to increase efficiency in system execution. Typically, for each mode a different set of subsystems are executing.

We have developed a mode switch approach for component-based software [1], in which we consider component-based systems built by hierarchically organized components. If multiple modes are supported, some components may reconfigure themselves during mode switch in order to provide different functionalities. Figure 1 illustrates the component hierarchy of a simple multi-mode system (used throughout this paper). The system supports two operational modes: M_1 and M_2. At top level, the system consists of components *a* and *b*. Component *a* consists of components *c*, *d* and *e*. However, Component *d* is deactivated (not in use) in mode M_2. Similarly, Component *b* has two subcomponents: *f* and *g* (*g* is deactivated in M_1). As *a* and *b* both have subcomponents, we call them *composite components*, and we call components that cannot be further decomposed (e.g., *c* and *d*) *primitive components*.

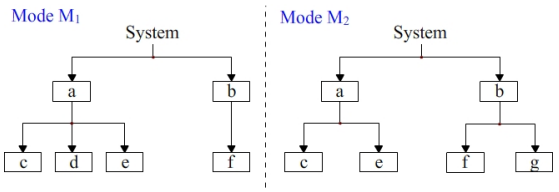

Fig. 1. Component hierarchy in different modes

Related research, includes mode switch protocols [2] and schedulability analysis during mode switch [3], as well as exploration of mode switch problems in CBD by various frameworks, including COMDES-II [4] and MyCCM-HI [5]. However, none of them comes up with a general mode switch logic (MSL) guiding the reconfiguration of hierarchically composed components. In this paper, we provide an overview of our MSL for multi-mode systems, explaining how the component reconfiguration is implemented. The contribution of this paper is that we introduce a timing analysis for mode switch in a component-based system using our MSL.

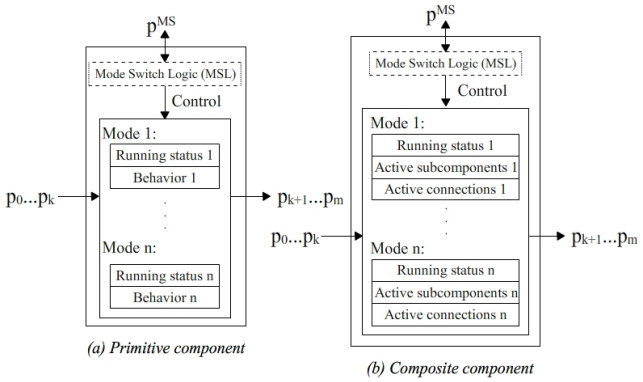

Fig. 2. Multi-mode components

II. The mode switch mechanism

To be compatible with our MSL, a component must be equipped with explicit interfaces related to mode switching and it must internally integrate certain rules to control its own mode switch process.

Figure 2 illustrates multi-mode primitive and composite components. Each component has one or more input and output ports, including a dedicated port p^{MS} for sending/receiving Mode Switch Requests (*MSR*s) and other mode switch related messages. The configuration of a primitive component consists of its running status (*activiated* or *deactivated*) and mode-specific behavior/code. The configuration of a composite component consists of its running status, activated subcomponents, connections in use between ports of its subcomponents and connections in use between its own ports and the ports of its subcomponents.

As an illustration, Figure 3 extends the example in Figure 1 with component connections. The sample system gets data from the input, processes data and generates output. The flow of data is indicated by arrows.

The mode switch must be performed such that severe problems and anomalies (e.g. mode or data inconsistency and mode switch failure) are avoided. Our MSL is designed to eliminate these potential problems.

We will in this paper make the following simplifying assumptions (which we intend to weaken in our future work):

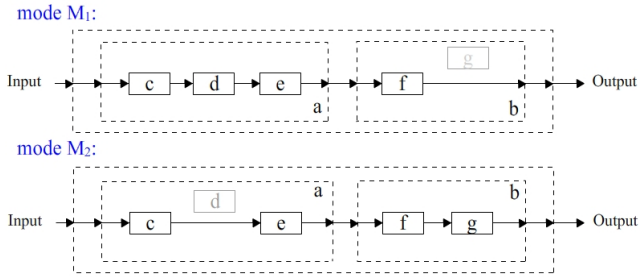

Fig. 3. System overview illustrating component connections

- The execution of primitive components can be aborted at any time (to allow immediate response to a *MSR*).
- All components support the same modes (to avoid the need for a mode mapping mechanism).

Also, note that the reconfiguration time of components is independent of interference of other components, since we assume a separate schedulability analysis handling this issue, i.e., in scheduling terms the reconfiguration time is a "response time".

Our MSL consists of a *MSR* propagation mechanism and dependency rules.

Mode switch request propagation: A *MSR* is initially issued by one of the primitive components. The *MSR* is then propagated to other components until all the components get notified. All composite components propagate the *MSR* to their subcomponents and parents.

Let's demonstrate the MSR propagation mechanism using the example in Figure 3. Suppose the *MSR* is initiated from Component *c* in both modes. Component *c* is primitive and propagates the *MSR* to its parent *a* and itself to trigger mode switch. Once Component *a* receives the *MSR*, it propagates it in two directions: to its subcomponents *d* and *e* and to its parent, which is the top level component. Since the subcomponents *d* and *e* are primitive, there is no further MSR propagation from them. The top level component has no parent, and will only propagate the *MSR* to its subcomponent *b*, which only needs to propagate the *MSR* to its subcomponents *f* and *g*. Since *f* and *g* are primitive, the MSR propagation is finally terminated. Once a component completes its MSR propagation, it can start its own mode switch process.

Dependency rules: The mode switch completion of a component have the following dependency on other components:

A. A composite component cannot complete its mode switch before the completion of the corresponding mode switch in its subcomponents.

B. A component cannot complete its mode switch before the mode switch completion of all components with the same parent connected to its ingoing ports. This is called the *forward dependency rule*, particularly made for pipes-and-filters type of systems. Other dependency rules, adapted to other types of systems/component are also possible, but will not be considered here.

C. For components with parents, Rule B cannot be applied until the parent has updated the component connections for the new mode.

Components that cannot proceed with the mode switch due to a dependency rule are temporarily blocked until the corresponding condition is satisfied. Here "blocked" means that a component is waiting for a message from other components.

Algorithm 1 and 2 describe the mode switch processes of primitive and composite components respectively, implementing the MSR propagation mechanism and dependency rules. Regarding these two algorithms, a few points should be mentioned:

- *MSR* is the mode switch request signal, carrying the identity of the new mode and the sending component.
- $parentOK$ is a signal from a composite component used to tell its subcomponents that its reconfiguration is completed. It may include a request for a response upon mode switch completion.
- ms_done is used to signal completion of mode switch and the ms_done transmission is based on the dependency rules.
- Reconfiguration means that a component changes its configuration in the current mode to the configuration in the new mode.

Details of the algorithms can be found in [1].

Algorithm 1 $PrimitiveComponent.mode\_switch$

loop
  Wait for MSR*;*
  Reconfiguration;
  Wait for parentOK;
  ms_done transmission;
  Execute in the new mode;
end loop

Algorithm 2 $CompositeComponent.mode\_switch$

loop
  Wait for MSR*;*
  MSR propagation;
  Reconfiguration;
  Broadcast parentOK;
  Wait for parentOK if not top level;
  ms_done transmission;
end loop

III. Mode switch timing analysis

For real-time embedded systems supporting multiple modes, not only is the correctness of the mode switch important, but also the time it takes for the system to complete a mode switch. We have successfully verified the correctness of our MSL using the UPPAAL model checker [6]. Here we will provide an analytical model for the mode switch timing analysis of component-based multi-mode real-time systems.

We will analyze the global mode switch time, i.e., the time from initial triggering of the mode switch to system-wide completion of the mode switch. We divide the global mode switch into three phases:

A. MSR propagation
B. Component reconfiguration
C. Mode switch completion

The *MSR* propagation starts when the triggering source initiate the mode switch and ends when all components have received the *MSR*. Due to the *MSR* propagation delay, the reconfiguration starting times of different components may be different. The reconfiguration phase ends when all components have completed their reconfigurations. The reconfiguration completion times (RCTs) can be calculated for each component and end of reconfiguration will then simply be the largest such time. Mode switch completion starts from the completion of component reconfiguration and ends when the top component receives an ms_done message, which is a confirmation that all components are executing in the new mode.

A. The timing analysis in the MSR propagation phase

Let DL_i denote the depth level of Component i in the component hierarchy (cf. Figure 1) defined such that it is 0 for the top level component and $j+1$ for the children of a component with depth j.

The *MSR* propagation time is based on the number of transmitted *MSR* messages. In calculating the *MSR* propagation time we use the following notation: components *x*, *y*, and z are the triggering source, the target component, and the closest common ancestor of *x* and *y*, respectively.

In propagating the *MSR* from x to y the *MSR* is first propagated upwards in the component hierarchy to z and then downwards from z to y. Assuming a fixed transmission time t_{MSR} for all *MSR* messages, the total propagation time t^{xy}_{total} can be calculated by the following equation:

$$t^{xy}_{total} = t_{MSR} * (DL_x - DL_z + DL_y - DL_z)$$

B. The timing analysis in the component reconfiguration phase

In the component reconfiguration phase, different components reconfigure themselves in parallel. Due to the dependency rules in our MSL, some components which complete their reconfigurations earlier than other components may be blocked by the reconfiguration of other components. There are two blocking factors. One is the need to wait for the $parentOK$ message from the parent, indicating that the component connections have been updated. The other is waiting on the ms_done message either from a neighboring component or from a subcomponent. indicating that the mode switch of the sender is completed.

For each component, its Reconfiguration Completion Time (RCT) is calculated as the sum of its *MSR* propagation delay and its reconfiguration time. The RCTs of different components are then compared with each other from the bottom level to the top level. The component with the largest RCT, corresponding to the end of reconfiguration, is thus identified.

C. The timing analysis in the mode switch completion phase

In phase *B* we have identified the component *w* with largest RCT. For the other components, some have completed their mode switches while the rest are all blocked by *w* directly or indirectly. We call this chain blocking. At the beginning of the final phase, all blocked components are waiting for the ms_done message, either from their neighboring components or subcomponents. Three different scenarios of ms_done message transmission must be considered:

1) If Component *w* is composite, all its subcomponents are directly blocked by its $parentOK$ message. Therefore, the ms_done message transmission between all its subcomponents must be considered. Apart from that, the $parentOK$ message sent by *w* is also considered as it is transmitted after its reconfiguration.
2) Among the components with the same parent as *w*, the ms_done transmission between some of them may be blocked by *w* due to the forward dependency rule.
3) Similar to Scenario 2, the parent and ancestor components of *w* may also block the ms_done transmission of other components with the same parents due to the forward dependency rule.

Since these ms_done messages must be transmitted in a sequential order due to chain blocking, the transmission time from w to the top level component could be substantial. In essence, the timing analysis in the final phase boils down to calculating the number of transmitted ms_done messages. In the calculation we will use the recursive function $Calculate(current\_C, DL)$ described in Algorithm 3. It begins by counting the number of active output ports ($n_{AOP}(current\_C)$) of Component *current_C*, and then follows those active output ports ($AOP(n)(current\_C)$) and finds the next component (by calling the $GetNext$ function). Each active output port corresponds to one ms_done message transmission. The calculation terminates when the current component is requested to send a response back to the parent, i.e. $last(current\_C) = true$. (We assume that only one component is asked to send back this response and this component blocks no other components at the same depth level). Function $Calculate(current\_C, DL)$ finally returns the total number of transmitted ms_done messages at depth level DL in the final phase.

To calculate the total ms_done transmission time t_{total} in the final phase at all related depth levels we use the integer array $n[1..DL_w + 1]$ to store the number of message at each depth level. The calculations are realized by Algorithm 4, where the following additional notations are used: $first\_C(DL)$ denotes the component with no ingoing components at Depth Level DL and (by the forward dependency rule) this component will not be blocked by other components at the same depth level (the case with multiple such components is not considered in this paper); t is the time spent transmitting one ms_done message; n_{total} is the total number of ms_done messages transmitted in the final phase; and $w(i)$ denotes the ancestors of w at depth i (or also w itself when $w = w(i)$ as a special case).

Please note that if a system has more than one potential mode switch triggering source, the timing analysis must be

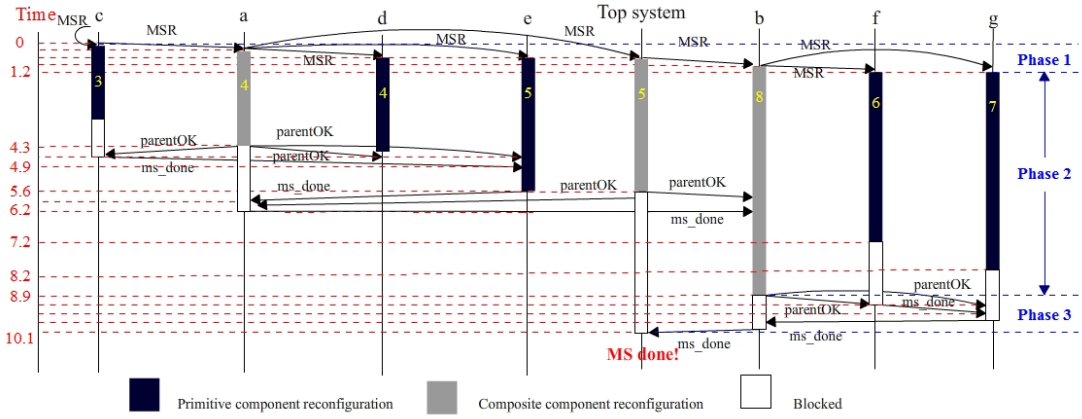

Fig. 4. Global mode switch timing analysis demonstration (from M_1 to M_2)

Algorithm 3 $Calculate(current\_C, DL)$

if $last(current\_C)$ **then**
  $n[DL] += 1$;
else
  for $i = n_{AOP}(current\_C)$ **downto** 1 **do**
    $next = GetNext(AOP(i)(current\_C))$;
    $n[DL] += Calculate(next, DL)$;
  end for
end if
return $n[DL]$;

Algorithm 4 *ms_done transmission time calculation*

if $Type(w(DL_w)) = composite$ **then**
  $n[DL_w + 1] = Calculate(first\_C(DL_w{+}1), DL_w{+}1)$;
end if
for i *from* DL_w **downto** 1 **do**
  $n[i] = Calculate(w(i), i)$;
end for
$n_{total} = \sum_{i=1}^{DL_w+1} n[i]$;
$t_{total} = t * n_{total}$;

repeated for all triggering sources, since the mode switch times will probably be different.

Figure 4 presents the mode switch from M_1 to M_2 of the example introduced in Figure 1. Component reconfiguration time is marked on the bars. The message transmission for *MSR*, $parentOK$ and ms_done are all assumed to be 0.3 time units. Component *b* has the largest RCT (8.9) and the mode switch time from triggering to system-wide completion is 10.1.

IV. Discussion and Future Work

We have presented a Mode Switch Logic (MSL) for component-based systems with multiple operational modes, for which we introduced an analysis to calculate the time to perform a global mode switch.

There are several important issues that we intend to address in our continued work. We intend to lift the restrictions in the considered setup with the aim to provide a MSL and timing analysis that is fully applicable in a real industrial setting. For instance, it is unrealistic to assume that a primitive component can terminate its execution at any time due to mode switch. Lifting this assumption may increase mode switch latency and must thus be analyzed. Also, in the final phase of the global mode switch, our timing analysis assumes the forward dependency rule, which could be rather inefficient in some situations by requiring blocking of a large number of components. Other dependency rules should be explored so that the one that minimizes the global mode switch time can be selected. In addition, we are currently focusing on pipes-and-filters systems, but intend to look into mode switch in other types of systems. Finally, we intend to analyze the time complexity and scalability of our algorithms.

Acknowledgment

This work is supported by the Swedish Research Council.

References

[1] Y. Hang, E. Borde, and H. Hansson, "Composable mode switch for component-based systems," in *APRES '11: Third International Workshop on Adaptive and Reconfigurable Embedded Systems*, 2011, pp. 19–22.

[2] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Systems*, vol. 26, no. 2, pp. 161–197, 2004.

[3] P. Pedro and A. Burns, "Schedulability analysis for mode changes in flexible real-time systems," in *Euromicro Conference on Real-Time Systems*, 1998, pp. 172–179.

[4] X. Ke, K. Sierszecki, and C. Angelov, "COMDES-II: A component-based framework for generative development of distributed real-time control systems," in *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007)*, 2007.

[5] E. Borde, G. Haïk, and L. Pautet, "Mode-based reconfiguration of critical software component architectures," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2009, pp. 1160–1165.

[6] K. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *STTT-International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 134–152, 1997.