Towards Mode Switch Handling in Component-based Multi-mode Systems

Yin Hang, Jan Carlson, Hans Hansson
Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
{young.hang.yin, jan.carlson, hans.hansson}@mdh.se

ABSTRACT

Component-based software engineering (CBSE) is becoming a prominent solution to the development of complex embedded systems. Meanwhile, partitioning system behavior into different modes is an effective approach to reduce system complexity. Combining the two, we get a component-based multi-mode system, for which a key issue is its mode switch handling. The mode switch of such a system corresponds to the joint mode switches of many hierarchically organized components. Such a composable mode switch is not trivial as it amounts to coordinate the mode switches of different components. In this paper, we identify the major challenges of the composable mode switch handling and classify existing approaches with respect to how they handle these challenges. We also provide a more detailed presentation of the corresponding solutions included in our approach – the Mode Switch Logic (MSL).

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Real-time and embedded systems*; D.2.13 [**Software Engineering**]: Reusable Software

Keywords

component-based, multi-mode, mode switch

1. INTRODUCTION

Component-based software engineering (CBSE) provides a promising design paradigm for the development of complex embedded systems. CBSE boasts quite a number of appealing features, such as complexity management, increased productivity, higher quality, faster developing time, lower maintenance costs and reusability [2]. A key purpose of CBSE is to allow systems to be built from reusable components, i.e. a system does not have to be developed from scratch, instead, some of its components or subsystems may be directly obtained from a repository of pre-developed components.

In contrast to CBSE, a common approach in embedded system design is to partition the system behavior into different operational modes. Each mode corresponds to a specific system behavior. Although some applications may run in all modes, there are also mode-specific applications, which run only in selected modes. A multi-mode system starts by running in a default mode and switches to another appropriate mode when some condition changes. A representative multi-mode system is the control software of an airplane, which could run in the modes *taxi* (the initial mode), *taking off*, *flight* and *landing*. Different sets of applications are running in different modes. For instance, the application for controlling the wheels only runs in *taxi* mode whereas the navigation application may run only in *flight* mode.

Traditional component models do not include handling of operational modes, whose traditional handling does not assume systems built from reusable components. To get the benefits of both approaches there is a need for techniques that in a predictable and consistent way integrate the two.

Our focus is on multi-mode systems developed in a component-based manner, called component-based multi-mode systems (MMS for abbreviation), assuming that the system is always component-based. Fig. 1 illustrates the hierarchical component structure of a typical MMS. The system consists of three components: *a*, *b* and *c*. Component *b* is composed of three other components: *d*, *e* and *f*. With respect to the terminology of CBSE, we distinguish two basic types of components: (1) a *primitive component*, whose behavior is given directly by associated software, thus cannot be further decomposed into other components; (2) a *composite component* which is a composition of other components. Obviously, in Fig. 1, *a*, *c*, *d*, *e* and *f* are primitive components whereas *Top* and *b* are composite components. Since the component hierarchy has a tree structure, a composite component and its subcomponents have a parent-and-children relationship. For instance, *b* is the parent of *d*, *e* and *f*, which in turn are the children of *b*. Moreover, the system supports two modes: $m1$ and $m2$. When the system is in mode $m1$, Component f is deactivated (i.e. not running), as Fig. 1 shows by not displaying f in mode $m1$. In contrast, when the system is in mode $m2$, f is activated whilst c and e are deactivated. Besides, Component a has different mode-specific behaviors represented by black and grey colors in Fig. 1. This example will be used throughout this paper.

From Fig. 1, it can be observed that an MMS exhibits

CBSE'12, June 26–28, 2012, Bertinoro, Italy.

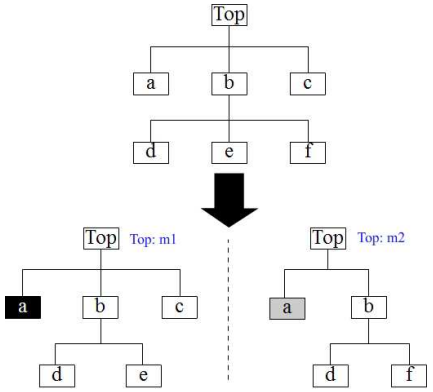

Figure 1: The component hierarchy of a component-based multi-mode system

unique system behavior in each mode reflected by particular configurations of its components at various levels. When an MMS switches from one mode to another mode, the switch is achieved by the joint mode switches of components composed together to form the complete system. We call this *composable mode switch*. We have developed a Mode Switch Logic (MSL) [4] guiding the composable mode switch of MMSs. In this paper, we shed light on the composable mode switch issue and identify related challenges. As a second contribution, we investigate and classify different existing approaches with respect to their mode switch handling. The last contribution of this paper is that it presents the solutions of our MSL to some major challenges of mode switch handling in MMSs. By integrating solutions to these challenges, our MSL provides guarantees for the correctness and consistency of mode switch in MMSs.

The rest of the paper is organized as follows: In Section 2, we investigate existing approaches to mode switch. In Section 3, challenges of composable mode switch are identified and existing approaches are classified accordingly. Section 4 presents our mode-aware component model. Section 5 discusses the mode incompatibility problem and mode mapping. Section 6 describes the mode switch runtime mechanism, and in Section 7 we make our conclusion.

2. EXISTING APPROACHES TO MODE SWITCH

There are hitherto few approaches supporting composable mode switch, including Rubus [7], AADL [3], COMDES-II [9] and MyCCM-HI [1]. These approaches are not dedicated to mode switch handling of MMSs, although we will only introduce them from the mode switch perspective.

Rubus: Rubus is an industrial component model targeting embedded control systems for ground vehicles [7]. In Rubus, mode is a system-level concept as different modes are typically defined at the top level through a mode/state transition diagram. In each mode, there is a system-wide static configuration of components. A mode switch of the system corresponds to the switch between different such configurations. Individual components are not mode-aware and it is up to the system integrator to integrate them into a multi-mode system. There is no published information about the Rubus mode switch handling at runtime.

AADL: the Architecture Analysis & Design Language [3], is a modeling language that supports analyses of a system's architecture with respect to performance-critical properties at the component level. AADL represents modes as states within a state machine abstraction, where each state corresponds to a distinct mode and the transitions between different states represent mode switch. For each component, a mode switch is triggered by a predefined mode switch event arriving at its input event port(s).

COMDES-II: COMponent-based design of software for Distributed Embedded Systems-version II, employs a hierarchical model to specify the system architecture [9]. The mode switch of a multi-mode component is controlled by a state machine. Component configuration in each mode in COMDES-II is similar to AADL (e.g. predefined modes and mode-specific behaviors), yet COMDES-II allows the composition of multi-mode components whereas, in AADL such compositions are not explicitly addressed.

MyCCM-HI: Make Your Component-Container Model-High Integrity, is a component framework for critical, distributed, real time and embedded software [1]. Components and their behaviors are described by the input architecture description language COAL (Component-Oriented Architecture Language), which supports enumerating different operational modes of the system and each component. Each multi-mode component is associated with a mode automaton component implementing its mode switch mechanisms. A composite component and its subcomponents can interact with each other during a mode switch via their mode automata components.

MSL – Our approach: None of the above presented approaches provide a systematic strategy to cope with the coordination of the mode switches of different components, while the mode switches of many components can often be interdependent. Our MSL circumvents this unexplored issue, by focusing on the management of interdependent mode switches. A component is not only aware of its modes and its configuration in each mode, but also able to communicate with its parent and children during a mode switch. Thus the system mode switch becomes a complex procedure involving many mode switches of different components with interdependency. The final goal is to guarantee the correctness of the mode switch of the entire system.

Other research into mode switch: The research on mode switch in real-time systems is relatively more productive. Sha et al. [13] have developed a simple mode switch protocol in a prioritized preemptive scheduling environment guaranteeing short and bounded mode switch latency. This protocol is improved by Tindell et al. [15] for periodic and sporadic tasks in a single processor system. Pedro and Burns [10] provide a simple mode switch model and improve schedulability during mode switch by adding proper offsets. Real and Crespo [12] conduct a survey of different mode switch protocols and propose several new protocols along with associated schedulability analysis. Phan et al. present a multi-mode automaton [11] model for modeling multi-mode applications. Moreover, mode switch can be supported by a few programming languages/models. Apart

from AADL, Giotto [8] and TDL [14] also support multi-mode and mode switch.

3. CLASSIFICATION OF MODE SWITCH APPROACHES

In this section we provide a classification of the introduced approaches to handling mode switch, as we will distinguish different approaches based on how they handle a set of major challenges related to composable mode switch.

3.1 Problems and challenges of composable mode switch

The handling of composable mode switch is not trivial, as it poses multiple potential problems and challenges. Many contributing factors need to be considered, such as component model, component hierarchy, component connection, system architecture and component execution pattern. The following are the major challenges that have to be addressed:

Component model

CBSE specifies that a software component must conform to a component model. A large number of component models have already been proposed targeting various application domains, e.g. AUTOSAR, COMDES-II, MyCCM-HI, ProCom and Rubus. Among these component models, only COMDES-II, MyCCM-HI and Rubus have multi-mode support. To compose multi-mode components in an MMS, traditional component models must be extended, since in an MMS a component must be aware of its own modes which could be switched at runtime. In our MSL, we have proposed a mode-aware component model taking into account both the mode switch of a single component and the mode switch synchronization between different components.

Multi-mode component composition

An MMS can be composed of both single-mode and multi-mode components. For a multi-mode component, its supported modes should also be composable. Since reusable components are developed independently, it is very likely that the supported modes and the number of supported modes differ between components. Hence, when a composite component is built by reusable components from a component repository, the supported modes of each component are likely to be inconsistent with the modes of the composed component. Therefore, the correspondence between the modes of different components must be defined during composition. Even though COMDES-II and MyCCM-HI both allow a multi-mode component to be composed into another multi-mode component, this mapping between modes is implicit. The mode mapping mechanism of our MSL explicitly addresses the mode mapping problem.

The mode switch runtime mechanism

The mode switch of an MMS must be carried out under the guidance of some runtime mechanism to ensure its correctness and efficiency. This is to a large extent neglected by existing mode switch handling approaches. Such runtime mechanism includes many aspects, such as

Mode switch propagation: A mode switch can be triggered by any component. Other components that change their configurations in the new mode must be informed of this mode switch. Since a component has no global knowledge of the system component hierarchy, a mode switch event cannot be simply broadcast from one component to all the other components. Instead, a stepwise propagation mechanism is required; by taking advantage of the hierarchical component structure, a mode switch event can be directly or indirectly propagated to any related component. Later we shall show how this can be achieved by our Mode Switch (MS) propagation mechanism.

The guarantee of consistent mode switch: The mode switch of a system may correspond to the mode switches of many components. When the system completes a mode switch, all its components must be in a consistent state. For instance, it must be avoided that components supposed to run in the new mode are still running in the old mode after the system has completed a mode switch. A correct MS propagation mechanism plays a significant role in notifying different components of the mode switch event, however, additional rules should be applied to guarantee the overall mode switch consistency. MyCCM-HI uses different mode switch protocols to keep system state consistency, yet this is not sufficient for component mode switches with interdependency. In our MSL, we use a mode switch dependency rule to guarantee consistent mode switch.

Mode switch and atomic execution: When a mode switch is triggered, the MMS is supposed to stop running in the old mode and start its mode switch as soon as possible. However, there could be any ongoing atomic execution in one or a set of components that cannot be aborted by a mode switch triggering. Our MSL handles atomic component execution by an extended MS propagation mechanism [6]. This is however separate work and in this paper we assume that component execution is not atomic.

Conflict handling for multiple mode switch triggering: Normally, an MMS performs a mode switch rather swiftly, yet not instantaneously. Multiple mode switch triggering could happen either simultaneously or within a short interval. Without proper treatment, an ongoing mode switch could be compromised by the triggering of a new mode switch. In our MSL, the initial solution is to use an arbitration mechanism managed by a component with high authority to resolve the conflict. Further details and the in-depth solution are included in our future work.

3.2 The Classification

Table 1 compares Rubus, AADL, COMDES-II, MyCCM-HI with our MSL in terms of the three main challenges addressed in this paper; namely component model, mode mapping, and runtime mechanism:

Component model: We consider if the approach supports multi-mode and if the mode switch of a component can be treated as a global or local activity. Only the Rubus component model is not multi-mode. In Rubus, mode switch is always a global activity as it is handled at the system level. In AADL and COMDES-II, mode switch is a local activity for an individual component. In MyCCM-HI and MSL, mode switch can be both global and local.

Mode mapping: This is not considered by Rubus and AADL. COMDES-II and MyCCM-HI implicitly support

	Component model		Mode mapping	Runtime mechanism	
	Multi-mode	Global/Local mode switch		Mode switch propagation	Consistency guarantee
Rubus	✗	Global	✗	?	?
AADL	✓	Local	✗	✗	✗
COMDES-II	✓	Local	Implicit	✗	✗
MyCCM-HI	✓	Global/Local	Implicit	✗	Little
MSL	✓	Global/Local	✓	✓	✓
✓: Supported; ✗: Not supported/considered; ?: Information not available					

Table 1: The comparison between different existing approaches and the MSL

mode mapping as they allow the composition of multi-mode components, but neither of them provides any mode mapping mechanism, which is an essential part of our MSL.

Runtime mechanism: This is hardly supported by existing approaches. The runtime mechanism of Rubus is not publicly known, since Rubus is commercial with limited available information. Mode switch propagation is only handled by MSL. AADL and COMDES-II do not have any runtime mechanism. MyCCM-HI uses mode switch protocols to achieve mode switch consistency. They are however insufficient for composable mode switch, since these protocols actually work for tasks while ignoring the component hierarchy and the mode switch interdependency between different components. MSL guarantees mode switch consistency by its mode switch dependency rule. The runtime mechanism of MSL is not only limited to mode switch propagation and consistency guarantee; other aspects covered, such as atomic component execution and conflicting handling of multiple mode switch triggering, will not be discussed in this paper.

It is obvious that MSL covers many mode switch issues that have not been tackled or even considered in related existing research.

4. COMPONENT MODEL

A mode-aware (or multi-mode) component is different from traditional components in terms of the interface, internal properties, and other aspects. We propose a mode-aware component model for both primitive and composite components. The mode-aware component model in our MSL is not based on any existing component model. Instead, it is rather generic and most existing component models can be extended to be mode-aware guided by the principles of MSL. Just like traditional component models, both primitive and composite components in MSL have a set of ports to communicate with other neighboring components. Furthermore,

- Both primitive and composite components support one or more modes. Each mode of a multi-mode component is associated with a unique mode ID.
- A primitive component has a dedicated mode switch port for the communication with its parent during a mode switch. A composite component has an additional dedicated port for the communication with its subcomponents/children.
- Both primitive and composite components have a separate configuration in each mode. Component configuration is a collection of mode-specific information such as the mode-specific functional behavior and mode-related properties of a component.

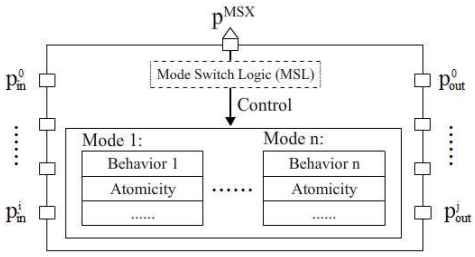

Figure 2: The mode-aware primitive component model

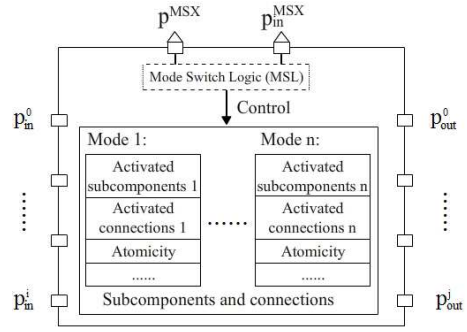

Figure 3: The mode-aware composite component model

- Component reconfiguration during a mode switch is controlled by the runtime mechanism of MSL integrated in the component.

Our mode-aware component model is illustrated in figures 2 and 3, from which the similarity and discrepancy of both primitive and composite component models can be easily perceived. Both figures assume the pipe-and-filter type of communication, as $p^0_{in} \cdots p^i_{in}$ denote the input ports and $p^0_{out} \cdots p^j_{out}$ denote the output ports. p^{MSX} and p^{MSX}_{in} are the dedicated mode switch ports, the former for communicating with the parent and the latter for communicating with the children. The configuration of primitive and composite components are different. For instance, a primitive component can have mode-specific behavior in each mode while a composite component may have different sets of activated subcomponents and connections in each mode. Fig. 4 illustrates the component connections of the example in Fig. 1. When the system is in $m1$, the connection between *do* and *ei* is activated, while this connection becomes deactivated and is replaced by a new activated connection between *do* and *fi* as the system switches to $m2$.

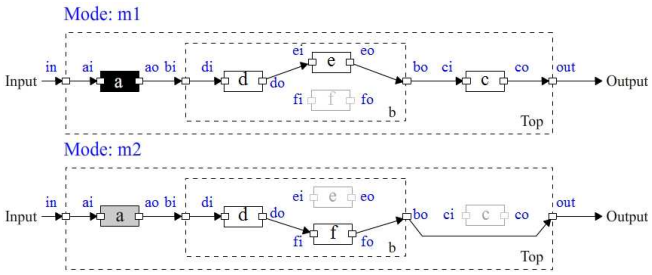

Figure 4: The component connection of a CBMMS

5. MODE MAPPING

As introduced in Section 3, the supported modes of different components may need to be mapped during the multi-mode component composition. As an illustration, consider Table 2 where the components have the same hierarchy as b in Fig. 1, but with different supported modes. Based on the system requirement, b is expected to support $m1b$ and $m2b$, which are different from the supported modes of any of its subcomponents.

When d, e and f are used to compose b, it must be clearly specified how their supported modes are mapped to each other. We call this *mode mapping*. Our mode mapping mechanism [5] is based on the following:

- Each component (primitive or composite) knows its supported modes, its initial mode and its current mode, but knows nothing about the mode information of other components in the system.
- Composite components know all the mode information of their subcomponents, but have no mode information of components at deeper nested levels.

These two properties enable a composite component to manage the local mode mapping between itself and its subcomponents. In [5], we propose a Mode Mapping Automata (MMA) managed by each composite component to handle these mappings.

Component	Supported modes
b	$m1b$, $m2b$
d	$m1d$, $m2d$, $m3d$
e	$m1e$
f	$m1f$, $m2f$, $m3f$, $m4f$

Table 2: The modes of b and its subcomponents

6. RUNTIME MECHANISM

As the principal ingredient of our MSL, the mode switch runtime mechanism synchronizes and coordinates the mode switches of different components to achieve a correct mode switch of all involved components. The mode switch starts when a mode switch decision is made and is terminated when all involved components have completed their mode switches. In this section, we look into this runtime mechanism with regard to mode switch propagation and mode switch dependency rule. For better presentation, we introduce the following notations. Let PC denote the set of primitive components and CC the set of composite components: $PC \cap CC = \emptyset$. The top component is denoted by Top. For $c_i \in CC$ ($c_i \neq Top$), P_{c_i} denotes the parent of c_i and SC_{c_i} the set of subcomponents of c_i. .

6.1 Mode switch propagation

In Section 3.1, we have stated that our MS propagation mechanism is able to distribute a mode switch event to all involved components. There are a variety of mode switch events, among which a typical one can be an alarm from a sensor whose value is above a threshold. A mode switch event is initially triggered by a Mode Switch Source (MSS) which could be any component, primitive or composite. An MSS is mode dependent and a system can have multiple MSSs in each mode. Let c_i denote an MSS with $c_j = P_{c_i}$. Our MS propagation mechanism works as follows:

MS propagation mechanism: *When c_i triggers a mode switch, it sends a Mode Switch Request (MSR) to c_j. As a composite component, c_j refers to the local mode mapping and makes a decision upon receiving the MSR:*

- *If the MSR does not imply any mode switch of c_j, then c_j approves the MSR by sending a Mode Switch Instruction (MSI) to SC_{c_j} based on its mode mapping.*
- *If the MSR implies the mode switch of c_j whose condition does not allow such a mode switch, then c_j rejects the MSR by doing nothing.*
- *If the MSR implies the mode switch of c_j whose condition allows such a mode switch and $c_j \neq Top$, then c_j will forward the MSR to P_{c_j} and let P_{c_j} make further decisions. If this MSR is finally approved, c_j will receive an MSI from P_{c_j} and send the MSI to SC_{c_j} based on its mode mapping.*

$\forall c_k \in CC$, c_k handles an incoming MSR or MSI exactly in the same way as c_j. $\forall c_l \in PC$, c_l can only receive an MSI but does not propagate the MSI. An MSS is not blocked after issuing an MSR. If Top is an MSS, it can directly issue an MSI to SC_{Top}. The mode switch propagation is terminated when all related components have received an MSI associated with the same MSR from an MSS.

where sending an MSI to SC_{c_j} means that the MSI is sent to the children of c_j for which the mode mapping results in a mode different than the current mode. The other children should not change mode, and hence, require no MSI. Fig. 5 depicts the relation between the local mode mapping and the mode switch propagation within Component b, referring to Table 2. The local mode mapping of b is presented by the MMAs. Due to the limited space, we skip the description of these MMAs which can be found in [5].

Apparently, each MSR will be eventually either approved or rejected by a component, called Mode Switch Decision Maker (MSDM). Depending on the location of an MSDM, mode switch propagation can be either a local or global activity. If Top is the MSDM that approves an upstream MSR, the mode switch is a global activity. Otherwise, it becomes a local activity within the MSDM.

6.2 Guaranteeing consistent mode switch

After MSI propagation, a component will start its reconfiguration. To guarantee that all components are in a consistent state after a mode switch, the mode switch of a system

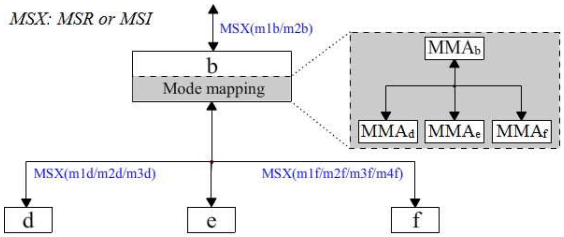

Figure 5: Mode mapping and MS propagation

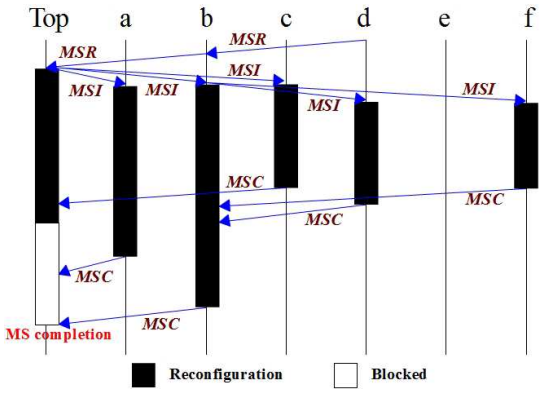

Figure 6: The complete mode switch process

cannot be completed until all reconfigurations are over. The reconfigurations of different components should be properly synchronized, guided by our mode switch dependency rule, which works as follows:

Mode switch dependency rule: *Each component starts its reconfiguration after its MSI propagation.* $c_i \in PC$ *sends a Mode Switch Completion (MSC) signal to* P_{c_i} *upon completion of its reconfiguration to indicate mode switch completion.* $c_j \in CC$ *completes its mode switch when its reconfiguration is completed and it has received the MSC from all* $c_k \in SC_{c_j}$ *that* c_k *has previously received an MSI from* c_j*. Thereafter,* c_j *sends an MSC to* P_{c_j} *if* $c_j \neq Top$ *and* c_j *is not the MSDM. A mode switch is completed when the corresponding MSDM completes its mode switch.*

The mode switch dependency rule above guarantees that a composite component always completes its mode switch after its subcomponents. Fig. 6 demonstrates the entire mode switch process of the example in Fig. 1, assuming (1) d is the MSS issuing an MSR; (2) this MSR leads to the mode switches of all components except e, hence Top is the MSDM. Since e does not switch mode in this scenario, b does not send any MSI to e. Component reconfiguration is illustrated by black bars. The top component has a short reconfiguration time, thus its mode switch is temporarily blocked by the MSC from its subcomponents a and b.

7. CONCLUSION AND FUTURE WORK

In this paper, we have introduced a component-based multi-mode system (MMS), which not only supports multiple operational modes but also reusable (multi-mode) components. Some representative existing approaches have been studied for handling the composable mode switch of MMSs, from which we have identified a number of challenges, such as the mode-aware component model, the mode incompatibility problem during composition and the mode switch runtime mechanism. Based on these challenges, we classify the existing approaches and compare them with our Mode Switch Logic (MSL). Moreover, we present pertinent solutions in our MSL to some major challenging problems of composable mode switch.

To date we have addressed several important problems related to composable mode switch. It is our ambition to implement our MSL into state-of-the-art component models such as ProCom [16]. Furthermore, we plan to evaluate the suitability of our MSL by an industrial case-study.

Acknowledgments

This work is supported by the Swedish Research Council.

8. REFERENCES

[1] E. Borde, G. Haïk, and L. Pautet. Mode-based reconfiguration of critical software component architectures. In *DATE'09*, pages 1160–1165.

[2] I. Crnkovic and M. Larsson. *Building reliable component-based software systems*. Artech House, 2002.

[3] P. H. Feiler, D. P. Gluch, and J. J. Hudak. The architecture analysis & design language (AADL): An introduction. Technical Report CMU/SEI-2006-TN-011, Software engineering institute, MA, Feb. 2006.

[4] Y. Hang, E. Borde, and H. Hansson. Composable mode switch for component-based systems. In *APRES '11*, pages 19–22.

[5] Y. Hang and H. Hansson. A mode mapping mechanism for component-based multi-mode systems. In *CRTS'11*, pages 38–45.

[6] Y. Hang and H. Hansson. Timing analysis for mode switch in component-based multi-mode systems. In *ECRTS'12 (To appear)*.

[7] K. Hänninen, J. Mäki-Turja, M. Nolin, M. Lindberg, J. Lundbäck, and K.-L. Lundbäck. The Rubus component model for resource constrained real-time systems. In *SIES'08*, pages 177 –183, june.

[8] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: A time-triggered language for embedded programming. In *Proceedings of the IEEE*, pages 166–184, 2001.

[9] X. Ke, K. Sierszecki, and C. Angelov. COMDES-II: A component-based framework for generative development of distributed real-time control systems. In *RTCSA'07*.

[10] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *ECRTS'98*, pages 172–179.

[11] L. T. X. Phan, I. Lee, and O. Sokolsky. Compositional analysis of multi-mode systems. In *ECRTS'10*, pages 197–206.

[12] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161–197, 2004.

[13] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1:243–264, 1989.

[14] J. Templ. TDL specification and report. Technical report, Department of Computer Science, University of Salzburg, Nov. 2003.

[15] K. W. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority pre-emptively scheduled systems. In *RTSS'92*, pages 100–109.

[16] A. Vulgarakis, J. Suryadevara, J. Carlson, C. Seceleanu, and P. Pettersson. Formal semantics of the ProCom real-time component model. In *SEAA'09*, pages 478–485.