Component-based software development of multi-mode systems — An extended report

Hang Yin  
Chalmers University of Technology, Gothenburg, Sweden  
yhang@chalmers.se

Hans Hansson  
Mälardalen University, Västerås, Sweden  
hans.hansson@mdh.se

Daniele Orlando  
University of L’Aquila, L’Aquila, Italy  
me@danieleorlando.com

Francesco Miscia  
University of L’Aquila, L’Aquila, Italy  
frac.miscia@gmail.com

Simone Di Marco  
University of L’Aquila, L’Aquila, Italy  
simonedimarco89@gmail.com

November 13, 2016

Abstract  
Growing software complexity is an increasing challenge for the software development of modern embedded systems. A classical strategy for taming the software complexity is to partition system behaviors into different operational modes specified at design time. Such a multi-mode system can change behavior by switching between modes at runtime. Component-Based Software Engineering (CBSE) is a complementary approach to the software development of complex systems that fosters reuse of independently developed software components. CBSE and the multi-mode approach are fundamentally conflicting in that component-based development conceptually is a bottom-up approach, whereas partitioning systems into operational modes is a top-down approach. In this report we show that it is possible to combine and integrate these two fundamentally conflicting approaches. The key to simultaneously benefitting from the advantages of both approaches lies in the introduction of a hierarchical mode concept that provides a conceptual linkage between the bottom-up component-based approach
and system level modes. As a result, systems including modes can be developed from reusable mode-aware components in the modeling phase. The conceptual drawback of the approach—the need for extensive message exchange between components to coordinate mode switches—is eliminated by an algorithm that collapses the component hierarchy and thereby eliminates the need for inter-component coordination. As this algorithm is used from the design to implementation level ("compilation"), the CBSE design flexibility can be combined with efficiently implemented mode handling. At the more specific level, this report presents (1) a mode mapping mechanism which formally specifies the mode relation between composable multi-mode components, (2) a mode transformation technique that transforms component modes to system-wide modes to achieve efficient implementation, and (3) a prototype tool that implements the mode mapping mechanism and mode transformation technique.

1 Introduction

Growing software complexity is posing a challenge for the software design of embedded systems. A common practice to manage software complexity is to partition system behaviors into different mutually exclusive operational modes so that each mode corresponds to a specific system behavior. Such a multi-mode system [1] usually runs in one mode and can switch to another mode under certain conditions. For instance, the control software of an airplane could run in the modes taxi (the initial mode), taking off, flight and landing. Multi-mode systems have also been motivated by many other reasons:

1. More efficient software design, testing, and verification. A multi-mode system allows for the separate design and parallel testing of different modes, thus facilitating system development.

2. Diversity of system functionality. A multi-mode system exhibits distinctive functionalities in different modes. Therefore, it is usually easier for a multi-mode system to provide more diversified services compared to a single-mode system.

3. Adaptivity. A multi-mode system can be considered as a type of adaptive system that actively adjusts its behavior and performance to accommodate new conditions. For instance, an adaptive media player of an embedded device with constrained resource may switch between the degraded Quality of Service (QoS) mode and normal mode, depending on available runtime resources.

4. Saving resources. For those systems with a lot of tasks which only need to run under certain conditions, the constant running of all tasks
would be a waste of resources. It is more efficient to deactivate the
tasks that are not currently used in the system. Different modes could
be specified based on the set of running tasks.

5. Fault tolerance. Most safety-critical systems are designed to be fault-
tolerant so as to reduce the risks for catastrophic consequences. A
practical fault-tolerant strategy is to switch to a safe mode in case a
fault occurs.

6. More precise analysis of system properties. Many system properties
are associated with a wide range of values, however, the maximum
and minimum could have extremely low probability of occurrence. A
typical example is the Worst-Case Execution Time (WCET) of a task
in a real-time system. The WCET calculation is typically rather pes-
simistic because it is often much larger than the average-case execution
time. If a task runs in multiple modes, its WCET can be calculated
or measured for each mode, thereby yielding more precise WCET es-
timates.

7. Extensibility and scalability. For a single-mode system, adding a new
function risks polluting the software structure of the original system
and necessitates the test of the complete new system. Alternatively, if
a new mode is introduced for the new function, the system behavior in
existing modes will not be affected. Additional development and test
effort is limited to the new mode and the mode switch from or to this
new mode. This also implies good scalability of a multi-mode system.

As a complementary approach to multi-mode systems, Component-
Based Software Engineering (CBSE) [2] advocates the reuse of independently
developed software components as a promising technique for development
of complex software systems. The success of CBSE has been evidenced by a
variety of component models proposed both in industry and academia [3, 4].
CBSE suggests software modularity and reusability to facilitate the develop-
ment of high-quality software. For instance, different functional modules or
even subsystems of the control software of an airplane can be encapsulated
into reusable software components which can be reused multiple times for
the same system or for other systems in the same software product line.

Applying CBSE in multi-mode systems or the other way round has been
a largely unexplored research area, possibly because multi-mode systems
and CBSE are fundamentally conflicting in the sense that the former tradition-
ally is a top-down approach, whereas the latter is a bottom-up approach.
Our research goal in this report is to despite this apparent conflict combine
these approaches and benefit from the advantages of both multi-mode sys-
tems and CBSE. Hence, we propose component-based software development
of multi-mode systems, characterized by the independent development and
reuse of multi-mode components (i.e., components which can run in multiple modes) in the system modeling phase.

1.1 A guiding example

As a guiding example, consider a proof-of-concept healthcare monitoring system. The system consists of two subsystems: Data Acquisition Subsystem and Monitoring Subsystem. The Data Acquisition Subsystem uses cameras and microphones to collect video and audio data from a ward or a private home. The data is encoded, encrypted, and transmitted to the Monitoring Subsystem over a reliable network. The data arriving at the Monitoring Subsystem is decrypted, decoded, and reported to the health center. Video and audio data are decoded separately. The video is displayed on a screen, while the audio is played by a speaker. The Monitoring Subsystem also includes an alarm which is triggered when the person being monitored encounters a dangerous situation, such as falling or suffering from a heart attack.

Our focus is on the software architecture of the Monitoring Subsystem (MoS), which is composed by multi-mode components. The component structure of the system is modeled and illustrated in Fig. 1 where its component hierarchy is presented on the left and component connections are depicted on the right. The system is represented by Component MoS at the top level, composed by three subcomponents: DaD for data decryption, the multimedia decoder MuD, and EvA for event analysis. Due to the tree structure of the component hierarchy, DaD, MuD, and EvA are also called the children of MoS, which consequently is their parent. The system can run in two different modes: Regular monitoring mode (denoted as $Rm$) and Attention mode (denoted as $Att$).

The system runs in mode $Rm$ if nothing special happens to the healthcare target. This mode has very low requirement on video resolution and there is no need to transmit audio data. Hence it is assumed that only video data is transmitted and a fast video encoding/decoding algorithm is used to provide medium video quality which can be maintained by low CPU and bandwidth usage. Shown in Fig. 1, the video data from the network is first decrypted by DaD and then decoded by MuD, which sends the decoded video data to the display in the health center. Represented by the dimmed color, EvA is deactivated when MoS runs in $Rm$. Meanwhile, DaD runs in a Regular mode $R1$, while MuD runs in a Regular decoding mode $Rd$. The internal structure of MuD for mode $Rd$ is also depicted in Fig. 1. MuD consists of three subcomponents: VAE for video/audio extraction, a video decoder ViD and an audio decoder AuD. While MuD runs in $Rd$, ViD is its only activated subcomponent running in a Regular video decoding mode $Rvd$, since MuD receives only decrypted video data.

When an accident is detected by the Data Acquisition Subsystem, both
subsystems will switch to an *Attention* mode *Att*. This mode intends to raise the attention of the health center. Both video and audio data are expected to be transmitted. Moreover, the video resolution should be higher, thus requiring a different encoding/decoding algorithm. Component EvA is activated, running in a *Regular* mode *R2*, to analyze the detected event and may trigger an alarm when necessary. Due to the growing video data size and the additional audio data, the network load will be substantially increased. Depending on the current network condition, MuD may run in an *Enhanced decoding* mode *Ed* or a *Degraded QoS* mode *Dq*. Under satisfying network conditions, MuD runs in *Ed*, with all its subcomponents activated. The subcomponent VAE runs in a *Regular* mode *R3*, separating decrypted video and audio data and sending them to ViD and AuD, respectively. To produce higher quality video, the video decoder ViD runs in an *Enhanced video decoding* mode *Evd* by applying a different video decoding algorithm. This is represented by the grey color of ViD in Fig. 1. The audio decoder AuD runs in a *Regular audio decoding* mode *Rad*. However, if the network condition deteriorates to a certain level, it may become unsuitable to transmit both video and audio data. A possible strategy is to skip the transmission of audio data to maintain the video quality which could be more important. As a result, MuD switches to the mode *Dq*, in which AuD is deactivated to skip audio decoding. When the network condition returns to normal, MuD will automatically switch back to *Ed*. Note that VAE only receives video
data while MuD runs in $Dq$. This implies that VAE can be deactivated in the same way as AuD. However, VAE remains activated when MuD runs in $Dq$ to avoid frequent activation and deactivation. Without audio data, VAE simply forwards the video data to ViD.

We distinguish two types of components in the Monitoring Subsystem: *primitive components* and *composite components*. DaD, EvA, VAE, ViD, and AuD are primitive components which are directly implemented by code, while MoS and MuD are composite components composed by other components. What makes this system distinctive compared with traditional component-based systems is its constitution of multi-mode components.

### 1.2 Contributions

In achieving the component-based software development of multi-mode systems, this report includes the following key contributions:

- A mode mapping mechanism for mapping component modes during the composition of multi-mode components. This mechanism uses Mode Mapping Automata (MMA) to locally map the modes of a composite component to the modes of its subcomponents at design time. The mechanism partially builds on the MMA in [5], which are substantially refined, including detailed specification of MMA syntax and execution semantics, as well as formal definition of MMA composition.

- A mode transformation technique that transforms component modes to system-wide modes to achieve efficient implementation. This is obtained by flattening the hierarchical structure of component modes mapped at different levels. Mode transformation can be included in the mapping from the design to implementation level (“compilation”), after the mode mappings of all composite components in a system have been specified. An initial version of the mode transformation technique is presented in [6].

- A prototype tool, MCORE—the Multi-mode COmponent Reuse Environment, which implements both the mode mapping mechanism and the mode transformation technique for building multi-mode systems with multi-mode components. MCORE is implemented as a preprocessor for the commercial component-based development environment Rubus ICE [7], and thus extends Rubus ICE, which supports system level modes only, with the ability to benefit from reusable multi-mode components.

Mode mapping elegantly links modes and software component reuse. The hierarchical composition of multi-mode components easily allows one to build multi-mode systems with multi-mode behaviors at various levels.
A potential drawback of this approach is the runtime overhead due to inter-component communication for coordination of the mode switches of different components. To eliminate this runtime drawback, while still being able to design systems from reusable multi-mode components we introduce mode transformation, which enables efficient centralized mode management of a multi-mode system while preserving the flexible reuse of multi-mode components.

The rest of this report is structured as follows: Section 2 elaborates on the composition of multi-mode components and the mode mapping mechanism. Section 3 presents the mode transformation technique. The prototype tool MCORE is described in Section 4. Related work is reviewed in Section 5. Finally, Section 6 concludes the report and discusses some future work.

2 The composition of multi-mode components

As an essential step in the composition of multi-mode components, mode mapping unambiguously specifies the mode relation between different multi-mode components at design time. This section highlights the essential properties of multi-mode components and the motivation of mode mapping, followed by a thorough explanation of Mode Mapping Automata (MMA), i.e., a formal description of mode mapping, as well as MMA composition.

2.1 Multi-mode components and mode mapping

In general, a multi-mode component supports multiple modes and has a unique configuration defined for each mode. Fig. 2 illustrates the key properties of a reusable multi-mode component. The configuration for each mode relies on various factors. For instance, a multi-mode primitive component may have different mode-specific behaviors for different modes; while a multi-mode composite component may have different set of subcomponents activated depending on its mode.

Figure 2: The illustration of a multi-mode component
A multi-mode component can switch between certain modes at runtime, either on its own initiative or passively as requested by another component. A mode switch leads to the reconfiguration of the component by changing its configuration in the current mode to a new configuration in the target mode. A local mode-switch manager is used to handle the mode switch of a multi-mode component. By having such a mode-switch manager in each component, a multi-mode component is able to exchange mode information with its parent and subcomponents via dedicated mode-switch ports (the blue ports in Fig. 2) during a mode switch even without knowing the global mode information. We have previously developed distributed mode-switch algorithms [8, 9] running in the mode-switch manager for the cooperative mode switch of different components.

Since we assume that multi-mode components are independently developed, they typically support different number of modes and name them differently. It is necessary to specify the relation between the modes of different components at design time without ambiguity. Such a specification is called mode mapping. To ensure reusability, the mode mapping must never violate the following principles:

- A primitive component knows the mode information (supported modes, initial mode and current mode) of itself, but knows nothing about the other components in the system.

- A composite component knows the mode information of itself and its immediate subcomponents, but knows nothing about the other components in the system.

Hence, it is the responsibility of each composite component to map its own modes to the modes of its subcomponents, whereas mode mapping is not needed for a primitive component. Tables 1 and 2 present the basic mode mappings of composite components MoS and MuD of the example in Fig. 1. Modes in the same column are mapped to each other. For instance, when MoS runs in mode $\text{Att}$, among its subcomponents, DaD runs in $\text{R1}$, MuD can run in either $\text{Ed}$ or $\text{Dq}$, and EvA runs in $\text{R2}$. These tables intuitively present a set of mode mapping rules within MoS and MuD. However, there are some other mode mapping rules which are beyond the description of both tables. For example, when MoS switches from $\text{Rm}$ to $\text{Att}$, according to Table 1, MuD may switch to either $\text{Ed}$ or $\text{Dq}$. To eliminate such non-determinism, one of $\text{Ed}$ and $\text{Dq}$ must be specified as the default target mode of MuD. To be able to formally specify all types of mode mapping rules, we came up with a more powerful presentation—Mode Mapping Automata.

Before going into the details of the Mode Mapping Automata, we would like to point out that it is not our intention to develop a complete theory for a new type of automata. Rather, our focus is to, based on established
automata concepts, provide a formalization of mode mapping that suits our specific purposes.

2.2 Mode Mapping Automata

Let $c$ be a composite component with $\mathcal{SC}_c$ being the set of subcomponents of $c$ and $P_c$ being the parent of $c$. When $c$ is running in one of its supported modes, it should always know its current mode and the current modes of all $c_i \in \mathcal{SC}_c$ by its mode mapping, which can be presented by mode mapping tables such as Table 1. Moreover, whenever the mode-switch manager of $c$ notices the mode switch of $c_i \in \mathcal{SC}_c \cup \{c\}$, it will refer to the mode mapping which should tell which other components among $\mathcal{SC}_c \cup \{c\} \setminus \{c_i\}$ should also switch mode as a consequence, and the new modes of these components.

The complete mode mapping of $c$ can be formally presented by a set of MMA, which consist of one Mode Mapping Automaton of $c$ (denoted as $\text{MMA}_c^s$) and one MMA of each subcomponent $c_i \in \mathcal{SC}_c$ (denoted as $\text{MMA}_{c_i}^c$). Here we call $\text{MMA}_c^s$ a self MMA and $\text{MMA}_{c_i}^c$ a child MMA.

As an example, Fig. 3 presents the set of MMA of the component MuD in Fig. 1, including a self MMA ($\text{MMA}_{\text{MuD}}^s$) and three child MMA ($\text{MMA}_{\text{VAE}}^c$, $\text{MMA}_{\text{ViD}}^c$, and $\text{MMA}_{\text{AuD}}^c$). These MMA are hierarchically organized in the same way as the corresponding components. Each MMA can receive and emit internal or external signals. Internal signals are used to synchronize the pair of the self MMA and its child MMA while external signals interact with its local mode-switch manager for requesting and returning mode mapping results. As shown in Fig. 2, a multi-mode component can exchange moderelated information with its parent and subcomponents via its dedicated

<table>
<thead>
<tr>
<th>Component</th>
<th>MoS</th>
<th>Rm</th>
<th>Att</th>
</tr>
</thead>
<tbody>
<tr>
<td>DaD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MuD</td>
<td></td>
<td>Rd</td>
<td>Ed</td>
</tr>
<tr>
<td>EvA</td>
<td></td>
<td></td>
<td>Dq</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component</th>
<th>Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MuD</td>
<td>Rd</td>
</tr>
<tr>
<td>VAE</td>
<td>Deactivated</td>
</tr>
<tr>
<td>ViD</td>
<td>Rvd</td>
</tr>
<tr>
<td>AuD</td>
<td>Deactivated</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component</th>
<th>Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DaD</td>
<td></td>
</tr>
<tr>
<td>MuD</td>
<td></td>
</tr>
<tr>
<td>VAE</td>
<td>Deactivated</td>
</tr>
<tr>
<td>ViD</td>
<td></td>
</tr>
<tr>
<td>AuD</td>
<td></td>
</tr>
</tbody>
</table>
mode-switch ports. Every time MuD receives a mode-switch request from its parent MoS or a subcomponent via a dedicated mode-switch port, its mode-switch manager will send an external signal to the set of MMA to derive the modes of MuD and the modes of its subcomponents.

Figure 3: The role of the mode mapping of MuD at runtime

A self or child MMA can be formally defined as follows:

**Definition 1. Mode Mapping Automaton (MMA):** An MMA is defined as a tuple:

\[ \langle S, s^0, SI, T \rangle \]

where \( S \) is a set of states; \( s^0 \in S \) is the initial state; \( SI = I \cup E \) (\( I \cap E = \emptyset \)) is a set of signals received or emitted during a state transition, with \( I \) as the set of internal signals and \( E \) as the set of external signals; \( T \subseteq S \times SI \times 2^SI \times S \) is a set of transitions of the MMA.

In addition to the formal definition above, an MMA can be graphically represented as a state machine with states and transitions. Each state is represented by a circle, with the initial state being marked by a double circle. If the MMA is a self MMA of \( c \), then each state corresponds to a mode of \( c \). If the MMA is a child MMA of \( c \) associated with \( c_i \in SC_c \), then each state corresponds to a mode of \( c_i \) or the deactivated status of \( c_i \), if \( c_i \) can be deactivated, denoted as \( D \). A transition \( t \in T \) is represented by an arrow from a state \( s \) to a state \( s' \), denoted as \( s \xrightarrow{In/Out} s' \), where In/Out is
the label of the transition, “In” is the input that triggers the transition and “Out” is the output of the transition.

Fig. 4 depicts $\text{MMA}_{\text{MuD}}^s$, i.e, the self MMA of MuD of the example in Fig. 1. Three states are included in this MMA, implying that MuD can run in three modes. The state transitions of $\text{MMA}_{\text{MuD}}^s$ and the corresponding child MMA $\text{MMA}_{\text{VAE}}^c$, $\text{MMA}_{\text{ViD}}^c$, and $\text{MMA}_{\text{AuD}}^c$ (later shown in Fig. 5) are manually specified to determine the mode mapping of MuD. Shown in Fig. 4, each transition has a label with input and output separated by “/”. The input and output of a transition are typically internal and external signals. We define two types of internal signals and a single type of external signal denoted as follows:

- $x.I(y)$, an internal signal emitted by a self MMA to the recipient $\text{MMA}_x^c$, which is asked to change state to $y$.
- $x.I(z \rightarrow y)$, an internal signal emitted by $\text{MMA}_x^c$ which requests to change state from $z$ to $y$.
- $x.E(y)$, an external signal asking $\text{MMA}_x$, which is either a self MMA or a child MMA, to change state to $y$.

Figure 4: The self Mode Mapping Automaton of MuD

Note that an internal signal emitted from a self MMA to a child MMA is in the form of $x.I(y)$, while an internal signal emitted from a child MMA to a self MMA is in the form $x.I(z \rightarrow y)$. A self MMA of a composite component $a$ sends an internal signal $b.I(m_b)$ to a child MMA $\text{MMA}_b^c$ when the mode mapping of $a$ wants $b$ to switch to a new mode $m_b$, regardless of the current mode of $b$. In contrast, the reason for using $x.I(z \rightarrow y)$ instead of $x.I(y)$ in the latter case is that both the current mode and the target mode of a component can affect the mode mapping result of its parent.

In order to get a self MMA synchronized properly with the corresponding child MMA, the set of MMA of a composite component $c$ must fulfill several
constraints regarding their transitions. We define the constraints by taking the following principles into account:

- The mode mapping of \(c\) is determined by its self MMA, not any child MMA.

- The input of the set of MMA must be an external signal which is either the input of \(MMA_c^s\) (if the mode switch is initiated by \(c\)) or the input of a child MMA \(MMA_{c_i}^c\), where \(c_i\) is a subcomponent of \(c\) (if the mode switch is initiated by \(c_i\)).

- The output of the set of MMA must be either a set of external signals (if the input external signal implies the mode switch of at least another component in \(SC_c \cup \{c\}\)) or an empty set otherwise.

- The internal synchronization between \(MMA_c^s\) and the child MMA is based on only internal signals. There is no direct synchronization between child MMA.

- An internal signal from \(MMA_c^s\) to a child MMA \(MMA_{c_i}^c\) only needs to indicate the new mode of \(c_i\), while an internal signal from \(MMA_{c_i}^c\) to \(MMA_c^s\) needs to indicate both the current mode and new mode of \(c_i\), as both parameters affect the mode mapping results of \(MMA_c^s\).

- Usually the starting state and ending state of a transition in an MMA should be different. However, there is one exception in \(MMA_c^s\). If the mode switch of a subcomponent \(c_i\) does not imply the mode switch of \(c\), this will result in a transition in \(MMA_c^s\) starting and ending in the same state.

The principles above yield the following constraints applied to both self and child MMA:

1. For the self MMA \(MMA_c^s\), the input of each transition must be either an external signal \(c.E(m_c)\) or an internal signal \(c_i.I(m_{c_i}^1 \rightarrow m_{c_i}^2)\), where \(m_c\) is the target mode of \(c\) of the transition, \(c_i \in SC_c\), and \(m_{c_i}^1\) and \(m_{c_i}^2\) are the modes of \(c_i\). Then,

   (a) If the input is \(c.E(m_c)\), then the output is either an empty set or a set of internal signals in the format \(c_i.I(m_{c_i})\), where \(c_i \in SC_c\) and \(m_{c_i}\) is the target mode of \(c_i\).

   (b) If the input is \(c_i.I(m_{c_i}^1 \rightarrow m_{c_i}^2)\), then the output is a set with two parts. The first part is an external signal \(c.E(m_c)\) if the starting state of this transition is different from the ending state \(m_c\), or an empty set otherwise. The second part is either an empty set or a set of internal signals in the format \(c_i.I(m_{c_i})\), where \(c_i \in SC_c\) and \(m_{c_i}\) is the target mode of \(c_i\).
2. For each child MMA $\text{MMA}_{c_i}^c$, where $c_i$ is a subcomponent of $c$, if it has only one state, then there is no transition in $\text{MMA}_{c_i}^c$. If $\text{MMA}_{c_i}^c$ has at least two states, then the starting state and ending state of each transition must be different. For each transition to the state $m_{c_i}$, the input must be either an external signal $c_i.E(m_{c_i})$ or an internal signal $c_i.I(m_{c_i})$. If the input is $c_i.E(m_{c_i})$, then the output must be an internal signal $c_i.I(m_{c_i}' \rightarrow m_{c_i})$ where $m_{c_i}'$ is the current mode of $c_i$; if the input is $c_i.I(m_{c_i})$, then the output must be an external signal $c_i.E(m_{c_i})$.

3. For each internal signal $c_i.I(m_{c_i})$ in the output of a transition of $\text{MMA}_{c_i}^c$, there must be one and only one transition of $\text{MMA}_{c_i}^c$ with the input $c_i.I(m_{c_i})$, synchronized with this transition of $\text{MMA}_{c_i}^c$ and vice versa. For each internal signal $c_i.I(m_{c_i}' \rightarrow m_{c_i})$ in the output of a transition of $\text{MMA}_{c_i}^c$, there must be one and only one transition of $\text{MMA}_{c_i}^c$ with the input $c_i.I(m_{c_i}' \rightarrow m_{c_i})$, synchronized with this transition of $\text{MMA}_{c_i}^c$ and vice versa.

Following the constraints above, we specify the MMA synchronization semantics as follows: Let $c$ be a composite component with $\mathcal{SC}_c = \{c_1, c_2, \ldots, c_n\}$ ($n \in \mathbb{N}$). Let $\mathcal{A} = \{\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_n\}$ be the set of MMA of $c$, with $\mathcal{A}_0$ associated with $c$ and $\mathcal{A}_k$ associated with $c_k$ ($k \in [1, n]$). Hence $\mathcal{A}_0$ is the self MMA while the others are child MMA. Any external signal from the mode-switch manager of $c$ at runtime can potentially lead to the MMA synchronization of $c$. An MMA synchronization is an atomic transaction which is always performed between $\mathcal{A}_0$ and the other child MMA. The synchronization depends on if the external signal arrives at $\mathcal{A}_0$ or at a child MMA:

1. An external signal arrives at $\mathcal{A}_0$ asking $c$ to switch to mode $m_c$. This triggers a transition $t$ of $\mathcal{A}_0$ with the input $c.E(m_c)$, giving rise to two possible subsequent events:

   - The mode switch of $c$ under this condition does not imply the mode switch of any subcomponent of $c$. Then the output of the transition $t$ is an empty set $\emptyset$, i.e. there is no internal synchronization between the set of MMA of $c$ and no external signals are expected to be emitted from them to the mode-switch manager of $c$.

   - The mode switch of $c$ under this condition implies the mode switch of at least one of its subcomponents. Let $\mathcal{G} \subseteq \mathcal{SC}_c$ be the set of its subcomponents which also need to switch mode. Then the output of $t$ is a set $\mathcal{R}$ such that for each $c_k^i \in \mathcal{G}$ which is expected to switch to the mode $m_{c_k^i}$, there must exist an internal
signal $c^k_i.I(m_{c^k_i}) \in \mathcal{R}$, synchronizing $t$ with the transition $t_i$ with the label $c^k_i.I(m_{c^k_i})/c^k_i.E(m_{c^k_i})$ of the child MMA $A_i$. The output $c^k_i.E(m_{c^k_i})$ of $t_i$ is sent from $A_i$ to the mode-switch manager of $c$ which will inform $c^k_i$ to switch to mode $m_{c^k_i}$.

2. An external signal arrives at $A_k$ ($k \in [1,n]$) asking $c^k_i$ to switch from mode $m_1$ to $m_2$. This triggers a transition $t_k$ of $A_k$ with the label $c^k_i.E(m_2)/c^k_i.I(m_1 \rightarrow m_2)$, synchronized with the transition $t$ of $A_0$ with the input $c^k_i.I(m_1 \rightarrow m_2)$. The output of $t$ may give rise to four possible subsequent events:

- The mode switch of $c^k_i$ under this condition does not imply the mode switch of any component among $c$ and $SC_c \setminus \{c^k_i\}$. Then the output of $t$ is $\emptyset$ and no further synchronization and external signal are expected.
- The mode switch of $c^k_i$ under this condition implies the mode switch of $c$ but not the mode switch of any other subcomponent of $c$. Let $m_c$ be the new mode of $c$. Then the output of $t$ is $c.E(m_c)$.
- The mode switch of $c^k_i$ under this condition implies the mode switch of at least another subcomponent of $c$ but not the mode switch of $c$. Then the subsequent event will be the same as the second case under Condition (1). Hence the output of $t$ is the set $\mathcal{R}$.
- The mode switch of $c^k_i$ under this condition implies the mode switches of both $c$ and at least another subcomponent of $c$. Let $m_c$ be the new mode of $c$. Then the output of $t$ will be $\{c.E(m_c)\} \cup \mathcal{R}$, where $c.E(m_c)$ is an external signal emitted from $A_0$.

Our MMA synchronization semantics is partially illustrated by Fig. 4 and Fig. 5 which provide the graphical presentations for MMA, and the corresponding child MMA.

We assume that MoS can initiate a mode switch between $Rm$ and $Att$, while MuD can initiate another mode switch between $Ed$ and $Dq$. Let’s use one scenario to demonstrate the synchronization between MMA, and the child MMA. Suppose that MoS initiates a mode switch from $Rm$ to $Att$ due to the detection of a dangerous situation reported to the health center, leading to the mode switch of MuD from $Rd$ to $Ed$ as a consequence. Then the mode-switch manager of MuD is supposed to send an external signal $Mu.D.E(Ed)$ to MMA, triggering the transition $Rd \xrightarrow{Mu.D.E(Ed)/\{VAE.I(R3),Vsl.I(Evd),AuD.I(Rad)\}} Ed$ of MMA, which is synchronized with the transition $D \xrightarrow{VAE.I(R3)\{VAE.E(R3)\}} R3$ of
$MMA_{VAE}^c \xrightarrow{Rvd} Evd$ of $MMA_{VI}^c$, and the transition $D \xrightarrow{AuD.I(Rad)/\{AuD.E(Rad)\}} Rad$ of $MMA_{AuD}^c$. The external signals $VAE.E(R3)$, $ViD.E(Evd)$, and $AuD.E(Rad)$ will be sent to the mode-switch manager of MuD, which will inform VAE, ViD, AuD to switch mode to $R3$, $Evd$, and $Rad$, respectively.

![Diagram of Mode Mapping Automata of MuD](image)

**Figure 5:** The child Mode Mapping Automata of MuD

Similarly, Fig. 6 and Fig. 7 display the set of MMA of MoS which replace Table 1, including $MMA_{MoS}^s$, $MMA_{DaD}^c$, $MMA_{MuD}^c$, and $MMA_{EvA}^c$. To simplify the graphical presentation of a self MMA, if multiple transitions share the same output, starting and ending states, we combine them into one transition where their different inputs are separated by the notation “||”. For instance, $MMA_{MoS}^s$ in Fig. 6 has two transitions $Att \xrightarrow{MuD.I(Ed\rightarrow Dq)/\emptyset} Att$ and $Att \xrightarrow{MuD.I(Dq\rightarrow Ed)/\emptyset} Att$, which are combined into one transition $Att \xrightarrow{MuD.I(Ed\rightarrow Dq) || MuD.I(Dq\rightarrow Ed)/\emptyset} Att$.

An interesting phenomenon that deserves particular attention is the distinction between $MMA_{MoD}^s$ in Fig. 4 and $MMA_{MoD}^c$ in Fig. 7. The former is a self MMA included in the mode mapping of MuD, while the latter is a child MMA included in the mode mapping of MoS.

![Diagram of Mode Mapping Automaton of MoS](image)

**Figure 6:** The self Mode Mapping Automaton of MoS
The internal synchronization between a set of MMA of a composite component is actually invisible to the mode-switch manager of this composite component. What the mode-switch manager sees is the composition of these MMA. For each composite component, the basic idea of deriving its MMA composition is to analyze the possible outputs of a set of MMA for each possible input from the mode-switch manager. According to the MMA synchronization semantics, an external signal from the mode-switch manager of a composite component may give rise to six possible subsequent events with respect to the internal MMA synchronization and the output of the set of MMA. All these six cases must be taken into account in defining the MMA composition.

Also, note that the MMA composition will be defined to suit our purposes only, i.e., it is not a general composition that can be used to compose any set of MMA. Instead, it is only applicable for composition of a single self MMA with a non-empty finite set of child MMA, where these MMA are required to satisfy all the constraints we impose on these types of MMA. As a consequence the composition cannot be further composed using our composition.

In the below definition of MMA composition, we assume that the el-
elements of sets are indexed such that we by the indexing can identify the MMA that a specific element is related to, and \( x[i] \) will be used to denote the element of \( x \) indexed with \( i \).

**Definition 2. MMA composition:** For a set \( A = \{ A_0, A_1, \ldots, A_n \} (n \in \mathbb{N}) \) of MMA, where \( A_0 = (S_0, s_0^0, SI_0, T_0) \) corresponds to a self MMA and \( \forall k \in [1, n], A_k = (S_k, s_k^0, SI_k, T_k) \) correspond to the child MMA synchronized with \( A_0 \), the MMA composition of \( A \) is an MMA defined by the tuple

\[
\langle S, s^0, SI, T \rangle
\]

where

\[
S \subseteq S_0 \times S_1 \times \cdots \times S_n
\]

\[
s^0 = (s_0^0, s_1^0, \cdots, s_n^0)
\]

\[
SI = \bigcup_{i \in [0, n]} E_i
\]

\( S \) and \( T \) are the least sets satisfying:

1. \( s^0 \in S \)
2. If

\[
\exists s = (s_0, s_1, \cdots, s_n) \in S \land s_0 \xrightarrow{e_0/O_0} s'_0 \in T_0 \quad [\text{An external signal } e_0 \in E_0 \text{ arrives at } A_0.]
\]

then

\[
s \xrightarrow{e_0/O} s' \in T \land s' = (s'_0, s'_1, \cdots, s'_n) \in S
\]

where \( O \) and \( s'_k (k \in [1, n]) \) are defined by the following:

- \( O = \bigcup_{k \in [1, n]} O_k \)

- \( \forall k \in [1, n], \text{ if } \exists s_k \xrightarrow{O_k/O'_k} s''_k \in T_k, \text{ then } O_k = O'_k \land s'_k = s''_k; \text{ else } O_k = \emptyset \land s'_k = s_k. \)

3. If

\[
\exists s = (s_0, s_1, \cdots, s_n) \in S \land s_i \xrightarrow{e_i/O_i} s'_i \in T_i \land s_0 \xrightarrow{O_i/O_0} s'_0 \in T_0 \quad [\text{An external signal } e_i \in E_i(i \in [1, n]) \text{ arrives at } A_i.]
\]

then

\[
s \xrightarrow{e_i/O} s' \in T \land s' = (s'_0, s'_1, \cdots, s'_n) \in S
\]

where \( O \) and \( s'_k (k \in [1, n], k \neq i) \) are defined by the following:
• If $s_0 = s'_0$, then

$$\mathcal{O} = \bigcup_{k \in [1,n], \ k \neq i} \mathcal{O}_k$$

• Else ($s_0 \neq s'_0$),

$$\mathcal{O} = \{\mathcal{O}_0[0]\} \cup \bigcup_{k \in [1,n], \ k \neq i} \mathcal{O}_k$$

• $\forall k \in [1,n]$, if $\exists s_k \xrightarrow{\mathcal{O}_0[k]/\mathcal{O}'_k} s''_k \in \mathcal{T}_k$, then $\mathcal{O}_k = \mathcal{O}'_k \land s'_k = s''_k$; else $\mathcal{O}_k = \emptyset \land s'_k = s_k$.

By Definition 2, a set of MMA is closed under composition. This means that the MMA after composition itself is also an MMA. However, note that the MMA resulting from a composition cannot be further composed by our composition, as it does not qualify as a self or child MMA. Also, the intention is obviously that the composed MMA behaves the same as the set of MMA being composed from the perspective of the corresponding mode-switch manager, i.e., for each input external signal from the mode-switch manager, the same set of external signals are output by the composed MMA as by the set of MMA being composed. By Definition 2, for each input external signal arriving at either the self MMA or a child MMA, MMA composition merges all the intermediate steps for the internal synchronization and emits all the possible external signals as the output in a single transition. This means that the mode-switch manager receives the same set of external signals before and after MMA composition.

Based on Definition 2, let’s revisit the six cases mentioned by the MMA synchronization semantics:

1. An external signal arrives at $\mathcal{A}_0$ and $\mathcal{O}_0 = \emptyset$, when the mode switch of a composite component implies no mode switch among its subcomponents.

2. An external signal arrives at $\mathcal{A}_0$ and $\mathcal{O}_0 \neq \emptyset$, when the mode switch of a composite component implies the mode switch of at least one of its subcomponents.

3. An external signal arrives at $\mathcal{A}_i$ ($i \in [1,n]$) and $\mathcal{O}_0 = \emptyset$, when the mode switch of a component $c$ implies no mode switch among its parent and its siblings, i.e. components with the same parent as $c$. 

18
4. An external signal arrives at $A_i$ ($i \in [1, n]$) and $O_0$ only contains an external signal $e_0$, when the mode switch of a component only implies the mode switch of its parent but not its siblings.

5. An external signal arrives at $A_i$ ($i \in [1, n]$) and $O_0$ only contains a set of internal signals $i_k$ ($k \in [1, n]$), when the mode switch of a component only implies the mode switch of at least one of its siblings but not its parent.

6. An external signal arrives at $A_i$ ($i \in [1, n]$) and $O_0$ contains both $e_0$ and $i_k$ ($k \in [1, n]$), when the mode switch of a component implies the mode switch of both of its parent and at least one sibling.

As an example, Fig. 8 demonstrates Case 1 by showing the MMA composition between a composite component $a$ and its two subcomponents $b$ and $c$. Fig. 8(a) shows the mode mapping table of $a$, while Fig. 8(b) shows the set of MMA within $a$. The MMA after composition is shown in Fig. 8(c). Suppose an external signal $a.E(m_2^a)$ arrives at $MMA_a^a$ (the self MMA), triggering the transition $m_1^a \xrightarrow{a.E(m_2^a)/\phi} m_2^a$. Since this external signal does not imply the mode switch of $b$ or $c$, no state transition will occur in $MMA_b^c$ or $MMA_c^c$. Let $\mathcal{A}$ be the MMA after composition, with two states $s_1 = (m_1^a, m_1^b, m_1^c)$ and $s_2 = (m_2^a, m_1^b, m_1^c)$. According to Definition 2, $\mathcal{A}$ will undergo the transition $s_1 \xrightarrow{a.E(m_2^a)/\phi} s_2$.

Figure 8: MMA composition—Case 1

As a more concrete example, consider the MMA composition of $MMA_{MuD}^c$ in Fig. 4, $MMA_{VAE}^c$ and $MMA_{VI}^c$, and $MMA_{AuD}^c$ in Fig. 5. Case 2 is involved in the MMA composition. The composition result is illus-
trated in Fig. 9, where $s_1 = (Rd, D, Rvd, D)$, $s_2 = (Ed, R3, Evd, Rad)$, and $s_3 = (Dq, R3, Evd, D)$.

![Diagram of MMA composition for MuD]

**Figure 9: MMA composition for MuD**

### 2.4 Mode mapping verification

The mode mapping of a composite component specified by MMA can be easily verified by model checking. We use the model checker UPPAAL [10] for mode mapping verification. Since UPPAAL is a convenient tool for modeling and verifying concurrent state transition systems, it is fairly straightforward to graphically model a set of MMA in UPPAAL.

Using UPPAAL, we have modeled the mode mapping of MuD specified by the set of MMA in Fig. 4 and Fig. 5. The behavior of the local mode-switch manager of MuD, the self MMA of MuD, and the child MMA of its subcomponents are modeled as separate automata in UPPAAL. For instance, Fig. 10 showcases three typical UPPAAL models for the mode-switch manager of MuD, $MMA_{cVAE}$, and $MMA_{sMuD}$. Each mode of a component is represented by a state in these UPPAAL models (e.g., $mode_{R3}$ represents mode $R3$ in Fig. 10(b)). These models also contain committed states marked with “C” in a circle, which are intermediate states during a mode switch. External and internal signals are simulated as channels synchronized between multiple UPPAAL models. For example, $VAE.I[R3]!$ denotes the internal signal $VAE.I(R3)$ emitted by $MMA_{sMuD}$, while $VAE.I[R3]?$ denotes the same signal $VAE.I(R3)$ received by $MMA_{VAE}$. The UPPAAL model of $MMA_{sMuD}$ in Fig. 10(c) is consistent with $MMA_{sMuD}$ in Fig. 4. The reason why the UPPAAL model contains one or more intermediate states for each mode switch is that receiving and sending each signal needs to be modeled sequentially in UPPAAL. This essentially does not change the execution semantics, as all intermediate states are committed states, whose incoming and outgoing transitions are performed as a single atomic transaction. In

---

1The complete UPPAAL model is available at [http://www.cse.chalmers.se/~yhang/JUCS/](http://www.cse.chalmers.se/~yhang/JUCS/)
addition, shown in Fig. 10(a), the mode-switch manager of MuD consists of two states. *InitialS* is the initial state, where the mode-switch manager can send an external signal to MMA\textsubscript{MuD} and switch to the state *ModeSwitching*. Meanwhile, a boolean variable *switching* is set to true, indicating an ongoing mode switch. Depending on the current mode of MuD and the new mode indicated by the external signal from the mode-switch manager, there are four possible events, leading to different transitions among these components: (1) \(k_1\): MuD requests to switch from \(Rd\) to \(Ed\); (2) \(k_2\): MuD requests to switch from \(Ed\) to \(Rd\); (3) \(k_3\): MuD requests to switch from \(Ed\) to \(Dq\); (4) \(k_4\): MuD requests to switch from \(Dq\) to \(Ed\). Each event ID is assigned to a variable *eventID* as shown in Fig. 10(c).

![UPPAAL model for the mode-switch manager of MuD](image1)

(a) UPPAAL model for the mode-switch manager of MuD

![UPPAAL model for the child MMA of VAE](image2)

(b) UPPAAL model for the child MMA of VAE

![UPPAAL model for the self MMA of MuD](image3)

(c) UPPAAL model for the self MMA of MuD

**Figure 10:** UPPAAL models of the mode mapping of MuD

Based on the UPPAAL models, we can verify that the set of MMA of MuD satisfy the expected constraints by checking properties formulated in the UPPAAL query language which is a subset of Timed Computation Tree Logic [11]. The following are four types of properties addressing different constraints:

**P1.** \(A\not\downarrow\not\text{deadlock}\): The complete set of UPPAAL models are deadlock-free. This is not directly related to mode mapping, but it is a fundamental property that we expect the model to satisfy.
P2. \( E<> sMMA\_MuD\_mode\_Ed \): It is possible for \( \muD \) to run in mode 
\( Ed \). This property should be verified for all the modes of \( \muD \) and its 
subcomponents.

P3. \( A[] (sMMA\_MuD\_mode\_Rd \text{ and } !\text{ModeSwitchManager.switching}) \) 
imply \( (cMMA\_VAE\_mode\_D \text{ and } cMMA\_ViD\_mode\_Rvd \text{ and } 
\text{cMMA}\_AuD\_mode\_D) \): When \( \muD \) runs in \( Rd \), its subcomponents 
\( \text{VAE} \) and \( \text{AuD} \) must be deactivated while the other subcomponent 
\( \text{ViD} \) must run in \( Rvd \). This property should be verified for all possible 
mode combinations between \( \muD \) and its subcomponents according 
to the mode mapping table in Table 2.

P4. \( (\text{ModeSwitchManager.switching and eventID==k1}) \) –
\( > (sMMA\_MuD\_mode\_Ed \text{ and } cMMA\_VAE\_mode\_R3 \text{ and } 
\text{cMMA}\_ViD\_mode\_Evd \text{ and } \text{cMMA}\_AuD\_mode\_Rad) \): An external 
signal requesting \( \muD \) to switch from \( Rd \) to \( Ed \) will make \( \text{VAE} \), 
\( \text{ViD} \), \( \text{AuD} \) switch to \( R3 \), \( Evd \), and \( Rad \), respectively. This property should 
be verified for all possible events from \( k1 \) to \( k4 \).

All these properties are satisfied with verification time less than 4ms.
Also, our UPPAAL models can be used as a common template for mod-
eling any other mode mapping specified by MMA. Due to the graphical 
resemblance between an MMA and the corresponding UPPAAL model, it is 
possible to generate UPPAAL models from MMA described by a graphical 
or textual domain-specific language.

3 Mode transformation

Our previous research results [9] show that the mode switch of a multi-mode 
component may lead to mode switches of other multi-mode components in 
the same system and it is not trivial to coordinate the mode switches of 
different components at runtime. The local mode-switch manager of each 
component needs to run delicate algorithms to communicate with the parent 
and subcomponents of the component via dedicated mode-switch ports to 
switch mode cooperatively. Such inter-component communication incurs 
runtime computation overhead and mode-switch latency. For instance, when 
\( \text{MoS} \) in the healthcare monitoring system triggers a mode switch from \( Rm \) 
to \( \text{Att} \), the mode-switch event is first propagated from \( \text{MoS} \) to \( \muD \) and 
\( \text{EvA} \), and \( \muD \) subsequently propagates the mode-switch event to \( \text{VAE} \), 
\( \text{ViD} \), and \( \text{AuD} \). Further, more handshake messages are exchanged between 
these components to keep mode consistency. The communication overhead 
grows as the component hierarchy becomes more complex.

The purpose of mode transformation is to eliminate the need for the 
mode-switch coordination among different multi-mode components by a
centralized mode management, and thereby achieve better runtime performance, provided that (1) all components are deployed on the same hardware platform, and (2) the mode information of each component is globally accessible. Illustrated in Fig. 11, mode transformation transfers the responsibility of mode-switch handling from the local mode-switch manager of each component to a single global mode-switch manager of the system in the system modeling phase. As a result of mode transformation, each multi-mode component becomes unaware of modes. Instead, a global mode transition graph is generated for the global mode-switch manager to handle mode switch at system level.

![Figure 11: The overview of mode transformation](image)

Our mode transformation process includes two sequential steps. First, given the mode mappings of all composite components, we construct an intermediate representation, a Mode Combination Tree (MCT) where all the possible system modes are identified. In the second step, based on a list of possible mode-switch events defined in the system, we add transitions between the identified system modes to construct the mode transition graph. The two steps are further explained in the following subsections separately.

### 3.1 Construction of the mode combination tree

The aim of constructing the MCT is to identify all the system modes. Let $M_c$ denote the set of supported modes of a component $c$ and $D$ denote the current mode of a deactivated component. Then we define system modes as follows:

**Definition 3. System modes based on component modes:** For a system composed by a set of components $C = \{c_1, c_2, \cdots, c_n\}$ $(n \in \mathbb{N})$, the set of system modes is defined as $M_s \subseteq \times_{i \in [1, n]} \{M_{c_i} \cup \{D\}\}$. Each system mode $m \in M_s$ is a mode combination of all components.

By Definition 3, each system mode $m = (m_{c_1}, m_{c_2}, \cdots, m_{c_n})$, where $m_{c_i} \in M_{c_i} \cup \{D\}$ for $i \in [1, n]$. In order to more explicitly indicate the relationship between $c_i$ and $m_{c_i}$, we shall hereafter use an alternative ex-
pression to represent a system mode: 

\[ m = \{(c_i, m_{c_i}) | i \in [1, n]\} \],

where \( m_{c_i} \in \mathcal{M}_{c_i} \cup \{D\} \). Using the same formalism, an MCT is defined as follows:

**Definition 4. Mode Combination Tree:** An MCT is a tree with a set of nodes \( \mathcal{N} = \{\mathcal{N}_0, \mathcal{N}_1, \cdots, \mathcal{N}_n\} \) (\( n \in \mathbb{N} \)), where \( \mathcal{N}_0 = \emptyset \) is the root node, and each other node \( \mathcal{N}_1 = \{(c_j, m_{c_j}) | j \in [1, k], k \in \mathbb{N}\} \) (\( i \in [1, n]\)), where for all \( j \), \( m_{c_j} \in \mathcal{M}_{c_j} \cup \{D\} \) and all \( c_j \) have the same depth level in the system component hierarchy.

Definition 4 implies that each node of an MCT, except the root node, provides a mode combination of components with the same depth level. A typical outlook of MCT is displayed in Fig. 12, where the construction of the MCT will be further explained.

Before the formal description of the MCT construction process, we need to introduce a number of additional notations and concepts. First, we introduce the valid Local Mode Combination (LMC) of a composite component \( c \), which is a feasible combination of a mode of \( c \) and the modes of all its subcomponents as per the local mode mapping of \( c \). To formally define the valid LMC of a composite component, let \( \mathcal{P}c \) and \( \mathcal{CC} \) be the set of primitive components and composite components in a system, respectively. Let \( \text{Top} \) be the component at the top of the component hierarchy in a system. For each \( c \in \mathcal{CC} \), a valid LMC of \( c \) is formally defined as follows:

**Definition 5. Valid Local Mode Combination:** For \( c \in \mathcal{CC} \) with \( \mathcal{SC}_c = \{c^1_1, \cdots, c^n_1\} \) (\( n \in \mathbb{N} \)), we call the set \( \mathcal{V}_c = \{(c, m_c), (c^1_1, m_{c^1_1}), \cdots, (c^n_1, m_{c^n_1})\} \) a valid LMC of \( c \), where \( m_c \in \mathcal{M}_c \cup \{D\} \) and \( \forall k \in [1, n], m^k_{c_k} \in \mathcal{M}_{c_k} \cup \{D\} \), if \( m_c \) and all \( m^k_{c_k} \) (\( k \in [1, n]\)) can be simultaneously executed by the corresponding components, i.e. conforming to the mode mapping of \( c \).

Note that each element in \( \mathcal{V}_c \) is a pair \((x, y)\) where \( x \in \mathcal{SC}_c \cup \{c\} \) and \( y \in \mathcal{M}_x \cup \{D\} \). For instance, the mode mapping of MoS in Table 1 implies three valid LMCs of MoS: (1) \{MoS, Rm\}, (DaD, R1), (MuD, Rd), (EvA, D))

(2) \{MoS, Att\}, (DaD, R1), (MuD, Ed), (EvA, R2)

(3) \{MoS, Att\}, (DaD, R1), (MuD, Dq), (EvA, R2) \}. Actually, each state after MMA composition represents a valid LMC.

Based on Definition 5, we further introduce the valid LMC concerning a specific mode of a composite component \( c \), which is a feasible combination of the modes of all subcomponents of \( c \) as per the local mode mapping of \( c \) when \( c \) is running in a particular mode. A formal definition is given as follows:

**Definition 6. Valid LMC concerning a specific mode:** For \( c \in \mathcal{CC} \) with \( \mathcal{SC}_c = \{c^1_1, \cdots, c^n_1\} \) (\( n \in \mathbb{N} \)), if when \( c \) is running in \( m_{c_e} \), and \( \forall k \in \mathcal{SC}_c \), \( k \in [1, n]\), \( \exists m^k_{c_k} \) such that \( \{(c, m_c), (c^1_1, m_{c^1_1}), \cdots, (c^n_1, m_{c^n_1})\} \) is a valid LMC of \( c \), then the set \( \mathcal{V}_{c,m_c} = \{(c^1_1, m_{c^1_1}), \cdots, (c^n_1, m_{c^n_1})\} \) is a valid LMC of \( c \) for \( m_c \).
Depending on the mode mapping of $c$, multiple valid LMCs of $c$ may exist for $m_c$. Let $W_{c,m_c}$ be the set of all valid LMCs of $c \in CC$ for $m_c$. Each element in $W_{c,m_c}$ is a set $V_{c,m_c}$. The total number of all valid LMCs of $c$ for $m_c$ is $|W_{c,m_c}|$. For instance, according to Table 1, $W_{MoS,Att} = \{V_{MoS,Att}^1, V_{MoS,Att}^2\}$, where $V_{MoS,Att}^1 = \{(DaD,RI),(MuD,Ed),(EvA,R2)\}$ and $V_{MoS,Att}^2 = \{(DaD,RI),(MuD,Dq),(EvA,R2)\}$. It is easy to automatically generate $W_{c,m_c}$ from the mode mapping of $c$. On the one hand, the valid LMCs of $c$ for $m_c$ can be easily identified from the $m_c$ column of the mode mapping table of $c$. On the other hand, after MMA composition, each state of the composed MMA represents a valid LMC of $c$. Then the valid LMCs of $c$ for $m_c$ can be easily found in the states with $m_c$. Next we introduce an important operator for combining different valid LMCs:

**Definition 7. Valid LMC operation:** Consider two sets of valid LMCs $W_1 = \{V_1, V_2, \ldots, V_m\}$ and $W_2 = \{V_{k+1}, V_{k+2}, \ldots, V_{k+n}\}$, where $m, n, k \in \mathbb{N}$ and $k \geq m$. Let $\oplus$ be an operator such that $W_1 \oplus W_2 = \{V_i \cup V_{k+j} | i \in [1,m], j \in [1,n]\}$. In addition, for each $l \in \mathbb{N}$, $W_1 \oplus W_2 \oplus \cdots \oplus W_l$ can be represented as $\bigoplus_{o \in [1,l]} W_o$.

For the sake of lucidity, let us clarify the $\oplus$ operator using a small example. Suppose $W_1 = \{V_1, V_2\}$ where $V_1 = \{(a,m^1_a), (b,m^1_b)\}$ and $V_2 = \{(a,m^2_a), (b,m^2_b)\}$; and $W_2 = \{V_3, V_4\}$ where $V_3 = \{(c,m^1_c), (d,m^1_d)\}$ and $V_4 = \{(c,m^2_c), (d,m^2_d)\}$. Then,

$$W_1 \oplus W_2 = \{V_1 \cup V_3, V_1 \cup V_4, V_2 \cup V_3, V_2 \cup V_4\}$$

$$= \{(a,m^1_a), (b,m^1_b), (c,m^1_c), (d,m^1_d)\},$$

$$\{(a,m^1_a), (b,m^1_b), (c,m^2_c), (d,m^2_d)\},$$

$$\{(a,m^2_a), (b,m^2_b), (c,m^1_c), (d,m^1_d)\},$$

$$\{(a,m^2_a), (b,m^2_b), (c,m^2_c), (d,m^2_d)\}\}

Given the component hierarchy of a system and the mode mappings of all composite components in the system, the MCT of the system can be constructed by creating nodes top-down from the root node. For each node $N$ of an MCT, let $d_N$ be its depth level, and $\lambda_N$ be the number of new nodes created from this node. We use $N_i \succ N_j$ to denote that a new node $N_i$ is created from on old node $N_j$. Moreover, let $M_{Top} = \{m^1_T, m^2_T, \ldots, m^{M_{Top}}_T\}$ be the set of supported modes of $Top$. The MCT is constructed by the following steps:

1. From $N_0$, create $\lambda_{N_0} = |M_{Top}|$ new nodes, such that for each new node $N_i \succ N_0, N_i = \{(Top, m^i_T)\}$ $\{i \in [1,|M_{Top}|]\}$.

2. From each $N_i = \{(Top, m^i_T)\}$ $\{i \in [1,|M_{Top}|]\}$, create $\lambda_{N_i} = |W_{Top,m^i_T}|$ new nodes, such that for each $N' \succ N_i, N' \in W_{Top,m^i_T}$. Moreover, if $\lambda_{N_i} > 1$, then for each $N', N'' \succ N_i$, we have $N' \neq N''$. 25
3. For each node $\mathcal{N} = \{(c_1, m_{c_1}), (c_2, m_{c_2}), \ldots, (c_n, m_{c_n})\}$ ($n \in \mathbb{N}$) with $d_{\mathcal{N}} \geq 2$, if $\forall i \in [1, n]$, $c_i \in \mathcal{PC}$, then $\mathcal{N}$ is marked as a leaf node and no new node is created from $\mathcal{N}$. Otherwise, if $\exists i \in [1, n]$ such that $c_i \in \mathcal{CC}$, then create $\lambda_{\mathcal{N}} = \prod_{i \in [1, n]} |\mathcal{W}_{c_i, m_{c_i}}|$ new nodes, such that for each $\mathcal{N}' \succ \mathcal{N}$, $\mathcal{N}' \notin \bigoplus_{i \in [1, n]} \mathcal{W}_{c_i, m_{c_i}}$. Moreover, if $\lambda_{\mathcal{N}} > 1$, then for each $\mathcal{N}', \mathcal{N}'' \succ \mathcal{N}$, we have $\mathcal{N}' \neq \mathcal{N}''$.

4. Repeat Step 3 until all branches of the MCT have reached the leaf node.

The MCT construction process is implemented as Algorithm 1, which is a recursive function $\text{constructMCT}(\mathcal{N}, d_{\mathcal{N}})$ that has two input parameters: $\mathcal{N}$ is the node currently being explored and $d_{\mathcal{N}}$ is the depth level of $\mathcal{N}$. Initially, $\mathcal{N} = \emptyset$ and $d_{\mathcal{N}} = 0$. We assume that $\text{Top}$ must have subcomponents. Otherwise, $\text{Top}$ itself will be the entire system and mode transformation will be meaningless. Moreover, for each component $c$ running in mode $m$, we assume that $\mathcal{W}_{c,m}$ is an indexed set such that $\mathcal{W}_{c,m}[i]$ represents the $i$th element of $\mathcal{W}_{c,m}$.

The complexity of Algorithm 1 depends on the structure of the component hierarchy, the number of modes of each component, and the mode mappings in the involved components. Since all these contributing factors are bounded, this algorithm will terminate. We have implemented the mode transformation and measured the transformation time (including both steps: deriving the MCT and mode transition graph) for the healthcare monitoring system\textsuperscript{2}. The transformation took 2.9-3.9ms. The computation overhead of the algorithm does not affect runtime performance, as the MCT is constructed at design time. The worst-case combination of factors such as number of components and the number of component modes may lead to a huge number of system modes, increasing the overhead exponentially. However, in practice, the expected number of system modes should be limited. If mode transformation becomes intractable due to extreme computation overhead, this would imply that the system is too complex to adopt centralized mode management. Then it may be more suitable to go for distributed mode management without mode transformation, although especially if the component hierarchy is deep, then the runtime overhead for the required message exchange may be substantial. Alternatively, a better solution could be partial mode transformation, i.e. performing mode transformation within a composite component instead of the entire system. Our mode transformation technique is flexible enough to support partial mode transformation at any component level.

\textsuperscript{2}https://github.com/mcore-ide/mtt
Algorithm 1 \textit{constructMCT}(\mathcal{N}, d_{\mathcal{N}})

1: if $d_{\mathcal{N}} = 0$ then
2: \quad $\lambda_{\mathcal{N}} := |\mathcal{M}_{\text{Top}}|$;
3: \quad for $i$ from 1 to $\lambda_{\mathcal{N}}$ do
4: \quad \quad $\mathcal{N}_i := \{(\text{Top}, m_i)\}$;
5: \quad \quad constructMCT($\mathcal{N}_i$, 1);
6: \quad end for
7: end if
8: if $d_{\mathcal{N}} = 1$ then
9: \quad $\{(\text{Top}, m)\} := \mathcal{N}$;
10: \quad Derive $W_{\text{Top}, m}$;
11: \quad $\lambda_{\mathcal{N}} := |W_{\text{Top}, m}|$;
12: \quad for $i$ from 1 to $\lambda_{\mathcal{N}}$ do
13: \quad \quad constructMCT($W_{\text{Top}, m[i]}$, 2);
14: \quad end for
15: end if
16: if $d_{\mathcal{N}} \geq 2$ then
17: \quad $\{(c_1, m_{c_1}), (c_2, m_{c_2}), \ldots, (c_n, m_{c_n})\} := \mathcal{N}$;
18: \quad if \forall $i \in [1, n] : c_i \in \mathcal{PC}$ then
19: \quad \quad return ;
20: \quad else
21: \quad \quad Derive $W := \bigoplus_{i \in [1, n], c_i \in \mathcal{PC}} W_{c_i, m_{c_i}}$;
22: \quad \quad $\lambda_{\mathcal{N}} := \prod_{i \in [1, n], c_i \in \mathcal{PC}} |W_{c_i, m_{c_i}}|$;
23: \quad \quad for $i$ from 1 to $\lambda_{\mathcal{N}}$ do
24: \quad \quad \quad constructMCT($W[i]$, $d_{\mathcal{N}} + 1$);
25: \quad \quad end for
26: \quad end if
27: end if
As an example, Fig. 12 illustrates the MCT of the monitoring subsystem introduced in Section 1. The MCT consists of 9 nodes $N_0$–$N_8$ with four depth levels. Represented by the respective paths of the MCT, the three identified system modes are:

$m_1 = N_0 \cup N_1 \cup N_3 \cup N_6$

$= \{(MoS, Rm), (DaD, R1), (MuD, Rd), (EvA, D), (VAE, D), (ViD, Rvd), (AuD, D)\}$

$m_2 = N_0 \cup N_2 \cup N_4 \cup N_7$

$= \{(MoS, Att), (DaD, R1), (MuD, Ed), (EvA, R2), (VAE, R3), (ViD, Evd), (AuD, Rad)\}$

$m_3 = N_0 \cup N_2 \cup N_5 \cup N_8$

$= \{(MoS, Att), (DaD, R1), (MuD, Dq), (EvA, R2), (VAE, R3), (ViD, Evd), (AuD, D)\}$

Figure 12: The mode combination tree of the monitoring subsystem

Since we assume that the Monitoring Subsystem initially runs in mode $Rm$, $m_1$ is the initial system mode after mode transformation. Fig. 13 shows the configurations of the three system modes based on the component connections in Fig. 1.

An interesting property can be observed from an MCT, formulated as the following theorem:

**Definition 8.** A non-root node $N$ of an MCT is represented by a valid mode combination of components with depth level $d_N - 1$.

**Proof.** Since $N$ cannot be the root node, $d_N \geq 1$. When $d_N = 1$, $N = \{(Top, m)| m \in M_{Top}\}$ and $d_{Top} = d_N - 1 = 0$. When $d_N = 2$, according to Step 2 of the MCT construction procedure, $N \in \mathcal{W}_{Top,m}$ ($m \in M_{Top}$). The definition of $\mathcal{W}$ implies that $N$ is represented by a valid LMC of the subcomponents of $Top$, while each subcomponent of $Top$ has a depth level $d_N - 1 = 1$. Note that for $d_N \geq 3$, $N$ is created by the same rule (Step
3). Hence Theorem 8 can be proved by induction for $d_N \geq 3$. Suppose $\mathcal{N} = \{ (c_1, m_{c_1}), \cdots, (c_n, m_{c_n}) \} (n \in \mathbb{N})$.

**Basis:** When $d_N = 3$, according to Step 3 of the MCT construction procedure, the definition of $\mathcal{W}$, and the definition of the operator $\bigoplus$, $\mathcal{N} \in \bigoplus_{c_j \in \mathbb{C}} \mathcal{W}_{c_j, m_{c_j}}$. Then for all $(c_j, m_{c_j}) \in \mathcal{N}$, $d_{c_j} = d_N - 1 = 2$.

**Inductive step:** Suppose that when $d_N = k$, for all $(c_j, m_{c_j}) \in \mathcal{N}$, we have $d_{c_j} = k - 1$. Then we need to prove that for another node $\mathcal{N}' = \{ (c'_1, m_{c'_1}), \cdots, (c'_o, m_{c'_o}) \} (o \in \mathbb{N})$ with $d_{c'_i} = k + 1$, we have $d_{c'_i} = k$ ($i \in [1, o]$) for all $(c'_i, m_{c'_i}) \in \mathcal{N}'$. The node $\mathcal{N}'$ can be created either from $\mathcal{N}$ or from another node $\mathcal{N}''$ with $d_{\mathcal{N}''} = d_N$. Since $\mathcal{N}$ and $\mathcal{N}''$ include the same set of components with depth level $k - 1$, we always have $\mathcal{N}' \in \bigoplus_{d_{c_j} = k - 1, \ c_j \in \mathbb{C}} \mathcal{W}_{c_j, m_{c_j}}$.

By the definition of $\mathcal{W}$ and the operator $\bigoplus$, all $c'_i$ included in $\mathcal{N}'$ have a depth level equal to $k$, i.e. $d_{\mathcal{N}'} = 1$.

The induction above together with our analysis for $d_N = 1$ and $d_N = 2$ proves Theorem 8.

Theorem 8 further implies the following corollary:

**Definition 9.** All leaf nodes of an MCT have the same depth level.

**Proof.** Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two leaf nodes of the same MCT. By Theorem 8, both $\mathcal{N}_1$ and $\mathcal{N}_2$ include the same set of primitive components with the maximum depth level of the component hierarchy. Let $d$ be the depth level...
of these primitive components. Then \( d = d_{N_1} - 1 = d_{N_2} - 1 \). Therefore, 
\( d_{N_1} = d_{N_2} \), thus proving the corollary.

Once the MCT is constructed, the system modes can be derived as the set of paths from the root node to the leaf nodes of the MCT. Let \( N^k \) be the set of nodes of an MCT with depth level \( k \). Then,

**Theorem 1.** Given an MCT, a system mode \( m \) is represented by a valid mode combination \( \bigcup_{i=0}^{\delta} N_i \) where \( N_i \in N^i \) is a node representing a valid mode combination of components with depth level \( i - 1 \), \( N_{i+1} \) is a node created from \( N_i \), and \( \delta \) is the maximum depth level of the MCT, i.e. \( N_\delta \) is a leaf node. The total number of system modes is equal to the total number of leaf nodes of the MCT.

**Proof.** Let \( C = \{c_1, c_2, \ldots, c_n\} \ (n \in \mathbb{N}) \) be the set of components of a system. First we prove that \( \bigcup_{i=0}^{\delta} N_i \) includes the modes of all components. Then we prove that each component is included in \( m \) only once. Finally, we prove the valid mode combination as a result of \( \bigcup_{i=0}^{\delta} N_i \).

By Theorem 8, each \( N \neq N_0 \) with \( d_N = k \) is represented by a valid mode combination of components with depth level \( k - 1 \). Since the maximum depth level of the MCT is \( \delta \), the maximum depth level of a component in the system is \( \delta - 1 \). Each path of the MCT is a union \( \bigcup_{i=0}^{\delta} N_i \) that includes the mode combination of components with depth levels from 0 to \( \delta - 1 \). Since the depth level of any component is within \([0, \delta - 1]\), the mode of the component must be included in \( \bigcup_{i=0}^{\delta} N_i \), i.e., a path of the MCT.

Next consider any two pairs \((c_i, m_{c_i}) \in m \text{ and } (c_j, m_{c_j}) \in m \ (i, j \in [1,n])\). If \( d_{c_i} \neq d_{c_j} \), then apparently \( c_i \neq c_j \). If \( d_{c_i} = d_{c_j} = k \), then \((c_i, m_{c_i})\) and \((c_j, m_{c_j})\) must be both included in any node \( N \in N^{k+1} \) by Theorem 8. Components \( c_i \) and \( c_j \) may or may not share the same parent. If their parents are different, then apparently \( c_i \neq c_j \). If they share the same parent, by Definition 6, we also have \( c_i \neq c_j \). Therefore, \( c_i \neq c_j \) always holds regardless of the depth levels of \( c_i \) and \( c_j \). Stated otherwise, each component is included in \( m \) only once.

This theorem states that for each \( i \) in \( \bigcup_{i=0}^{\delta} N_i \), \( N_{i+1} \) is a node created from \( N_i \). By Theorem 8 and the MCT construction process, this implies that \( N_i \cup N_{i+1} \) represents a valid mode combination of components with depth level \( i - 1 \) and \( i \). Likewise, at one level down, \( N_{i+1} \cup N_{i+2} \) represents a valid mode combination of components with depth level \( i \) and \( i + 1 \). Hence \( N_i \cup N_{i+1} \cup N_{i+2} \) represents a valid mode combination of components with
depth level \( i, i + 1 \) and \( i + 2 \). By this induction from \( i = 0 \) to \( i = \delta \), \( \bigcup_{i=0}^{\delta} N_i \) represents a valid mode combination of all components, i.e., a system mode \( m \).

Furthermore, we know that each path of an MCT includes a leaf node. Since all leaf nodes are different, all paths are also different. The number of paths, i.e. the number of system modes, equals the number of leaf nodes.

Among the system modes, the initial system mode can be recognized based on the specification of the initial modes of all components.

### 3.2 Deriving the mode transition graph

Once the system modes are identified from the constructed MCT, the next step is to derive the mode transition graph on top of these system modes based on the definition of mode-switch events. We assume that a mode-switch event is triggered by a component \( c \) requesting to switch mode \( m_1^c \) to a new mode \( m_2^c \), denoted as \( c : m_1^c \rightarrow m_2^c \). The triggering of each mode-switch event \( k \) may lead to the mode switches of some other components in the same system. For a system with a set of identified system modes \( M = \{m_1, m_2, \cdots, m_n\} \) (\( n \in \mathbb{N} \)), a mode switch is a transition from \( m_{old} \) to \( m_{new} \), where \( m_{old}, m_{new} \in M \) and \( m_{old} \neq m_{new} \). A mode transition graph contains all the possible transitions between these system modes and associates each transition with the corresponding mode-switch event. Similar to an MMA, each state of a mode transition graph can be graphically represented by a circle, with the initial state being marked by a double circle. A graphical illustration of mode transition graph can be found in Fig. 14.

The key issue of deriving the mode transition graph is to identify the system modes \( m_{old} \) and \( m_{new} \) for each mode-switch event for which a system mode switch is possible. Consider a mode-switch event \( k \) identified as \( c : m_1^c \rightarrow m_2^c \). The only condition satisfying the triggering of \( k \) is that the triggering source \( c \) is currently running in mode \( m_1^c \). For each \( k \), \( m_{old} \) can be easily identified as long as \( (c, m_1^c) \in m_{old} \). Note that more than one system mode could be identified as \( m_{old} \). Depending on the current system mode, a mode-switch event may enable different transitions.

In contrast to \( m_{old} \), only one system mode can be the \( m_{new} \) for each mode-switch event \( k \). The identification of \( m_{new} \) for \( k \) is more difficult because it depends not only on \( m_2^c \), but also on the target modes of the other components. We identify the \( m_{new} \) for each mode-switch event with the assistance of a Component Target Mode (CTM) table. A CTM table is a table with \( n_1 \) rows and \( n_2 \) columns, where \( n_1 \) is the number of components of a system and \( n_2 \) is the number of mode-switch events. An example of a CTM table can be found above the mode transition graph in Fig. 14. In the CTM table, each row is associated with a component, each column is associated
with a mode-switch event, and each cell contains the target mode \( m_c \) of the corresponding component \( c \) for the corresponding mode-switch event \( k \). The cell with \( X \) indicates that \( m_c \) is independent of \( k \).

A CTM table can be automatically constructed offline based on the list of mode-switch events and the mode mapping of each composite component. Let \( m^k_c \) be the target mode of \( c \) for \( k \) in a CTM table. Taking advantage of the CTM table, the new system mode \( m_{\text{new}} \) for each mode-switch event \( k \) can be identified as follows: For each system mode \( m = \{(c_i, m_{c_i})|i \in [1, n], n \in \mathbb{N}\} \), if \( \forall i \) where \( m^k_{c_i} \neq X \) in the CTM table (i.e., \( k \) leads to the mode switch of \( c_i \) to a new mode \( m^k_{c_i} \)), we have \( m_{c_i} = m^k_{c_i} \), then \( m \) is the \( m_{\text{new}} \) for \( k \).

Algorithm 2 describes the process of building the mode transition graph, with a search space of \( O(|M| \cdot |K|) \).

---

**Algorithm 2** \( \text{constructMTG}(C, M, K) \)

1: \( C = \{c_1, \ldots, c_o\} \) (\( o \in \mathbb{N} \)); \{The set of all components\}
2: \( M = \{m_1, \ldots, m_n\} \) (\( n \in \mathbb{N} \)); \{The set of identified system modes\}
3: \( K = \{k_1, \ldots, k_l\} \) (\( l \in \mathbb{N} \)); \{The set of all mode-switch events\}
4: for all \( k_i \in K \) where \( k \in [1, l] \) and \( k_i = c : m^1_c \rightarrow m^2_c \) do
5: if \( \exists m_j \in M \) s.t. \( \forall c_p \in C \) and \( m^{k_i}_{c_p} \neq X, (c_p, m^{k_i}_{c_p}) \in m_j \) then
6: \( m_{\text{new}} = m_j \);
7: for all \( m_j \in M \) do
8: if \( (c, m^1_j) \in m_j \) then
9: addTransition\((m_j, m_{\text{new}}, k_i)\); \{Add a transition from \( m_j \) to \( m_{\text{new}} \) labeled with \( k_i \}\}
10: end if
11: end for
12: end if
13: end for

Fig. 14 presents the workflow for deriving the mode transition graph of the monitoring subsystem. The CTM table is derived based on two inputs: (1) the mode mapping of composite components MoS and MuD specified by MMA, and (2) the possible mode-switch events. For this example, four mode-switch events, from \( k_1 \) to \( k_4 \), are specified at design time. \( k_1 \) and \( k_2 \) are triggered by MoS for switching between modes \( Rm \) and \( Att \), while \( k_3 \) and \( k_4 \) are triggered by MuD for switching between modes \( Ed \) and \( Dq \). The target modes of all components in the monitoring subsystem for all mode-switch events are listed in the CTM table. Previously the MCT in Fig. 12 has identified three system modes \( m_1 \) (the initial mode), \( m_2 \), \( m_3 \). The CTM table additionally adds transitions between the system modes based on each possible mode-switch event, thereby yielding the mode transition graph in Fig. 14. The mode transition graph helps the global mode-switch manager to keep track of the current system mode and makes the system switch to the right target mode when a mode switch is triggered.

Using mode transformation, the number of identified system modes is sensitive to the number of modes of each single component and the mode
Figure 14: Deriving the mode transition graph of the monitoring subsystem

mapping of each composite component. As a consequence, mode transformation may end up with a huge number of system modes. Nonetheless, it only makes sense to distinguish one mode from another mode when the system behaviors are noticeably different in these modes. Depending on the application, it could be more efficient to merge several modes with similar global configurations into one mode. For instance, the two system modes $m_2$ and $m_3$ in Fig. 13 are fairly similar to each other, since the only difference is the activation/deactivation of component AuD. Therefore, it would be more efficient to merge these modes and toggle the status of AuD whenever necessary without mode switch. The criteria for merging system modes are application-dependent and out of the scope of the report. Nevertheless, we believe that it is possible to partially automate the merging of system modes in a later optimization phase by certain application-independent merging rules.

Finally, a potential drawback of mode transformation is the loss of potential concurrency between local mode managers. If multiple mode-switch events are triggered concurrently and affect disjoint sets of components, distributed mode management before mode transformation allows that these mode-switch events can be handled concurrently, whereas different mode-switch events have to be sequentially handled by the global mode-switch manager. Nonetheless, the centralized mode management after mode transformation eliminates inter-component communication, which is a complex process [9]. Hence mode transformation is still more likely to yield faster mode switch.
3.3 Mode transformation verification

In this section, we verify two key properties of our mode transformation technique in terms of correctness and completeness. First, we need to guarantee that the system modes identified from the MCT are correct and complete. An identified system mode is correct if it is a valid mode combination of the modes of all components as per the mode mapping. The set of identified system modes is complete if it includes all possible system modes as per the mode mapping.

**Theorem 2.** The system modes identified from the MCT are correct and complete, and each identified system mode is unique.

**Proof.**

- Correctness: The correctness is already proven by Theorem 1.
- Completeness: The completeness of system modes depends on the completeness of the nodes of an MCT. By the MCT construction process, particularly Line 10 and Line 21 of Algorithm 1, whenever we create new nodes from a node $N = \{(c_i, m_{c_i})|i \in [1,n], n \in \mathbb{N}\}$, we consider all the possible valid LMCs of each $c_i$ for $m_{c_i}$ and all the possible combinations of their valid LMCs. Hence an MCT has a complete set of nodes which further guarantee a complete set of identified system modes.

- Uniqueness: Let $m_1$ and $m_2$ be two different system modes. By Theorem 1, $m_1$ and $m_2$ correspond to two different paths of the same MCT. Then there must exist a node $N$ where the two paths diverge. Let $N_1$ and $N_2$ be the two nodes created from $N$ such that the path of $m_1$ follows $N_1$ and the path of $m_2$ follows $N_2$. Then $N_1$ and $N_2$ must be different because all new nodes created from $N$ must be different. Therefore, $N_1 \in m_1$ and $N_1 \notin m_2$ while $N_2 \in m_2$ and $N_2 \notin m_1$, implying that $m_1$ and $m_2$ are unique.

Next we verify the correctness of the mode transition graph, formulated as follows:

**Theorem 3.** The transitions of a mode transition graph are correct and complete.

**Proof.** We divide the proof into two parts. First we prove the correctness of the transitions of a mode transition graph. Then we prove the completeness of the transitions of a mode transition graph. Both parts can be proved by contradiction. Let $\mathcal{M} = \{m_1, \ldots, m_n\}$ be the set of identified system modes, where $m_i, m_j \in \mathcal{M}$ and $k = c_o : m_{c_o}^1 \rightarrow m_{c_o}^2$.

To prove the correctness, suppose a mode transition graph contains a wrong transition $L = m_i \xrightarrow{k} m_j$ (i.e. a transition from $m_i$ to $m_j$ for $k$) where either $m_i$ fails to satisfy the triggering condition of $k$ or $m_j$ is the wrong target system mode for $k$. By Algorithm 2, we know that $(c_o, m_{c_o}^1) \in m_i$. Hence $m_i$ must satisfy the triggering condition of $k$. Moreover, $m_j$ must
be the only system mode that matches the CTM table. The correctness of the identified system modes is guaranteed by Theorem 2. The construction of the CTM table is independent of our mode transformation technique. Instead, it only depends on mode mapping. We assume that the mode mappings of all composite components are correctly specified, thus ensuring the correctness of the CTM table, which further proves that \( m_j \) is the right target system mode for \( k \). This contradicts the assumption that \( L \) is a wrong transition. Therefore, the correctness of the transitions of a mode transition graph follows.

To prove the completeness, suppose a mode transition graph misses a transition \( L = m_i \xrightarrow{k} m_j \) which is supposed to be included. First, \( m_i \) must satisfy the triggering condition of \( k \), implying \((c_o, m_{c_o}^{1}) \in m_i\). Algorithm 2 ensures that all system modes satisfying this condition can be identified as the mode with an outgoing transition labeled with \( k \). Moreover, from the proof of the correctness above, we know that there is only one target system mode for \( k \) which can be correctly identified. Hence \( m_j \) must be the target system mode for \( k \), with an incoming transition labeled with \( k \). Since both \( m_i \) and \( m_j \) can be identified in association with \( k \), it is impossible for a mode transition graph to miss the transition \( L \). Therefore, the completeness of a mode transition graph follows.

4 MCORE: a prototype implementation

We have developed a prototype tool MCORE\(^3\), the Multi-mode COmpo-nent Reuse Environment, which supports the modeling of multi-mode sys-tems with multi-mode components by integrating the mode mapping mech-anism and mode transformation technique. MCORE was developed as a JointJS \([12]\) plugin. JointJS is a modern HTML 5 JavaScript library for visualization and interaction with diagrams and graphs. In MCORE, a user can develop a multi-mode component from scratch, save it in a library for future reuse, or reuse an existing component in the library.

Compared with other component-based development tools, a distinguishing feature of MCORE is the reuse of multi-mode software components. As far as we know, MCORE is the first (and possibly only) tool for building multi-mode systems with multi-mode components. Fig. 15 shows how the composite component MuD of the monitoring subsystem is internally mod-eled in the workspace of MCORE. On top of the workspace is a navigation bar with a \textit{New Component} button to the left and an Inspector menu to the right. The \textit{New Component} button is used to create new components. The menu provides a list of functions such as saving a component in the library and exporting the system model as xml files. In addition, navigation information is displayed in the middle of the navigation bar. For instance,

\(^3\)https://github.com/mcore-ide/ide
MuD in the navigation bar of Fig. 15, i.e., the current view in the workspace is within MuD.

![Figure 15: The workspace of MCORE](image)

In MCORE, the supported modes of a component is represented by tags located at its top right corner. The mode mapping of MuD can be manually specified by clicking the *Mode Mapping* button next to the *New Component* button in the navigation bar. After that the *Mode Mapping* button will be switched to a *Done* button. Then a user can map the modes of MuD and its subcomponents by clicking their modes. Each mode of MuD can be mapped to multiple modes of a subcomponent. If a subcomponent has no modes mapped to MuD, it is considered to be deactivated. Mode mapping is completed when the *Done* button is clicked.

The rightmost Inspector panel in Fig. 15 displays the local configuration of MuD, including its supported modes, port names, and properties. Depending on the actual requirement, a user can define different properties. For instance, Fig. 15 indicates that MuD has a property WCET (Worst-Case Execution Time) equal to 50 and another property memory usage with the value 25. Right below the configuration window is the library for saving and reusing components.

MCORE is not a stand-alone tool for the development of multi-mode systems. Instead, it functions as a preprocessor for Rubus ICE [7], which is an IDE for the Rubus component model [13] developed by Arcticus Systems[^4] in Sweden. Rubus ICE is a commercial tool in use at several companies, includ-

ing Volvo Construction Equipment, BAE Systems, Elektroengine, Haldex, and Hoerbiger. As an industrial component model, Rubus is targeting the component-based development of vehicular systems. Rubus supports multi-mode systems, however, modes can only be specified at system level and the reuse of multi-mode components is not supported. This limitation can be complemented by MCORE. The system model built by multi-mode components in MCORE is in compliance with the Rubus component model after mode transformation. Hence, depicted in Fig. 16, the system model designed in MCORE can be exported as xml files to be imported to Rubus ICE for further analysis, test and code generation. The development of MCORE has included several meetings with Arcticus Systems who confirms the practical value of this prototype implementation.

Figure 16: The workflow of MCORE together with Rubus ICE

5 Related work

The extended MECHATRONICUML [14] (EUML) allows the hierarchical composition of reconfigurable components, which are comparable to our multi-mode components. EUML introduces an additional reconfiguration port for each component, which resembles the dedicated mode-switch ports of a multi-mode component. In EUML, the reconfiguration of a composite component is handled by two dedicated subcomponents which play similar roles as the local mode-switch manager of a multi-mode component. Unlike our approach, EUML does not pre-define component configurations at design time, thus allowing more flexible reconfiguration at runtime. In contrast, we emphasize mode-switch predictability by assuming that all component modes, the corresponding configurations, and mode mappings are specified at design time.

Pop et al. propose an Oracle-based approach [15] that also supports the reuse of multi-mode components. The basic idea is to abstract component
behaviors into a global property network. The mode of each component is modeled as a property dependent on other property values. A single property change is propagated throughout the property network, potentially leading to the value change of other properties. Then component modes are updated top-down. A finite-state machine called Oracle is offline constructed (similar to our mode transformation) to guarantee predictable update time of the property network. The mapping between component modes is however not systematically specified in the Oracle-based approach.

Weimer et al. propose a set of input-output blocks for building multi-mode systems [16]. Each multi-mode component contains a set of Mode Blocks (MBs) while each MB includes all the components used for the corresponding mode. The mode switch of a component is achieved by switching the currently selected MB controlled by a Supervisor Block (SB). These blocks were implemented in Simulink [17]. Another work similar to this is the mode-oriented design [18] in Gaspard2 [19]. A multi-mode component is represented by a macro component which consists of a state graph component and one or more mode switch components. A mode switch component plays the same role as the MB in [16]. Both approaches in [16] and [18] use completely different components for different modes, whereas in our approach it is possible to share some components and connections in different modes. Hence our approach is more suitable for reuse of multi-mode components.

Mode switch has been addressed in a number of component models, e.g. SaveCCM [20], COMDES-II [21] and MyCCM-HI [22], to name a few. There are also some other component models which have been commercialized, e.g. Koala [23] (targeting consumer electronics) and Rubus [13] (targeting ground vehicles). These component models have different notions of mode-switch handling. For instance, in Koala and SaveCCM, a special connector switch is introduced to achieve the structural diversity of a component. Depending on the input data, switch can select one of multiple outgoing connections. COMDES-II uses a state-machine component to switch component configurations in different modes. In Rubus, mode is treated as a system property. A system-wide static configuration of components is defined for each mode. MyCCM-HI provides a more advanced mechanism for handling mode switch. Each MyCCM-HI component is mode-aware and is associated with a mode automaton which implements its mode-switch mechanism. In addition, Fractal [24] is a component model supporting component reconfiguration. Each Fractal component has a membrane (a container for local controllers) that is able to control the reconfiguration of the component.

Mode switch has also been covered by some programming and specification languages, such as AADL [25], Giotto [26], TDL [27], the extended Darwin [28], and Mode-automata [29]. In AADL, a state machine is used to represent the mode-switch behavior of a component. Each state machine consists of a number of states (modes), transitions between these states (mode switches) and input/output event ports used for mode-switch trigger-
ing. Both Giotto and TDL are time-triggered languages for embedded programming, which requires periodic checking of conditions to decide whether to trigger a mode switch or not. The extended Darwin [28] extends the existing Architecture Description Language Darwin [30] by incorporating the notion of mode. The mode of a composite component is directly related to the modes of its subcomponents. Yet the mapping between modes is unclear in [28]. Mode-automata is a programming model supporting the description of running modes of reactive systems. The behavior of a system is a sequence of modes, each of which corresponds to a collection of execution states. Our MMA differs from mode-automata in the sense that mode-automata specifies the hierarchical structure of system-wide modes, whereas MMA specifies the relation between modes of a composite component and the modes of its subcomponents.

Dynamic Software Product Lines (DSPLs) [31, 32] is an emerging technique for developing adaptive systems. DSPLs originates from the conventional Software Product Lines (SPLs) [33] which has been successful in producing a family of software systems. Different systems configured from the same SPL share certain common features, whereas the SPL uses variation points to distinguish the unique features of each system. DSPLs allows the binding of variation points at runtime so that a system can dynamically change configurations on the fly to accommodate to the changing environment. To the best of our knowledge, DSPLs only considers global system configurations without considering reuse of adaptive software components.

Different types of automata have been proposed for component-based systems and multi-mode systems. For instance, constraint automata [34] is used to model the functional coordination of components, thereby enabling the formal verification of coordination mechanisms. Besides, multi-mode automata [35] is intended for compositional analysis of multi-mode real-time systems. The MMA presented in this report serves as a formalism for a unique and dedicated purpose: mode mapping, which to our knowledge has not been addressed by other existing automata.

6 Conclusions and future work

In this report, we propose component-based software development of multi-mode systems, characterized by reuse of multi-mode components, i.e., components which can run in different modes and switch mode guided by a local mode-switch manager. We present a mode mapping mechanism which enables each composite component to represent the relation between its own modes and the modes of its subcomponents by a set of Mode Mapping Automata. Mode mapping is then complemented by a mode transformation technique that transforms component modes to system modes in the system modeling phase for centralized mode management which is more efficient.
than distributed mode management, since it eliminates the need for inter-component communication to coordinate a mode switch, thereby reducing runtime mode-switch overhead and shortening mode-switch time. The mode mapping mechanism and mode transformation technique have been implemented in a prototype tool MCORE (the Multi-mode COmponent Reuse Environment). As a preprocessor to the commercial tool Rubus ICE developed by Arcticus Systems, MCORE enables the modeling of multi-mode systems by multi-mode components and performs a subsequent mode transformation. The system model after mode transformation by MCORE can be exported to Rubus ICE for further analysis, test and code generation.

Currently, some remaining efforts need to be invested to fully integrate MCORE with Rubus ICE. For instance, our mode transformation has been implemented separately along with MCORE and we need to integrate the mode transformation implementation into MCORE. Another task to be completed is exporting the model in xml format to be used by Rubus ICE. Moreover, it is our ambition to explore the applicability of our component-based development of multi-mode systems in a real-world system in addition to the proof-of-concept healthcare monitoring system introduced in this report.

Acknowledgements

This work is funded by the Swedish Research Council via the ARROWS project at Mälardalen University. The authors would like to thank Arcticus Systems for the discussions and support.

References


