Mode Switch Handling for the ProCom Component Model

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

Hongwan Qin
Lund University
Lund, Sweden
mas09hqi@student.lu.se

ABSTRACT

Component-Based Software Engineering has been deemed a suitable technique for the development of complex embedded systems, as component reuse makes it easier to manage software complexity. Another way of reducing software complexity is by partitioning system behavior into different operational modes. Such a multi-mode system can change its behavior by switching between modes. For a multi-mode system built by components, a challenge is its mode switch handling.

In this paper, a novel approach is presented to integrate our mechanism for handling mode switch (the Mode Switch Logic), in ProCom, which is a component model designed for the development of real-time embedded systems. The outcome is a slightly extended version of ProCom which not only supports the development of multi-mode applications, but also is able to handle mode switch.

Categories and Subject Descriptors

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

Keywords

ProCom; component; mode switch

1. INTRODUCTION

The growing complexity of the software of embedded systems entails new techniques for the development of complex embedded systems, as traditional techniques are becoming less suitable. Component-Based Software Engineering (CBSE) [4] is a promising paradigm for developing complex systems by virtue of its benefits such as the management of software complexity, reduced time to market and improved software quality. CBSE allows a system to be built by reusable components which are independently developed so that the system does not have to be developed from scratch. The success of CBSE has been evidenced by a variety of component models proposed both in industry and academia [5] [13]. Among these component models, and in the focus of this paper, ProCom [3] is a component model for real-time and embedded systems, particularly targeting vehicular, automation and telecommunication applications.

In contrast to CBSE, another common approach to reducing software complexity of embedded systems is to partition system behavior into different operational modes. A multi-mode system can start running in a default mode and switch to another mode under certain circumstances. A representative example is the control software of an airplane, which could run in the modes *taxi* (the initial mode), *taking off*, *flight* and *landing*. Different subsystems are running in different modes. For instance, the subsystem for controlling the wheels only runs in *taxi* mode whereas the navigation subsystem may only run in *flight* mode. Combining CBSE and multi-mode systems, we get a Component-Based Multi-Mode System (CBMMS), i.e. a multi-mode system developed in a component-based manner. Figure 1 illustrates a conceptual CBMMS, with its component hierarchy on the left and its component connections on the right. The system, i.e. Component *Top*, consists of three components: *a*, *b* and *c*. Component *b* is composed by *d* and *e*. Components *a*, *c*, *d* and *e* are primitive components because they cannot be further decomposed. Components *Top* and *b* are composite components because they are both compositions of other components. Since the component hierarchy has a tree structure, a composite component and its subcomponents have a parent-and-children relationship. For instance, *b* is the parent of *d* and *e*, which in turn are the children of *b*. Moreover, the system can run in two modes: m^1_{Top} and m^2_{Top}. When the system is in m^1_{Top}, Component *c* is deactivated (i.e. not running), shown in the component hierarchy in Figure 1 by not displaying *c* in mode m^1_{Top}. In contrast, when the system is in m^2_{Top}, *c* is activated whilst *e* is deactivated. Besides, Component *a* has different mode-specific behaviors represented by black and grey colors in Figure 1.

A key issue of a CBMMS is its mode switch handling. A mode switch may amount to the joint mode switches of many different components. For instance, a system mode switch from m^1_{Top} to m^2_{Top} in Figure 1 requires the activation of *c*, the deactivation of *e* and the behavior change of *a*. The mode switches of different components must be well synchronized and coordinated to guarantee a correct system mode switch. For that reason, we have developed the Mode

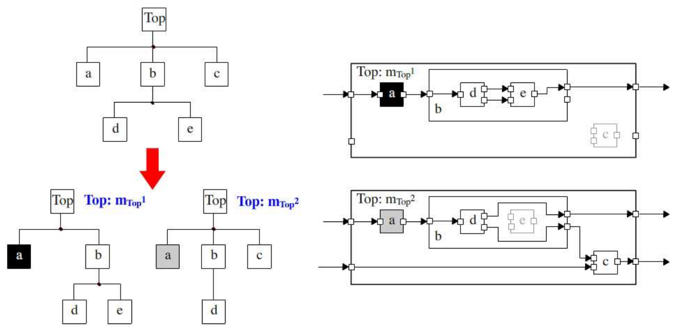


Figure 1: A component-based multi-mode system

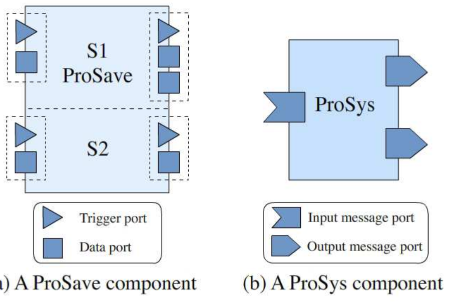


Figure 2: ProSave and ProSys components

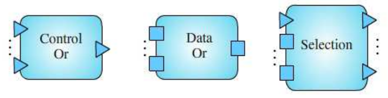


Figure 3: Typical connectors in ProSave

Switch Logic (MSL) [8] [7], a mechanism for handling the mode switch of CBMMSs.

With the ProCom component model and MSL as two background techniques, this paper provides a theoretical guidance for implementing MSL in ProCom. Currently, ProCom does not support multi-mode systems. However, the approach presented in this paper realizes the development of CBMMSs together with their mode switch handling in ProCom. The remainder of the paper is organized as follows: Section 2 introduces the ProCom component model. Section 3 gives a brief introduction of MSL. As the main contribution of the paper, Section 4 describes how MSL is implemented in ProCom. In Section 5, an example is used to illustrate the major elements in Section 4. Related work is reviewed in Section 6. Finally, Section 7 concludes the paper and discusses some future work.

2. THE PROCOM COMPONENT MODEL

ProCom [3] is a component model for the development of distributed real-time and embedded systems software. Compared with other existing component models, the most distinctive feature of ProCom is its two layers: ProSave—the lower layer, and ProSys—the higher layer. With different concerns, these two layers allow a system to be modeled at different levels of granularity. Next we shall give a brief introduction of each layer.

2.1 The ProSave layer

The ProSave layer is used to design subsystems allocated to a single physical node. It is based on a pipe-and-filter architectural style and has clear separation between control flow and data flow. A component belonging to this layer is called a ProSave component. A ProSave component can provide one or more services, each of which realizes a particular functionality. Each service has a single input port group and one or more output port groups. A port group consists of a trigger port and one or more data ports, with the trigger port dedicated to control flow and the data ports dedicated to data flow.

A ProSave component is passive in the sense that the execution of each of its services requires external activation. For each service S of a ProSave component, when the input trigger port is activated, S becomes active and performs computation based on its input data ports. After completing the computation, S writes the result to its output data ports, activates its output trigger port(s) and then becomes passive.

Figure 2(a) depicts a ProSave component with two services S_1 and S_2. Service S_1 has an input port group (consisting of an input trigger port and an input data port) and an output port group (consisting of an output trigger port and two output data ports). The ports of S_2 can be explained in the same way.

The communication between ProSave components is based on a single directional one-to-one connection between ports of the same types. An output trigger/data port of a ProSave component is directly connected to an input trigger/data port of another ProSave component. In addition, ProCom defines a couple of connectors for more advanced communication in ProSave. Figure 3 lists some common connectors that will be used in this paper:

- Control Or: It has multiple input trigger ports and one output trigger port. Its output trigger port is activated when any one of its input trigger ports is activated.
- Data Or: It has multiple input data ports and one output data port. The data arriving at any one of its input data ports is forwarded to its output data port.
- Selection: It has an input trigger port, at least one input data port and multiple output trigger ports. When its input trigger port is activated, it will activate exactly one of its output trigger ports according to the data written to its input data port(s).

The ProSave layer is hierarchical as a composite ProSave component can be composed by other ProSave components.

2.2 The ProSys layer

The ProSys layer is used to construct distributed subsystems. A component belonging to this layer is called a ProSys component. A ProSys component has a number of input and output message ports. Figure 2(b) depicts a ProSys component with one input message port and two output message ports. The communication between ProSys components is realized by asynchronous message passing. A message is sent from an output message port and received from an input message port via message channels. A message channel

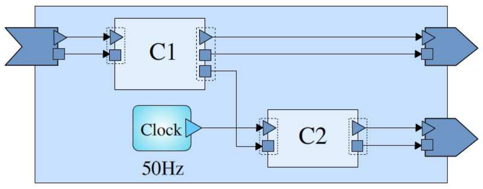


Figure 4: A ProSys component composed by ProSave components

can be associated with multiple input and output message ports, enabling many-to-many communication.

A ProSys component is active, as it has its own threads. Therefore, concurrent execution is allowed in ProSys. Just like ProSave, ProSys is also hierarchical in the sense that a composite ProSys component can be composed by other ProSys components.

The integration of ProSys and ProSave is realized by building a ProSys component with ProSave components, illustrated in Figure 4. In order to map the pipe-and-filter architecture to message passing, a message port is internally treated as a pair of a trigger port and a data port. In addition, a special connector *Clock* can be used for the periodic activation of ProSave components composing the ProSys component.

3. THE MODE SWITCH LOGIC

The Mode Switch Logic (MSL) [8] [7] is a systematic approach to the mode switch handling of CBMMSs. The major elements of MSL include a mode-aware component model, a mode mapping mechanism and a mode switch runtime mechanism. The following briefly introduces these elements.

The mode-aware component model defines essential features that a component should possess in order to support both individual mode switch and cooperative mode switch with other components. Illustrated in Figure 5, a component can support multiple modes and has a unique configuration defined for each mode. Controlled by the mode switch runtime mechanism of MSL, the mode switch of a component is realized by its reconfiguration, i.e. changing its configuration in the current mode to a new configuration in the target mode. Furthermore, to enable cooperative mode switch, dedicated mode switch ports are introduced for the cross-layer communication in the component hierarchy. A multi-mode primitive component has a dedicated mode switch port p^{MSX}, which is used to exchange mode related information with its parent during a mode switch. A multi-mode composite component has two dedicated mode switch ports: apart from p^{MSX} that has the same role as for primitive components, the other one is p_{in}^{MSX}, used to exchange mode related information with its subcomponents during a mode switch.

MSL also provides a mode mapping mechanism (see Chapter 4 in [7] for details) for the composition of multi-mode components and the derivation of the new mode for each component during a mode switch. Usually a multi-mode component is independently developed without knowing the context where it will be used. For a multi-mode composite component c_i, the mapping between the modes of c_i and its subcomponents must be properly specified. In other words,

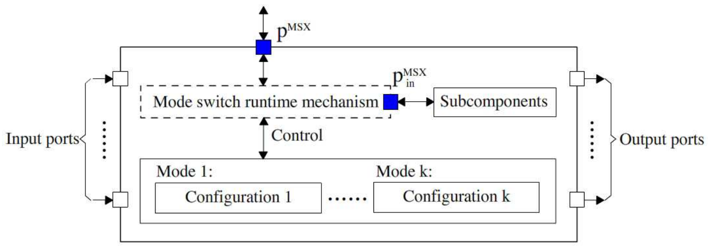


Figure 5: The mode-aware component model

c_i must be able to monitor and update the modes of itself and its subcomponents.

The mode switch runtime mechanism handles the mode switch of a CBMMS and the mode switches of its components at runtime. This mechanism includes two fundamental elements: the Mode Switch Propagation (MSP) protocol and the mode switch dependency rule. The MSP protocol specifies how a mode switch event is detected by an individual component and efficiently propagated to other related components. How and why a mode switch event is generated is outside the scope of the MSP protocol, and is something that from the perspective of the MSP protocol is handled by the code implementing the corresponding component. The mode switch dependency rule guarantees the mode consistency between a system and its components after each mode switch. Both elements of the mode switch runtime mechanism are based on the transmission of downstream and upstream primitives throughout the component hierarchy. A downstream primitive is sent from a composite component to its subcomponents via its dedicated mode switch port p_{in}^{MSX}. An upstream primitive is sent from a component to its parent via its dedicated mode switch port p^{MSX}. Due to limited space, the complete description of the mode switch runtime mechanism will not be presented here (see Chapter 3 in [7] for details).

4. IMPLEMENTING MSL IN PROCOM

In previous sections, the ProCom component model and MSL have been introduced separately. In this section, we describe the contribution of this paper—implementing MSL in the ProCom component model. The basic idea of our approach is to integrate the key elements of MSL in ProCom with minimum modification to ProCom. First, a ProCom component must be made mode-aware to become consistent with the mode-aware component model and the mode mapping mechanism. Second, the mode switch runtime mechanism must be included in each ProCom component for its mode switch handling. Furthermore, since component connections may change during a mode switch, ProCom must be able to provide multiple versions of component connections and switch between them when necessary. Next we shall present our approach in terms of the definition of multi-mode ProCom components, the mode switch handling in ProCom, and the support of varied component connections in different modes. To simplify the presentation, two assumptions are made: (1) the execution of a component in ProCom can be immediately aborted by a mode switch; (2) no new mode switch event is detected when a system is switching mode. The handling of atomic component execution which cannot be interrupted is presented in Chapter 5 of [7]. Without

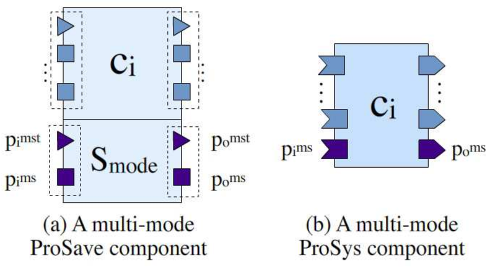


Figure 6: Multi-mode ProCom components

the second assumption, a conflict may occur due to multiple mode switch triggering. It is our ongoing work to provide handling of such conflicts.

4.1 Multi-mode ProCom components

Multi-mode components have not been considered by the current ProCom component model. However, we are able to define multi-mode ProCom components without extending ProCom. Since ProCom distinguishes ProSave and ProSys, multi-mode ProSave and ProSys components will be introduced separately in the following.

In ProSave, in order to separate the mode switch handling from the functional behavior of each component, a dedicated service S_{mode} is used for the mode switch handling of a multi-mode ProSave component. This service includes the definition of multiple modes, the configuration for each mode, mode mapping and the mode switch runtime mechanism. Furthermore, S_{mode} also has dedicated mode switch ports that correspond to p^{MSX} and p_{in}^{MSX} in the mode-aware component model. The service S_{mode} consists of an input port group and an output port group. The input port group comprises an input trigger port p_i^{mst} and an input data port p_i^{ms}, while the output port group comprises an output trigger port p_o^{mst} and an output data port p_o^{ms}. Figure 6(a) shows a typical multi-mode ProSave component c_i with two services, the lower service being S_{mode}. The dedicated mode switch ports of S_{mode} are highlighted in purple.

In ProSys, no concept of service exists and concurrent execution is allowed in a ProSys component, hence a dedicated internal thread can be used for the mode switch handling of a multi-mode ProSys component. Similar to ProSave, a multi-mode ProSys component should also have dedicated mode switch ports. Since a ProSys component is equipped with message ports that integrate both control flow and data flow, the dedicated mode switch ports of a multi-mode ProSys component can be assigned to an input message port p_i^{ms} and an output message port p_o^{ms}. Figure 6(b) shows a typical multi-mode ProSys component c_i whose dedicated mode switch ports are highlighted in purple.

4.2 The mode switch handling in ProCom

Section 3 states that the mode switch of a CBMMS is handled by the mode switch runtime mechanism of MSL. In this subsection, we integrate this mode switch runtime mechanism in ProCom. For primitive multi-mode ProCom components, such mechanism can be simply implemented in the code (complete algorithms described in pseudo code can be found in [9]). In this paper, our focus is on the mode switch handling of composite multi-mode ProCom components which requires a more elaborate approach in

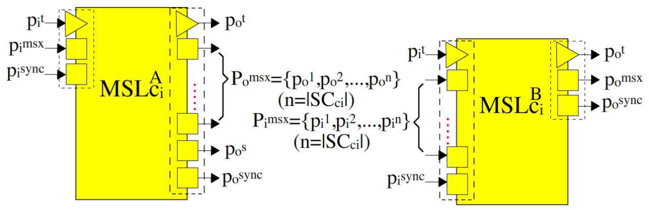


Figure 7: The port definition of $MSL_{c_i}^A$ and $MSL_{c_i}^B$

both ProSave and ProSys. Hereafter we by default imply multi-mode ProCom components while mentioning ProSave or ProSys components. The mode switch of a composite ProCom component is handled by dedicated subcomponents via its dedicated mode switch ports defined in Section 4.1.

4.2.1 The mode switch handling in ProSave

Since a composite ProSave component has no behavior and is just a composition of a set of enclosed ProSave components, a reasonable strategy is to introduce additional subcomponents that are dedicated to its mode switch handling. The same strategy can be applied to a ProSys component composed by ProSave components.

For a composite component c_i, which is either a composite ProSave component or a composite ProSys component composed by ProSave components, we introduce two primitive ProSave components: $MSL_{c_i}^A$ and $MSL_{c_i}^B$ as dedicated subcomponents of c_i for its mode switch handling. $MSL_{c_i}^A$ and $MSL_{c_i}^B$ jointly interact with the S_{mode} service of each subcomponent of c_i.

Let $c_i.p$ denote the port p of component c_i. Also, let $SC_{c_i} = \{c_j^1, c_j^2, \cdots, c_j^n\}$ $(n \in \mathbb{N})$ denote the set of subcomponents of c_i, excluding $MSL_{c_i}^A$ and $MSL_{c_i}^B$. Figure 7 illustrates the ports of $MSL_{c_i}^A$ and $MSL_{c_i}^B$, both of which are synchronized with each other via their synchronization ports p_i^{sync} and p_o^{sync}. Component $MSL_{c_i}^A$ has a single service with an input port group and an output port group. Apart from the synchronization ports, these port groups consist of the following ports:

- p_i^t: an input trigger port whose activation makes $MSL_{c_i}^A$ active.
- p_i^{msx}: an input data port for receiving a downstream primitive from the parent of c_i.
- p_o^t: an output trigger port activated after $MSL_{c_i}^A$ completes its current instance of execution.
- $P_o^{msx} = \{p_o^1, p_o^2, \cdots, p_o^n\}$ $(n = |SC_{c_i}|)$: a set of output data ports for sending a downstream primitive to SC_{c_i}.
- p_o^s: an output data port indicating the current mode of c_i.

Similarly, apart from the synchronization ports, $MSL_{c_i}^B$ also has the following ports:

- p_i^t: an input trigger port whose activation makes $MSL_{c_i}^B$ active.
- $P_i^{msx} = \{p_i^1, p_i^2, \cdots, p_i^n\}$ $(n = |SC_{c_i}|)$: a set of input data ports for receiving an upstream primitive from SC_{c_i}.

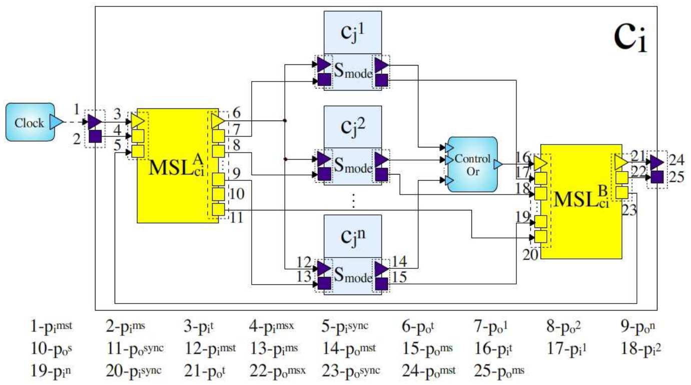

ci
cj1
cj2
cjn
Smode
Clock
MSLAci
MSLBci
Control Or
1-pimst 2-pims 3-pit 4-pimsx 5-pisync 6-pot 7-po1 8-po2 9-pon
10-pos 11-posync 12-pimst 13-pims 14-pomst 15-poms 16-pit 17-pi1 18-pi2
19-pin 20-pisync 21-pot 22-pomsx 23-posync 24-pomst 25-poms

Figure 8: The connections around $MSL_{c_i}^A$ and $MSL_{c_i}^B$

- p_o^t: an output trigger port activated after $MSL_{c_i}^B$ completes its current instance of execution.
- p_o^{msx}: an output data port for sending an upstream primitive to the parent of c_i.

The connections around $MSL_{c_i}^A$ and $MSL_{c_i}^B$ are illustrated in Figure 8. The ports associated with services other than S_{mode} of both c_i and SC_{c_i} have been omitted for simplicity. $MSL_{c_i}^A$ and $MSL_{c_i}^B$ are connected to both c_i and SC_{c_i}. Their connection with c_i is represented by the connection between $c_i.p_i^{ms}$ and $MSL_{c_i}^A.p_i^{msx}$ and the connection between $MSL_{c_i}^B.p_o^{msx}$ and $c_i.p_o^{ms}$. Their connection with SC_{c_i} is represented by the connection between $MSL_{c_i}^A.p_o^k$ ($k = [1, n]$) and $c_j^k.p_i^{ms}$ and the connection between $c_j^k.p_o^{ms}$ and $MSL_{c_i}^B.p_i^k$. A mode related control flow is established within c_i from $MSL_{c_i}^A$ to SC_{c_i} and then to $MSL_{c_i}^B$. A *Control Or* connector is used so that $MSL_{c_i}^B$ can be triggered by any subcomponent of c_i (i.e. c_i is able to receive an upstream primitive from any subcomponent). This connection pattern is repeated within all composite ProSave components. For instance, $\forall c_j^k \in SC_{c_i}$ ($k = [1, n]$) which is composite, the internal connections of c_j^k will exhibit the same connection pattern as c_i which enables the transmission of both downstream and upstream primitives. A downstream primitive from c_i to c_j^k can be transmitted from $MSL_{c_i}^A.p_o^k$ to $c_j^k.p_i^{ms}$ and c_j^k can propagate the primitive further to lower levels if it is composite and wants to. Conversely, an upstream primitive from c_j^k to c_i can be transmitted from $c_j^k.p_o^{ms}$ to $MSL_{c_i}^B.p_i^k$ and then $MSL_{c_i}^B$ will forward this primitive to $MSL_{c_i}^A$ via their synchronization ports. Let c_l be the parent of c_i, if c_i wants to propagate this primitive further to c_l, $MSL_{c_i}^B$ can send the primitive to $c_i.p_o^{ms}$ which must be externally connected to $MSL_{c_l}^B$ that is dedicated to the mode switch handling of c_l.

Attention must be paid to the connector *Clock* in Figure 8. Since ProSave components are passive and require external activation, a common *Clock* must be placed at the top ProSave level, periodically triggering the mode related control flow in ProSave. Since each ProSave component can only handle its mode switch when its S_{mode} service is active, the activation period of *Clock* highly affects the total mode switch time of a system.

Moreover, our initial intention was to use a single dedicated component to handle the mode switch of a composite ProSave component. The reason why two such components are used is attributed to the rigorous execution semantics in ProSave, which prohibits mutual triggering between two neighboring ProSave components. If a single component, say MSL_{c_i}, is used instead of $MSL_{c_i}^A$ and $MSL_{c_i}^B$, there must exist mutual triggering between MSL_{c_i} and SC_{c_i}. Consequently, the execution semantics of ProCom will be violated.

Since both $MSL_{c_i}^A$ and $MSL_{c_i}^B$ are primitive ProSave components, they can be easily implemented by following the mode mapping mechanism and mode switch runtime mechanism of MSL (see the algorithms provided in [9] which can also be automatically generated together with the structure of $MSL_{c_i}^A$ and $MSL_{c_i}^B$). In general, $MSL_{c_i}^A$ is responsible for handling a downstream primitive while $MSL_{c_i}^B$ is responsible of handling an upstream primitive. Besides, if c_i is able to initiate a mode switch by detecting a mode switch event, a primitive will be issued from either $MSL_{c_i}^A$ or $MSL_{c_i}^B$ depending on its direction.

4.2.2 The mode switch handling in ProSys

The mode switch handling in ProSys is similar to that in ProSave. For a composite ProSys component c_i, we introduce a dedicated subcomponent of c_i for its mode switch handling: MSL_{c_i} which plays an equal role as the pair of $MSL_{c_i}^A$ and $MSL_{c_i}^B$. However, message passing between ProSys components is more flexible than the pipe-and-filter communication in ProSave. Two ProSys components can send messages to each other, therefore, a single subcomponent MSL_{c_i} is sufficient for the mode switch handling of c_i.

Still, let $SC_{c_i} = \{c_j^1, c_j^2, \cdots, c_j^n\}$ ($n \in \mathbb{N}, n = |SC_{c_i}|$) denote the set of subcomponents of c_i, excluding MSL_{c_i}. Figure 9 illustrates the ports of MSL_{c_i}:

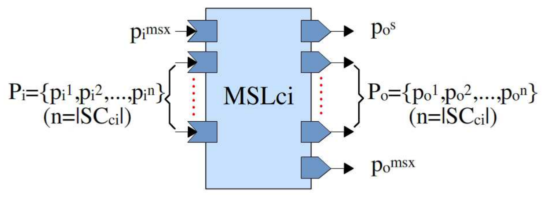


Figure 9: The port definition of MSL_{c_i}

- p_i^{msx}: an input message port for receiving a downstream primitive from the parent of c_i.
- $P_i = \{p_i^1, p_i^2, \cdots, p_i^n\}$: a set of input message ports for receiving an upstream primitive from SC_{c_i}.
- p_o^s: an output message port indicating the current mode of c_i.
- $P_o = \{p_o^1, p_o^2, \cdots, p_o^n\}$: a set of output message ports for sending a downstream primitive to SC_{c_i}.
- p_o^{msx}: an output message port for sending an upstream primitive to the parent of c_i.

The connections around MSL_{c_i} are illustrated in Figure 10, where the ports not related to the mode switches of both c_i and SC_{c_i} have been omitted for simplicity. Dark red shapes are message channels. Component MSL_{c_i} has direct communication with both c_i and SC_{c_i}. On the one hand,

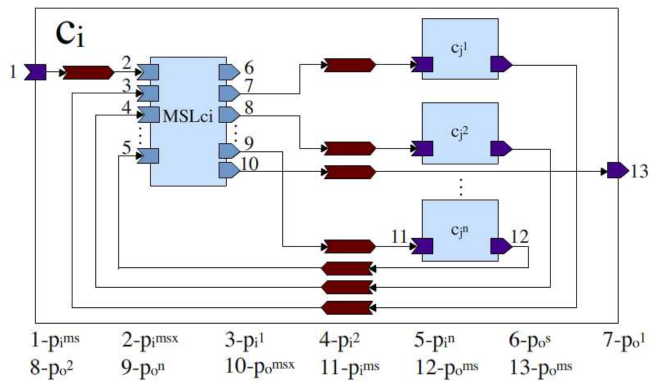

ci
MSLci
cj1
cj2
cjn
1
2
3
4
5
6
7
8
9
10
11
12
13
1-pims 2-pimsx 3-pi1 4-pi2 5-pin 6-pos 7-po1
8-po2 9-pon 10-pomsx 11-pims 12-poms 13-poms

Figure 10: The connections around MSL_{c_i}

$MSL_{c_i}.p_i^{msx}$ is connected to $c_i.p_i^{ms}$ and $MSL_{c_i}.p_o^{msx}$ is connected to $c_i.p_o^{ms}$. On the other hand, $MSL_{c_i}.p_o^k$ ($k = [1, n]$) is connected to $c_j^k.p_i^{ms}$ and $c_j^k.p_o^{ms}$ is connected to $MSL_{c_i}.p_i^k$. No *Clock* is needed in ProSys, because ProSys components are active and can execute without external activation. Additionally, since a message channel allows many-to-many communication, the *Control Or* connector in ProSave is removed. This connection pattern is repeated for all composite ProSys components while enabling the transmission of both downstream and upstream primitives.

MSL_{c_i} is a primitive ProSys component where the mode switch runtime mechanism of c_i is implemented and described in the algorithms provided in [9].

4.3 Managing the variability of ProCom component connections in multiple modes

Section 4.2 explains the mode switch handling of a ProCom component, yet without addressing how component reconfiguration is achieved during a mode switch in ProCom. Many properties of a component can be changed by reconfiguration, e.g. functional behavior and running status (activated or deactivated). Among these properties, our focus in this paper is on the inner component connections of a composite ProCom component. As indicated in Figure 1 at the beginning of this paper, the inner component connections of a composite component c_i can be different while c_i is in different modes. The inner component connections of c_i for each mode can be separately defined at design time and changed to each other during a mode switch at runtime. In order to manage the variability of component connections in different modes in ProCom, we provide a solution which can automatically generate a complete view of inner component connections of each composite ProCom component based on its inner component connections separately defined for each mode. Depending on the current mode of a composite component, the activated subcomponents and corresponding inner component connections are selected.

4.3.1 Managing the variability of component connections in ProSave

Consider a composite ProSave component c_i, whose inner component connections are mode-dependent. The basic idea of managing the variability of inner component connections of c_i is to package each $c_j^k \in SC_{c_i}$ ($k = [1, n], n = |SC_{c_i}|$) with additional connectors. Each connector integrates all the possible incoming or outgoing connections of a specific port for all modes and can select the correct connection based on the current mode of c_i.

Let $M_{c_i} = \{m_{c_i}^1, m_{c_i}^2, \cdots, m_{c_i}^q\} (q > 1)$ be the set of supported modes of c_i. Suppose the inner component connections of c_i for each mode $m_{c_i}^l$ ($l = [1, q]$) have been well-defined. Besides, for a ProSave component c, the following sets of ports are defined:

- P_i^t: the set of input trigger ports excluding $c.p_i^{mst}$.
- P_i^d: the set of input data ports excluding $c.p_i^{ms}$.
- P_o^t: the set of output trigger ports excluding $c.p_o^{mst}$.
- P_o^d: the set of output data ports excluding $c.p_o^{ms}$.

In order to merge the inner component connections of c_i into a complete view, connectors are automatically generated within c_i based on the following rules:

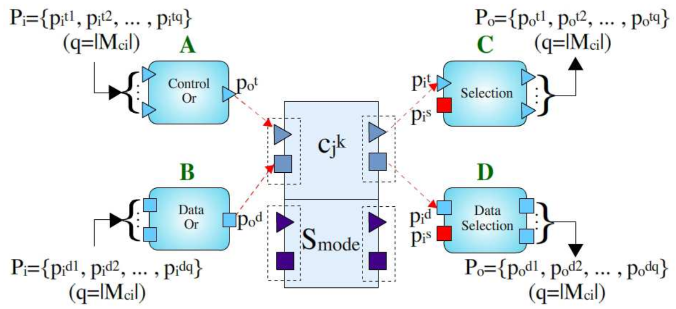
- For each p where $p \in c_j^k.P_i^t$ or $p \in c_i.P_o^t$, a *Control Or* connector A is generated, with a set of input trigger ports $P_i = \{p_i^{t1}, p_i^{t2}, \cdots, p_i^{tq}\}(q = |M_{c_i}|)$ and an output trigger port p_o^t. The incoming connection to $A.p_i^{tl}$ ($l = [1, q]$) follows the pre-defined connection while c_i is in mode $m_{c_i}^l$. The output trigger port $A.p_o^t$ is directly connected to p.

- For each p where $p \in c_j^k.P_i^d$ or $p \in c_i.P_o^d$, a *Data Or* connector B is generated, with a set of input data ports $P_i = \{p_i^{d1}, p_i^{d2}, \cdots, p_i^{dq}\}(q = |M_{c_i}|)$ and an output data port p_o^d. The incoming connection to $B.p_i^{dl}$ ($l = [1, q]$) follows the pre-defined connection while c_i is in mode $m_{c_i}^l$. The output data port $B.p_o^d$ is directly connected to p.

- For each p where $p \in c_j^k.P_o^t$ or $p \in c_i.P_i^t$, a *Selection* connector C is generated, with an input trigger port p_i^t, an input data port p_i^s and a set of output trigger ports $P_o = \{p_o^{t1}, p_o^{t2}, \cdots, p_o^{tq}\}(q = |M_{c_i}|)$. The input trigger port $C.p_i^t$ is directly connected to p. The input data port $C.p_i^s$ is connected to $MSL_{c_i}^A.p_o^s$ (see Section 4.2). The outgoing connection from $C.p_o^{tl}$ ($l = [1, q]$) follows the pre-defined connection while c_i is in mode $m_{c_i}^l$ according to the data from $C.p_i^s$: If the data returns $m_{c_i}^l$ ($l = [1, q]$), $C.p_o^{tl}$ will be triggered.

- For each p where $p \in c_j^k.P_o^d$ or $p \in c_i.P_i^d$, a *Data Selection* connector D is generated, with an input data port p_i^d, and another input data port p_i^s and a set of output data ports $P_o = \{p_o^{d1}, p_o^{d2}, \cdots, p_o^{dq}\}(q = |M_{c_i}|)$. The input data port $D.p_i^d$ is directly connected to p. The input data port $D.p_i^s$ is connected to $MSL_{c_i}^A.p_o^s$. The outgoing connection from $D.p_o^{dl}$ ($l = [1, q]$) follows the pre-defined connection while c_i is in mode $m_{c_i}^l$ according to the data from $D.p_i^s$: If the data returns $m_{c_i}^l$ ($l = [1, q]$), the data from $D.p_i^d$ will be forwarded exactly to $D.p_o^{dl}$.

The rules above also apply to a ProSys component composed by ProSave components by considering each message port as a port group consisting of a trigger port and a data port. Among these four generated connectors, *Data Selection* does not exist in the current ProCom component model.



Pi={pit1, pit2, ... , pitq}
(q=|Mci|)
A
Control Or
pot
C
Po={pot1, pot2, ... , potq}
(q=|Mci|)
pit
pis
Selection
cjk
B
Data Or
pod
D
pid
pis
Data Selection
Smode
Pi={pid1, pid2, ... , pidq}
(q=|Mci|)
Po={pod1, pod2, ... , podq}
(q=|Mci|)

Figure 11: Managing the variability of ProSave component connections

Nonetheless, it can be easily developed as its execution semantics is fairly similar to *Selection*. This is the only extension of ProCom required by our approach. The above presented rules are illustrated in Figure 11 where generated connectors are externally connected to $c_j^k \in SC_{c_i}$. Additional connectors internally connected to c_i can also be generated accordingly (further illustrated in Section 5).

4.3.2 *Managing the variability of component connections in ProSys*

In comparison with ProSave, the central idea of managing the variability of ProSys component connections is rather similar in that all the generated connectors in ProSave can be replaced with primitive ProSys components. However, since an input message port can receive messages from multiple senders, there is no need to generate ProSys components playing the role of *Control Or* or *Data Or*. Hence, the only ProSys component that needs to be generated is a *Selection* ProSys component which functions as both connectors *Selection* and *Data Selection*.

Let c_i be a composite ProSys component composed by ProSys components, with the set of supported modes $M_{c_i} = \{m_{c_i}^1, m_{c_i}^2, \cdots, m_{c_i}^q\}(q > 1)$ and the set of subcomponents $SC_{c_i} = \{c_j^1, c_j^2, \cdots, c_j^n\}$ $(n = |SC_{c_i}|)$. Suppose the inner component connections of c_i for each mode $m_{c_i}^l$ $(l = [1, q])$ have been well-defined. For a ProSys component c, let P_i be the set of input message ports excluding $c.p_i^{ms}$ and let P_o be the set of output message ports excluding $c.p_o^{ms}$. Then for each p where $p \in c_j^k.P_o$ $(k = [1, n])$ or $p \in c_i.P_i$, a primitive ProSys component, called *Selection* and denoted as E, is generated, with two input message ports p_i and p_s and a set of output message ports $P = \{p_o^1, p_o^2, \cdots, p_o^q\}$ $(q = |M_{c_i}|)$. The port $E.p_i$ is directly connected to p while $E.p_s$ is connected to $MSL_{c_i}.p_o^s$ (see Section 4.2). The outgoing connection from $E.p_o^l$ $(l = [1, q])$ follows the pre-defined connection while c_i is in $m_{c_i}^l$ according to the data from $E.p_s$: If the data returns $m_{c_i}^l$ $(l = [1, q])$, the message sent to $E.p_i$ will be forwarded to $E.p_o^l$. Figure 12 illustrates $c_j^k \in SC_{c_i}$ externally connected to the generated *Selection* component. Additionally, a *Selection* component can also be generated and internally connected to c_i (further illustrated in Section 5).

5. AN EXAMPLE

Section 4 has presented our principal ideas of implementing MSL in the ProCom component model. In this section, an example is used to illustrate this, covering all the key el-

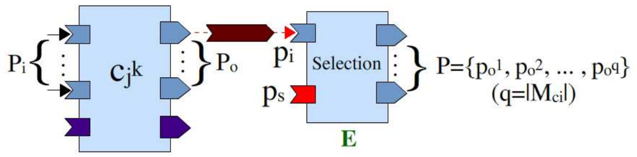


Figure 12: Managing the variability of ProSys component connections

ements in Section 4 and demonstrating how a CBMMS can be developed in ProCom under the guidance of MSL.

5.1 System description

Consider a system to be developed in ProCom, with its component hierarchy given in Figure 1. Components d and e are ProSave components while the others are ProSys components. Components *Top* and b are composite and their basic mode mappings are given in tables 1 and 2 where the modes of different components belonging to the same column are mapped. For instance, when *Top* is running in m_{Top}^1, b is running in either m_b^1 or m_b^3, and c is deactivated.

Component	Supported modes		
Top	m_{Top}^1		m_{Top}^2
a	m_a^1		m_a^2
b	m_b^1	m_b^3	m_b^2
c	Deactivated		m_c^1

Table 1: The basic mode mapping of *Top*

Component	Supported modes			
b	m_b^1	m_b^2		m_b^3
d	m_d^1	m_d^2	m_d^3	Deactivated
e	m_e^1			

Table 2: The basic mode mapping of *b*

It should be pointed out that tables 1 and 2 are insufficient for defining the full range of possible mode mappings of *Top* and b. Mode Mapping Automata (MMA) [7] can be used to define more mappings. It is our future work to also integrate MMA into ProCom for better mode mapping specification.

Component b is a ProSys component composed by ProSave components. The inner component connections of b in different modes, illustrated in Figure 13(a), are treated in ProSave where control flow and data flow are separate. In contrast, Figure 13(b) illustrates the inner component connections of *Top* in m_{Top}^1 and m_{Top}^2 in ProSys.

In order to develop such a CBMMS in ProCom, the first step is to define the multi-mode ProSave and ProSys components introduced in Section 4.1. Figure 14 shows the hierarchy of all multi-mode ProCom components. The dedicated mode switch ports of each component are marked in purple. Furthermore, as multi-mode ProSave components, d and e also have a dedicated service S_{mode}.

5.2 The mode switch handling

In this example, a, c, d and e are primitive components, whose mode switch handling can be directly implemented

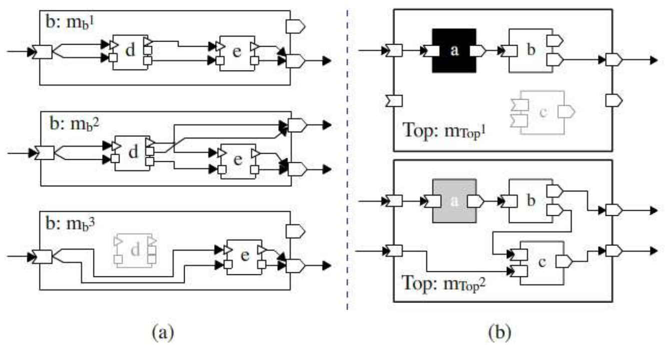


Figure 13: The inner component connections of *b* and *Top* in different modes

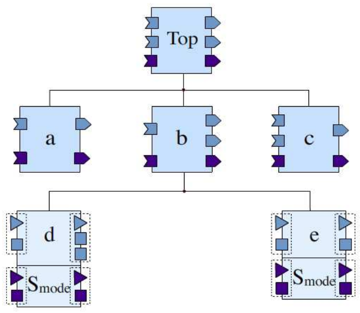


Figure 14: The component hierarchy in ProCom

by source code. In contrast, for composite components *Top* and *b*, additional subcomponents must be used to handle their mode switches.

Since Component *b* is composed by ProSave components, its mode switch can be handled by a pair of dedicated subcomponents MSL_b^A and MSL_b^B, both of which can be automatically generated, given the mode mapping of *b*. The ports of MSL_b^A and MSL_b^B and the connections around them are presented in Figure 15. Please note that $MSL_b^A.P_o^{msx} = \{MSL_b^A.p_o^d, MSL_b^A.p_o^e\}$ and $MSL_b^B.P_i^{msx} = \{MSL_b^B.p_i^d, MSL_b^B.p_i^e\}$, as d and e are the subcomponents of *b*. Components MSL_b^A and MSL_b^B jointly handle the mode switch of *b* and their internal behaviors are described by algorithms provided in [9].

Component *Top* is a ProSys component composed by ProSys components, thus its mode switch can be handled by a single dedicated subcomponent MSL_{Top} that can be automatically generated, given the mode mapping of *Top*. The ports of MSL_{Top} and its incoming and outgoing connections are presented in Figure 16. Since the subcomponents of *Top* are a, b and c, for MSL_{Top}, $P_i^{msx} = \{p_i^a, p_i^b, p_i^c\}$ and $P_o^{msx} = \{p_o^a, p_o^b, p_o^c\}$. Component MSL_{Top} handles the mode switch of *Top*, and its internal behavior is described by algorithms provided in [9].

5.3 Managing the variability of component connections

Figure 13 indicates the variability of inner component connections of both *b* and *Top*. In order to manage such variability, we can generate a complete view by adhering to the principles introduced in Section 4.3.

A complete view of the inner component connections of *b* is presented in Figure 15, automatically generated based on Figure 13(a) and including all the additional connectors introduced in Section 4.3, i.e. *Control Or*, *Data Or*, *Selection* and *Data Selection*. All these connectors have three input or output ports because *b* can run in three modes: m_b^1, m_b^2 and m_b^3. Each *Selection* or *Data Selection* has an input data port marked in red. This port is connected to $MSL_b^A.p_o^s$ which tells the current mode of *b* so that the correct outgoing connection is selected. Moreover, a *Clock*, MSL_b^A and MSL_b^B, and a *Control Or* connector connected to $MSL_b^B.p_i^t$ are also generated for handling the mode switch of *b*.

A complete view of the inner component connections of *Top* is presented in Figure 16, automatically generated based on the inner component connections of *Top* separately defined for each mode (see Figure 13(b)). The complete view includes MSL_{Top} for the mode switch handling of *Top* and a couple of *Selection* ProSys components defined in Section 4.3. Each *Selection* component has two output message ports because *Top* can run in two modes: m_{Top}^1 and m_{Top}^2. Meanwhile, each *Selection* component also has a particular input message port marked in red. This port is connected to $MSL_{Top}.p_o^s$ which tells the current mode of *Top* such that the correct outgoing connection is selected.

Please note that the generated complete views of component connections in figures 15 and 16 are not optimized yet. By default, the generation rules assume that any connection associated with any port not dedicated to mode switch is different for different modes. However, some connections remain the same for all modes. For instance, according to Figure 13(a), the outgoing connection of *e* is never changed regardless of the current mode of *b*. Then the four generated connectors between *e* and the second output port of *b* in Figure 15 can actually be removed. Such optimization can be employed at both the ProSave and ProSys layers, thus substantially simplifying the generated complete view.

6. RELATED WORK

Apart from ProCom, many other component models have been proposed for the development of embedded systems, such as SaveCCM [11] (the predecessor of ProCom), COMDES-II [14] and MyCCM-HI [2], to name a few. There are also some other component models which have been commercialized, e.g. Koala [15] (targeting consumer electronics) and Rubus [10] (targeting ground vehicles). These component models have different notions about the mode switch handling. For instance, in Koala and SaveCCM, a special *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, mode switch is also addressed by languages such as the Architecture Analysis & Design Language [6], where 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

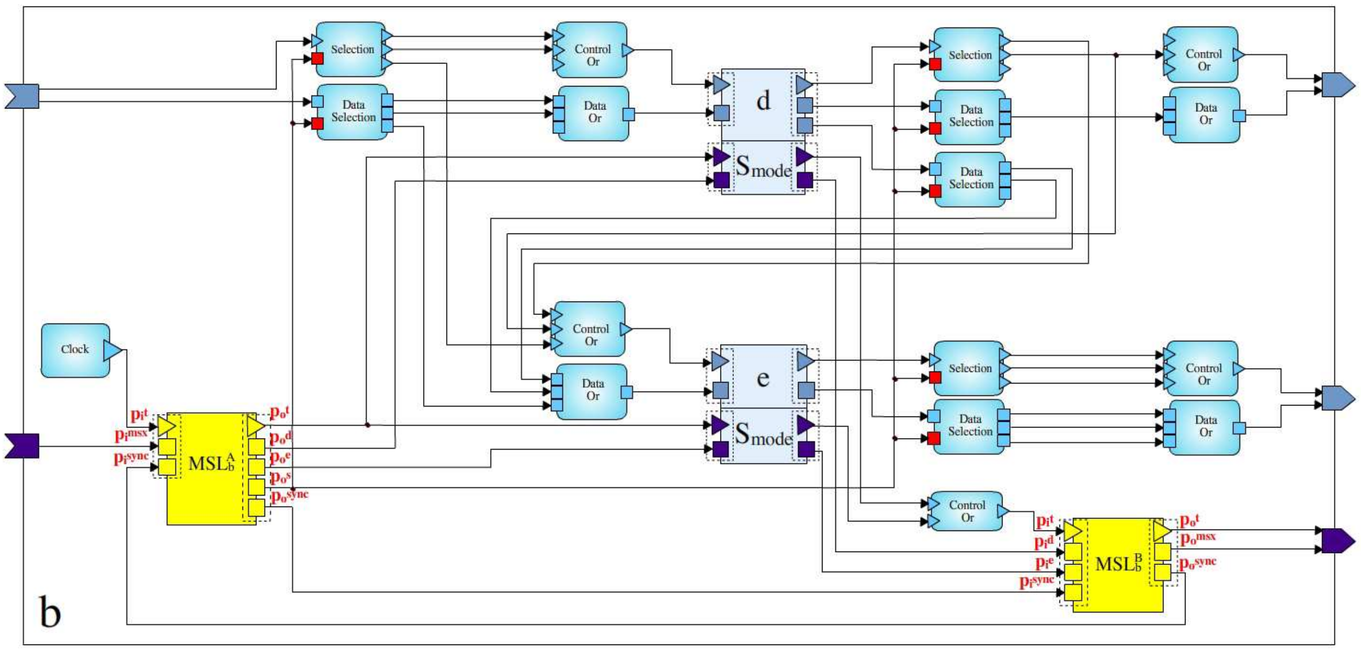


Figure 15: The complete view of inner component connections of *b*

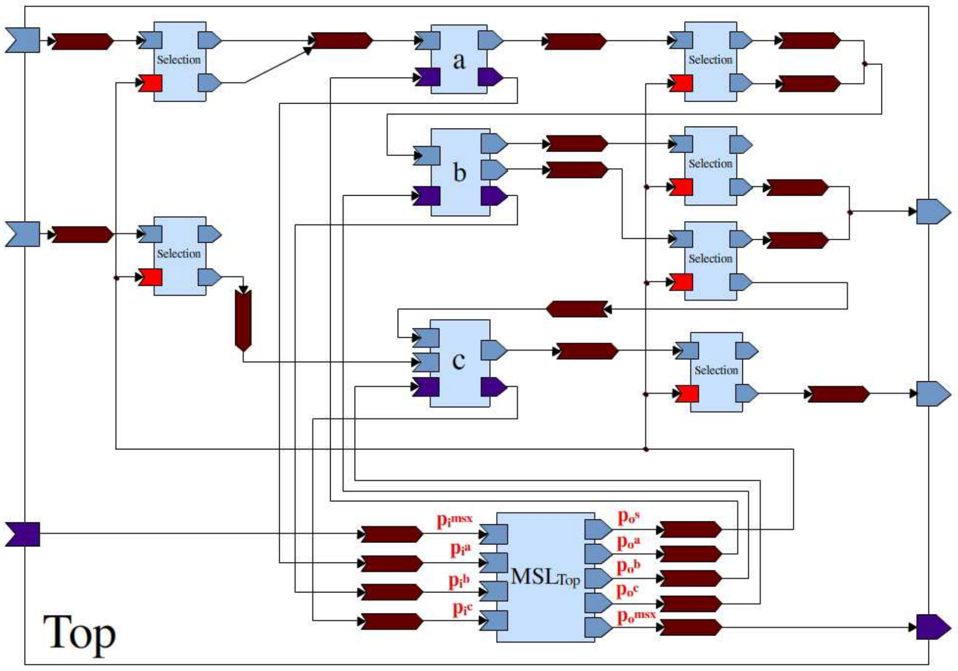


Figure 16: The complete view of inner component connections of *Top*

these states (mode switches) and input/output event ports used for mode switch triggering.

To the best of our knowledge, the extended MechatronicUML (EUML) [12] by Heinzemann et al. is currently the most closely related work to our MSL. However, EUML focuses more on component reconfiguration while mode is not addressed, hence it does not consider mode-related issues such as mode mapping. In general, MSL is relatively more mature since EUML is more recently developed. However, some initial ideas of EUML coincidentally resemble those in MSL. For instance, EUML suggests that component reconfiguration is not only locally performed but also propagated through the component hierarchy. This is similar to the MSP protocol of MSL. In EUML, the reconfiguration of a composite component is handled by two dedicated subcomponents: a *Manager* and an *Executor*, which play similar roles as the dedicated subcomponents of a composite ProCom component here.

Another recent work related to MSL is the oracle-based approach [16] by Pop et al. The basic idea is to abstract component behaviors into a property network spread throughout the component hierarchy. The mode of each component is modeled as a property and mapped from a set of properties to their valuations. A single property change can be propagated throughout the property network, potentially leading to the valuation change of other properties. And then the new mode of each component can be derived after the update of the property network. A finite-state machine called Oracle is offline constructed to guarantee predictable update time of the property network. The construction of Oracle implies that the mode switch handling requires global information of the property network. In contrast, MSL is fully distributed, requiring no global information.

7. CONCLUSION AND FUTURE WORK

This paper presents an approach to the mode switch handling of the ProCom component model guided by the Mode Switch Logic (MSL). It is shown that the mode switch of a Component-Based Multi-Mode System (CBMMS) can be properly handled after a slight extension of ProCom (i.e. the introduction of the *Data Selection* connector). Multi-mode ProSave and ProSys components are defined with reference to the mode-aware component model of MSL. Also, it is suggested that additional subcomponents can be used to handle the mode switch of each composite ProCom component. In order to manage the variability of component connections of a CBMMS, component connections in all modes are merged into a complete view with auxiliary elements generated at both the ProSave and ProSys layers. Thereby, each composite component is able to select the corresponding inner connections based on its current running mode. Finally, our approach is demonstrated by a simple example.

As future work, the theories presented in this paper will be refined and implemented in PRIDE [1], a developing environment based on the ProCom component model. It is also envisioned that the extended PRIDE will provide a practical platform for the evaluation of both MSL and our implementation of MSL in ProCom.

8. ACKNOWLEDGMENTS

This work is supported by the Swedish Research Council via the ARROWS project at Mälardalen University.

9. REFERENCES

[1] PRIDE. `http://www.idt.mdh.se/pride/?id=home`.

[2] E. Borde, G. Haïk, and L. Pautet. Mode-based reconfiguration of critical software component architectures. In *DATE*, 2009.

[3] T. Bureš, J. Carlson, I. Crnković, S. Sentilles, and A. Vulgarakis. ProCom - the Progress component model reference manual, version 1.0. Technical Report ISSN 1404-3041 ISRN MDH-MRTC-230/2008-1-SE, Mälardalen University, 2008.

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

[5] I. Crnković, S. Sentilles, A. Vulgarakis, and M. R. V. Chaudron. A classification framework for software component models. *IEEE Transactions on Software Engineering*, 37(5), 2011.

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

[7] Y. Hang. *Mode switch for component-based multi-mode systems.* Licentiate thesis, Mälardalen University, Västerås, Sweden, December 2012.

[8] Y. Hang, J. Carlson, and H. Hansson. Towards mode switch handling in component-based multi-mode systems. In *CBSE*, 2012.

[9] Y. Hang, H. Qin, J. Carlson, and H. Hansson. Mode switch handling for the ProCom component model. Technical Report ISSN 1404-3041 ISRN MDH-MRTC-271/2013-1-SE, MRTC, Mälardalen University, Jan 2013.

[10] K. Hänninen, J. Mäki-Turja, M. Nolin, M. Lindberg, J. Lundbäck, and K. Lundbäck. The Rubus component model for resource constrained real-time systems. In *SIES*, 2008.

[11] H. Hansson, M. Åkerholm, I. Crnković, and M. Törngren. SaveCCM - a component model for safety-critical real-time systems. In *Proceedings of Euromicro Conference, Special Session on Component Models for Dependable Systems*, 2004.

[12] C. Heinzemann, C. Priesterjahn, and S. Becker. Towards modeling reconfiguration in hierarchical component architectures. In *CBSE*, 2012.

[13] P. Hošek, T. Pop, T. Bureš, P. Hnětynka, and M. Malohlava. Comparison of component frameworks for real-time embedded systems. In *Component-Based Software Engineering*, volume 6092 of *Lecture Notes in Computer Science*. 2010.

[14] X. Ke, K. Sierszecki, and C. Angelov. COMDES-II: A component-based framework for generative development of distributed real-time control systems. In *RTCSA*, 2007.

[15] R. V. Ommering, F. V. D. Linden, J. Kramer, and J. Magee. The Koala component model for consumer electronics software. *Computer*, 33(3), 2000.

[16] T. Pop, F. Plasil, M. Outly, M. Malohlava, and T. Bureš. Property networks allowing oracle-based mode-change propagation in hierarchical components. In *CBSE*, 2012.