Feasibility of migrating analysis and synthesis mechanisms from ProCom to IEC 61499

Jan Carlson and Luka Lednicki
Mälardalen University, Västerås, Sweden

Technical report
MDH-MRTC-268/2012-1-SE

June 4, 2012

Contents

1 Introduction and background

This report presents the initial results from the ASSIST project at Mälardalen University, funded by the ABB Software Research Grant Program. The project aims to bridge the gap between recent academic research achievements in the area of control- and model based development of embedded systems, and concrete industrial needs and state of practice in this domain. Concretely, the focus of the project is to investigate how novel timing analysis and code synthesis techniques developed in the context of ProCom, an academic component model for embedded systems, can be extended and adapted in order to be applicable to IEC 61499.

The report is organized as follows: The two languages are presented in Sections 2 and 3, respectively, followed in Section 4 by a comparison highlighting a number of important similarities and differences that are particularly relevant with respect to the project scope. Section 5 discusses the possibilities and difficulties involved in migrating different key aspects of the analysis and synthesis mechanisms defined for ProCom to an IEC 61499 context.

2 ProCom

ProCom is a component model specifically developed to address the particularities of the embedded systems domain, including resource limitations and requirements on safety and timeliness. It was developed in the context of PROGRESS, a project at Mälardalen University funded by the Swedish Foundation for Strategic Research.

The development of ProCom is a continuation of previous work at Mälardalen University developing the component model SaveCCM [1], and covers aspects such as formal methods, static code analysis, management of extra-functional properties, deplyoment and code synthesis. There is also a development environment, PRIDE [5], which integrates editors and tools to support many of the addressed aspects.

This section presents an overview of the ProCom modeling concepts and key features. For an in-depth description of ProCom, the reader is referred to the reference manual [6] and the formal definition of the semantics [24]. There are also publications presenting the overall idea and key concerns of ProCom [7, 20].

2.1 The Component Model

ProCom is organized in two distinct layers, addressing the different concerns on different levels of granularity. The layers differ in terms of architectural style and communication paradigm. The lower layer, *ProSave* consists of smaller, passive components, and is based on a pipes-and-filters interaction style with an explicit separation between data and control flow. In the top layer, called *ProSys* a system is modelled as a collection of active, interconnected components that execute concurrently and communicate by asynchronous message passing.

Both layers are hierarchical, i.e., supporting *composite* components, internally defined by interconnected subcomponents. The way in which the two layers are linked together is that a *primitive* ProSys component can be modelled as a collection of ProSave subcomponents and clocks defining represent periodic activations. At the bottom of the hierarchical nesting, the primitive ProCom components are implemented by C functions.

From the perspective of this report, ProSave is the more interesting layer, since it more closely relates to the concepts in IEC 61499, and thus we will not describe the ProSys layer further.

2.1.1 Components

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

Figure 1 shows the graphical representation of a relatively simple ProSave component with one input trigger group (left side) and two output groups (right side). Trigger- and data ports are represented by triangles and squares, respectively.

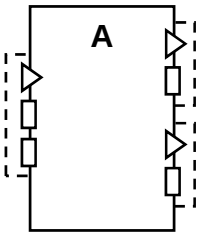

Figure 1: Example of a ProSave component.

The execution semantics of ProSave components is fairly restricted and follows a strict read-execute-write cycle. Initially, a component is in an idle state, just receiving data on its input data ports. When one of the trigger ports receive an activation, execution the following steps: First, the data at the input data ports of the trigger port are atomically copied to internal representations which remain unchanged until the end of the service execution, and then the functionality associated with the trigger port is executed. During the execution, each output port group associated with the activated trigger port is triggered once, meaning that the values of the data ports of the group are made available externally and that the triggering port of the group is activated. When all output port groups have been triggered, the the excution stops and the input trigger port is cleared to recieve a new activation.

ProCom permits concurrent (interleaved) execution of components. However, since data is only read at the beginning of the execution, any data produced by a component executing concurrently and arriving during the execution does not affect the component until next invocation.

It is also worth pointing out that the execution semantics is the same for primitive and composite components. For primitive components, execution corresponds to calling the C function associated with the triggered port, while the functionality of a composite component is defined by an internal collection of interconnected subcomponents, as described in the next section. The fact that different component types follow the same semantics means that components can be treated as black boxes when needed. For example, ProCom supports explicit modeling of *unimplemented* components, i.e., components with defined interface but without implementation. Later in the development, an unimplemented component can be changed into either primitive or composite components, and further elaborated.

2.1.2 Composite Components

A composite components internally consists of interconnected subcomponents that, together, define the functionality of the composite. In addition to simple connections between ports (of two subcomponent or one subcomponent and the enclosing composite), ProCom include a set of *connectors* to provide detailed control over the data- and control flow. These connectors include fork and join constructs, and a selection construct to define data dependent control flows.

As an example, assuming that the component in Figure 1 is a composite component, a possible internal structure is shown in Figure 2. The filled circle denotes a *control fork* connector.

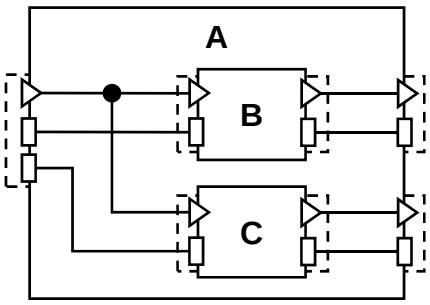

Figure 2: Example of a composite ProSave component.

2.2 Analysis

ProCom analysis is supported by two general and extensible frameworks. The *attribute framework* [19] handles the management of extra-functional properties,

including mechanisms to define new attribute types and the associated meta-data, and for associating new attribute values with different entities of a model. There is also an *analysis framework* which facilitates integration of different analysis tools by providing a common interface to invoke analysis and to access and store analysis results. These frameworks facilitate the introduction of new analysis techniques, but more importantly they provide means to easily decorate a component with additional information (provided by the developer or derived by analysis or measurements) to be reused when the component is reused in a new system.

Based on these frameworks, a number of concrete analysis techniques have been developed in the ProCom context. Some relatively simple methods have been developed to verify different aspects of ProCom models, for example type consistency of connected ports, or conformance of composite components to the component semantics. There are also a couple of more complex analysis mechanisms, as described below.

A *compositional model-level analysis of worst case execution time* (WCET) has been developed, covering both ProSave and ProSys and the connection between the two layers [8]. Given a composite component, and WCET information for the subcomponents, the analysis derives WCET information for the composite. For ProSave components, the WCET information is relatively simple, and consists of a single WCET value per input triggering port. The ProSys components, however, need more elaborate information since the can be active and since there is no explicit link between input and output.

There is also an alternative timing analysis method that can handle parametric WCET information [16], i.e., where the WCET of a single component is represented by a function over input values rather than by a single integer. In the current form, however, this analysis can only address a single hierarchical level and cannot be applied recursively since the analysis input and output information differ.

The *error propagation* analysis originally developed for SaveCCM [2,12], has been adjusted for ProCom components. It allows the developer to specify for individual components how errors at the input ports (value, late, omission, etc.) can lead to errors manifesting at the output ports. From this, and the system structure, the analysis determines how errors propagate through the system.

Complex behavioural modeling, including functionality, timing and resource usage and possible dependencies between these aspects, is supported by mean of REMES [18], a high-level language for behavioural modeling. Individual components can be decorated with REMES models specifying for example consumption and sharing of resources, allowing the overall system behaviour to be analysed by model checking techniques.

2.3 Synthesis

ProCom components are design time entities, and during the later stages of development the final system is generated from the complete system model in order to achieve the desired runtime efficiency. The final result is a collection

of tasks to be executed under a standard real-time operating system (currently FreeRTOS).

In addition to the model defining the architecture of the application software, in terms of interconnected components, the hardware topology is specified, as well as the allocation of components to hardware nodes. This allocation is done in two steps: Components are allocated to *virtual nodes*, concrete reusable units that provide a degree of temporal isolation. The virtual nodes, in turn, are allocated to the physical nodes of the system [9].

The ProCom code synthesis is not a single monolithic step performed at the very end of development. Rather, it is possible to synthesise individual components on all levels of the hierarchy separately, which allows the developer to trade-off between reusability and efficiency for different parts of the system. When a component is synthesised in isolation, no assumptions are made about the context. For example, since input can be produced by a component executing concurrently, a mechanism of locks and double buffering is used to ensure conformance to the semantics. The approach allows for some adjustment of these mechanisms once a pre-synthesised component is used in a particular context, such as removing the locks when the producer of the input is in the same composite component, and thus cannot execute concurrently [3] [4].

For maximum runtime efficiency, it is also possible to synthesise multiple hierarchical levels together, without producing reuseable code for the intermediate entities. This *flattening* approach allows for additional optimizations that are not possible by interface adjustments. Thus, the developer can decide what parts of the system hierarchy to generate as reusable units, and where full optimization is required [17].

3 IEC 61499

The IEC 61499 standard [25, 26] has been developed to accommodate development of industrial automation systems. It is proposed as a successor of the IEC 61131-3 standard widely used in industry. The new standard addresses high level requirements of new automation systems, such as portability, configurability, interoperability, reconfiguration and distribution of both devices and system intelligence.

Since IEC 61499 was first published in 2005, several semantic weaknesses have been detected. These ambiguities in some parts of the standard definition have resulted in different implementations of the standard [22, 23]. For the purpose of this report we have decided not to cover all implementation alternatives, but focus on one of the industrial implementation. As our target implementation we have chosen the open-source 4DIAC engineering tool and FORTE runtime environment [21].

3.1 Architectural Elements and their Semantics

The main architectural element of IEC 61499 is the Function Block. Function blocks are reusable units of software that implement a specific functionality with a clear separation between interface and implementation. To create complex functionality, function blocks can be connected into *Function Block Networks* (FBNs). Considering implementation, a function block can be of three possible types: Basic function block (BFB), Service interface function block (SIFB) and Composite function block (CFB). Below, a description of each of these elements is given.

3.1.1 Function Block Interface

The interface defines how a function block presents its functionality to the rest of the system. Although all block types share the same interface syntax, execution semantics is not uniquely defined at the interface level.

The interface explicitly separates event and data inputs and outputs. Event inputs and outputs are used to specify the execution flow of the system, but do not provide any means for exchanging data between function blocks. All data transfers are done by data inputs and outputs. A part of the component interface definition are also WITH qualifiers. A WITH qualifier connects one input or output event port with one or more data ports of the same direction. Such a connection denotes that consumption of an input event will also consume data from the associated ports, or that output of an output event will coincide with data being produced at the associated port.

Figure 3 shows an example of a function block with two input event ports (e_1 and e_2) three input data ports (d_1, d_2 and d_3), two output event ports (e_3 and e_4) and two output data ports (d_4 and d_5). The connected small squares denote WITH qualifiers.

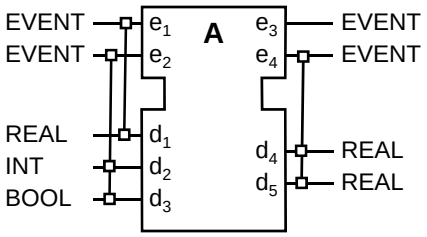

Figure 3: Example of a IEC 61499 function block.

3.1.2 Basic Function Block

A *basic function block* (BFB) is implemented by means of an *Execution Control Chart* (ECC) and one or more algorithms. The ECC is an automaton consisting of states and guarded transitions. Each state can be associated with zero or

more actions. An action can specify an algorithm which should be executed once the state is reached, and an output event port that will be activated. We differentiate between two types of states: stable states in which execution of ECC stops until a new event arrives at the input ports, and transitional states which do not require an event for ECC to move to another state.

ECC transitions are labeled by input events that cause the transition to be performed, and a transition can also be guarded by boolean conditions over the input variables. Moreover, transitions are ordered, to ensure deterministic behaviour in cases where more than one transition condition is satisfied for a given state. The use of input events in ECC transitions is not strictly defined by the IEC 61499 standard, but 4DIAC and FORTE implementation limits each transition to use only one event variable, which is reset after the transition is activated.

Figure 4 shows the ECC of the function block in Figure 3. From the startup state, the arrival of event e_1 results in the execution of the algorithm *init*, followed by the sending of output event e_2. Similarly, in the waiting state, the arrival of event e_3 results in execution of *main* and sending e_4, but only if data port d_2 contains a positive value. According to the WITH qualifier, *main* can sample data ports d_2 and d_3 and produce output data to both output data ports.

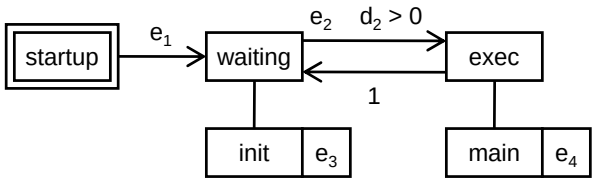

Figure 4: Execution Control Chart example.

BFBs (as implemented in FORTE) are strictly event driven – execution of a BFB can only start when an event is received at one of the input ports, and once the execution stops it will not continue until the next event arrives. One execution cycle of a BFB is called a *run*. A single run can traverse more than one ECC state in case the ECC contains transitional states, and thus result in an arbitrary number of algorithm executions and output events. Each run is atomic, meaning that no other function block execution will interrupt it.

3.1.3 Service Interface Function Blocks

Service interface function blocks (SIFBs) are designed to be used as interfaces to external hardware or services. The definition of SIFB functionality is not specified by the standard, and although they contain a sequence diagram describing their behavior, the functionality might not be fully documented in detail. The SIFBs can also bypass the function block interface for communication with other parts of the system.

SIFBs can be of two types: passive and active. The execution semantics of a passive SIFB is similar to that of BFBs, in that a passive SIFB is in idle state until an input event triggers it for execution. After the execution, a passive SIFB returns to idle state, in which it stays until it receives a new event. Active SIFBs, on the other hand, do not need input events to trigger their execution. They can also start their execution as a reaction to an external trigger (e.g. a resource interrupt or timer). Thus, active SIFBs can be viewed as event sources.

3.1.4 Function Block Network

A *function block network* (FBN) combines a number of function block instances to achieve more complex functionality. A FBN consists of a set of function blocks of arbitrary types (BFB, SIFB or CFB) and connections between the ports of these function blocks. As a result of the separation of event and data ports, the flow of control and data are clearly distinguished in FBNs.

The IEC 61499 standard does not provide a clear semantics of FBN execution. The FORTE implementation that we consider in this report adopts fully event-driven, sequential execution based on the *event dispatcher* concept. When one of the function blocks in the network is triggered for execution (either by an input even, or an external trigger in case of an active SIFB) it is added to a first-in-first-out queue in the event dispatcher. When no function block is currently executing, a new function block from the event dispatcher queue is selected and its execution is invoked.

There are also implementations that use a synchronous approach to function block execution. The synchronous approach defines discrete time steps in which function blocks are executed. The run-time framework performs cyclic scans of all function blocks and executes those that have pending input events. The propagation of output events and data produced by the function blocks during one execution cycle, is performed at the end of the cycle, and thus influence execution in the next cycle.

3.1.5 Composite Function Block

A *composite function block* (CFB) has an implementation defined by a FBN, with additional connections between the ports of the enclosing CFB and the ports of the function blocks in the network. There is, however, no clear specification of the execution semantics of CFBs. In some implementations a CFB is treated like a concrete entity with semantics similar to that of a BFB, which for example means that each invocation of the CFB is atomic. Thus, once a CFB is triggered for execution (and the triggering event is propagated to the FBN inside the CFB) no new events will be propagated to the network until the execution is finished. The downside of this approach is possible low reactivity of systems, since once execution of a CFB has started, this will block the execution of all other function blocks in the system until it is finished.

An alternative, used for example in the FORTE implementation, is the *transparent hull* approach. In this approach event ports of the CFB interface are

directly connected to the event ports of the enclosed function block, and thus any event arriving to the CFB is directly forwarded to the FBN (with a possibility of buffering). As a result, atomicity of CFB execution is not guaranteed with this approach. Although this approach results in more reactive systems it allows for data inconsistencies and makes systems more difficult to analyze.

3.2 Analysis

Currently, analysis of IEC 61499 models is mostly based on verifying functional correctness of systems [26,28]. Analysis of extra-functional properties of systems has rarely been explored. An examples of such analysis for IEC 61499 is worst-case reaction time analysis [15].

Although not defined by the standard itself, formal definition of model elements [10] and execution semantics [11] has been published in the research community. Because the standard specification does not unambiguously define execution semantics this work covers different versions of execution semantics currently used in implementations.

The IEC 61499 standard allows definition of attributes for modeling elements. The types of attributes are not pre-defined – they can hold any functional or extra-functional information about an element. An attribute type definition consists of attribute name, data type, default value, associated element and the points in the life-cycle during which the attribute can be used. Inheritance of attributes in supported, for example function block instances inherit attributes from their respective function block type.

3.3 Run-time Implementation

Main approach used for implementation of IEC 61499 is by providing a runtime environment which executes function block networks and dispatches execution of function blocks. ISaGRAPH [14], FORTE [21] and FBRT [13] are amongst most popular of such run-time environments. An advantage of run-time environments is that they provide flexibility for system deployment. Although use of such environments should promote portability of systems in the case of IEC 61499 it is not fully so. Poorly defined execution semantics resulted in all these environments providing different implementation of FB execution, thus sometimes limiting the ability to reuse systems on different platforms. The additional processing power and memory resources that the environments require can make them less suitable for use in embedded systems where such resources are often very scarce.

The ISaGRAF tool-set also supports generation of executable C code based on IEC 61499 system models. To deploy this code developers need to compile and link it with ISaGRAF C run-time libraries. Implementation of the IEC standard by code synthesis has also been explored [27]. The aim of using code synthesis is to provide a more efficient implementation of the standard. However, the presented synthesis results in monolithic code which is hard to maintain, update and reconfigure.

4 Comparison

This section will give a detailed comparison of the software levels used in ProCom and IEC 61499. We will first present similarities and differences of different model constructs of the two models. Next we will give an overview of analysis techniques developed for both and how they differ. In the end, we will describe run-time implementations of ProCom and IEC 61499 and explain benefits and downsides of the approaches.

4.1 Constructs and semantics

In this subsection we will describe how different ProCom and IEC 61499 model constructs could be mapped and what are their main differences and similarities. The comparison will cover both syntax and semantics of the constructs. We will also point out model elements and concepts that are present in one model have no equivalent in the other.

4.1.1 Component Interface

The ProSave component interface and the interface of function blocks are very similar. They both separate event and data inputs and outputs, and the explicit relation between data and event ports, which in ProSave is achieved by port groups, is specified by WITH qualifiers in IEC 61499. One difference, however, is that the IEC standard permits one data port to be associated to more than one event port, while in ProCom a data port always belong to exactly one port group.

A notable difference is that in ProCom component interface is a very strong concept that promotes use of components as black-boxes. All component types have the same syntax and semantics at the interface level. This facilitates reuse of components, analysis results and synthesized code without detailed knowledge about the component internals, and allows reasoning about unimplemented components in early stages of the development (see Section 4.1.6). Such strong definition of the function block interface is not present in IEC 61499. For example, ProSave component interface explicitly shows which output events will be produced after an input event is consumed, while in IEC 61499 this is visible only by looking at the component's internals. Moreover, the IEC standard does not provide same semantics on the interface level for all function block types.

4.1.2 Basic Function Blocks

The execution semantics of primitive ProSave components and BFBs are also quite similar. In both cases, component execution is triggered by an incoming event, their functionality is executed, and after that the component returns to an idle state until the next event arrives. In ProSave, this execution results in exactly one event produced at each trigger output, while the execution of a BFB can produce multiple events at a single output port.

Functionality of a primitive ProSave component is defined by one an entry function per input event port, and this function is executed once when the component is triggered for execution on that port. In BFBs the implementation is more complex, since the ECC determines what algorithm(s) to execute in response to an input event, based on the internal state and the current values of the input data ports. In ProCom, such variable execution would be handled inside the code, and thus not as easily available for analysis.

In IEC 61499, execution of a BFB is atomic, i.e., an executing BFB will not be interrupted, but ProCom components are allowed to preempt each other. However, the ProCom semantics defines that in such cases components will still functionally behave as if their execution is atomic, since input data ports are read only when the component is triggered, and the value used internally remains unchanged during the execution regardless of any new data received.

4.1.3 Service Interface Function Blocks

SIFBs are elements that are very hard to fit into the overall ProCom approach. The first concern is that the functionality (including communication with other elements) of a SIFB can be hidden. In ProCom components can interact with their environment only through their interfaces and their functionality is clearly defined by an algorithm or a composition of other components. However, SIFBs may provide a description of its functionality in the form of sequence diagrams.

Another important difference between SIFBs and all other ProSave components is that SIFBs can be active. Their execution can be triggered by means other than an event at their input port. In ProSave only passive components are supported. Active components are allowed only on the ProSys level of ProCom.

4.1.4 Function Block Networks

The equivalent of FBNs can be seen in a composition of ProSave components. In both cases components are combined together to provide a desired functionality. Also, both have event flow detached from data flow.

Apart from component instances and connections between them special *connector* entities are used when composing ProSave components. ProSave connectors enable operations such as forking and joining event and data paths or selection of event paths based on data values. The IEC 61499 standard does not provide such entities, but relies on a standardized set of ordinary function blocks for this functionality. For example, there exist standardized function blocks for forking or joining event paths. Also, IEC 61499 allows multiple connections to be attached to a single data port, thus providing forking and joining of data without explicit connectors.

4.1.5 Composite Function Blocks

As ProCom insists on a strong notion of component interface where all components, including composites, adhere to the same execution semantics. This

means that although parallel execution of composite components is allowed, the semantics still ensures that execution of a composite component will not be influenced by other components executing concurrently. When using the approach to execution of CFBs as entities same is true for the IEC standard(given that a CFB does not include an instance of an active SIFB). However, in case of transparent hull approach CFBs the execution semantics differs from the BFB execution semantics, and the execution semantics of composite ProCom components. During run-time systems behaves as if the whole function block hierarchy has been flattened so such systems could be compared to a non-hierarchical compositions of ProCom components.

4.1.6 Unimplemented Components

IEC 61499 does not support development and analysis of systems containing components of different maturity. The notion of unimplemented components which ProCom supports is not present in IEC 61499. Components with undecided realization bring more flexibility in system development enable a mix of top-down and bottom-up approaches. In ProCom, as opposed to IEC 61499, developers can analyze, do partial synthesis and experiment with different deployment configurations in early stages of system development.

4.2 Comparison of Analysis

One of the main aims guiding the development of ProCom was to support analysis of different functional and extra-functional properties of systems before their deployment. In IEC 61499, analysis is not one of the primary concerns. Most techniques just provide verification of system functionality. Support for analysis of extra-functional properties is very rare. Analysis in IEC 61499 is mostly on system level, and most techniques assume complete systems with fully implemented functionality. There is also no support to propagate analysis results back to individual model elements. ProCom allows for analysis of systems in their early stages of development using estimated values for parts of the system that have no implementation defined. Also, in ProCom analysis can be limited only to parts of a system (for example a subsystem or a composite component) specified by the developer. Results of such analysis can be propagated back to system model and used to influence further development of the system. Analysis results for subsystems or composite components can be attached to these model elements and possibly reused. As a result of the explicit relation between input and output in ProCom, the control and data flow in a collection of interconnected components is very explicit, which simplifies analysis in general but particularly facilitates compositional analysis addressing one hierarchical level at a time. Compositional analysis is currently not supported in IEC 61499.

The Attribute Framework which is part of ProCom provides some benefits over the attribute definition supported by the IEC standard. For example, it enables defining condition under which the attribute value is valid, when the component is reused in new contexts. The Attribute Framework also supports

defining more than one value for a single attribute, each of them having different validity conditions.

4.3 Comparison of Run-time Implementations

Most current IEC 61499 solutions implement systems by deploying them to run-time frameworks. ProCom uses the code synthesis approach for implementing and deploying systems to tasks of a real-time operating system, which is more common in hard real-time domain. Run-time frameworks provide more flexibility in deploying systems as the resulting systems consist of loosely bound elements. On the other side, code provided by synthesis tends to use less of memory and processing resources, and is more appropriate for optimizations. ProCom tries to make up for the reduced flexibility of code synthesis by a non-monolithic approach to synthesis. This includes advanced techniques like synthesis of reusable composite units, partial code generation in early stages of system development and flexible flattening of component hierarchy. Code generation techniques have also been explored in the context of IEC 61499 [27], but this implementation does not provide any advanced techniques mentioned above.

5 Feasibility of transfer

Comparison has shown that the ProCom component model and the IEC 61499 standard have many similarities, both in syntax and semantics. In this section we will describe how some of the ideas, techniques and parts of ProCom implementation could be implemented in the context of the IEC 61499 standard. Detailed descriptions of all model elements and techniques are given in previous sections.

5.1 Analysis

As shown in comparison, ProCom provides more advanced analysis techniques than the ones developed for the IEC 61499 model. This is especially true for analysis of extra-functional system properties. In this subsection we will describe what are the possible ProCom analysis techniques that could be applied to IEC 61499.

5.1.1 Transfer of the Attribute Framework

The 4DIAC IDE is an open source engineering tool for developing IEC 61499 systems. It is built on top of the Eclipse framework which allows for simple integration of new functionality through plug-ins. The PRIDE, IDE developed for ProCom, is also built as an Eclipse application. art of the PRIDE is the Attribute Framework – plug-in for management of extra-functional properties. As the implementation of Attribute Framework is not fully ProCom specific it may be possible to transfer the plug-in to the 4DIAC IDE and apply it to the

IEC 61499. This of course depends on the availability of the 4DIAC IDE source code and the concrete implementation of the tool.

5.1.2 Timing Analysis

Given the similarities of the two models, the Worst Case Execution Time (WCET) analysis developed for ProCom could be transferred to IEC 61499. This would allow analysis of timing properties of IEC 61499 systems based on system models. Such analysis can be performed in early stages of their development, even before the system is fully implemented. However, transfer of the analysis method would not be straightforward, as elements such as ECCs and active SIFBs need to be taken into account.

Dealing with the ECC

As there is no equivalent for ECC in ProCom it is not possible to directly transfer any of the ProCom analysis techniques to the IEC 61499. Hovewer, different solutions for taking the ECC into account can be envisioned.

Simplest solution for would be to limit the ECC for BFBs used in analysis to having only one ECC state. Similar approach would be to have the BFBs define values that are safe a overestimation of all ECC states at the interface level, removing the need to take the ECC into account during analysis. In both cases existing ProCom analysis techniques would not have to be changed. Although these solutions are valid they would severely limit use of the IEC 61499 standard (or the analysis techniques) and would result in analysis techniques that would hardly be useful.

Another way of dealing with the ECC would be to find all different ECC execution runs for each event input, but not taking into account data inputs or internal data values of BFB states. Then, depending on the analysis technique we could chose one or more run that would be used for analysis. Implementing such an approach seems very plausible, and would provide useful analysis results.

Best analysis results could be obtained if we would use parameterized analysis of the ECC, taking into account data inputs and internal BFB data and states. Applying such parameterized analysis would be quite difficult.

Supporting Active SIFBs

The lower level of ProCom, ProSave, does not support active components. Although the function block model of IEC 61499 is very similar to the ProSave level it supports active service interface function blocks. This means that we would not be able to directly apply timing analysis used on the ProSave level to IEC 61499. Solution to this problem would be to combine ProSave timing analysis with parts of analysis for the upper ProCom level, ProSys, which supports active components.

5.1.3 Fault Propagation Analysis

Fault propagation analysis for ProCom enables reasoning about how errors propagate through a system. It relies on definition of fault propagation between inputs and outputs of individual components. To transfer such analysis to IEC

61499 we would first have to provide a way to define fault propagation specification for IEC 61499 elements. Then, we would need to adapt the analysis algorithm to support semantics of IEC 61499 function blocks and function block networks. When trying to apply fault propagation analysis we would also face similar problems with ECCs and active SIFBs described in timing analysis section.

5.1.4 Applying REMES

The REMES language used to describe both functional and extra-functional behavior of ProCom components could also be used for the same purpose in the IEC 61499 function block model. REMES is not tightly connected to ProCom, and by that suitable to apply to other models. For the purpose of applying REMES to the IEC 61499 standard, we would have to define mapping between elements of the two models. This would result in the abbility to analyze functional and extra-functional properties of IEC 61499 systems using analysis methods developed for REMES.

5.2 Implementation

Comparison of ProCom and IEC 61499 has shown that the two have different approaches to run-time execution. Transferring the complete ProCom synthesis to IEC 61499 would require a considerable amount of effort and additional development as ProCom synthesis is still a prototype. However, some approaches used to synthesize ProCom systems could bi applied to existing IEC 61499 implementations. In the following subsections we will describe them stating possible benefits they could provide.

5.2.1 Synthesis of the IEC 61499 to a Real-Time Operating System Tasks

Synthesis of IEC 61499 systems to a real-time operating system tasks would provide an implementation suitable for hard real-time embedded systems. As with ProCom, methods for assuring that generated code follows the execution semantics of IEC 61499 would have to be developed. Such implementation of the IEC standard would allow for concurrent and more efficient execution of systems. Advanced techniques such as hierarchical scheduling could also be applied. However, implementing such synthesis would not be trivial as we would have to implement not only local function block communication, but also communication using different networks for distributed function blocks. ProCom synthesis has already been demonstrated in the research context. Implementation of such synthesis for IEC 61499 just for the purpose of proving the concept does not seem to justify the needed efforts.

5.2.2 Synthesis of Reusable Units

One of the approaches used in ProCom is synthesis of reusable units. This allows for generating code of composite components or subsystems which is context-independent. Once generated code can be stored together with other models of a component and can easily be reused in different systems, resulting in more scalable synthesis. Resulting systems are also more robust as changes in parts of the system do not require re-generating and changing code of the whole system. Such approach could possibly be applied to an existing IEC 61499 synthesis implementation.

5.2.3 Flexible Semantics of Composites

Two main approaches to CFB execution implemented for the IEC 61499 standard are execution as an entity and execution as a transparent hull. We have already described details of these approaches and how they impact predictability, analyzability and responsiveness of IEC 61499 systems. As each of these execution types brings some benefits it would be sound to try to combine them and provide the ability for trade-of of their characteristics. However, none of the current implementations support this.

A model transformation similar to that used in ProCom for flattening the model hierarchy could be used to allow for combination of the two CFB execution semantics. These transformation is used to preserve ProCom composite component execution semantics. It consists of insertion of components and connectors that simulate locking and synchronization normally done by the composite component interface. The same approach could be used for achieving entity-type execution of selected CFBs in IEC 61499 implementations which use the transparent hull approach. In implementations that execute CFBs as entities flattening can be used to provide transparent hull execution. By applying these techniques we could allow for both execution types to be used in the same systems. Developers would then be able to select execution type for each CFB, using the one which provides most benefits for the CFB they are developing.

6 Conclusions

The overall result of the comparison shows that ProCom and IEC61499 are very similar in terms of language constructs and semantics, and that the main difference is the degree of freedom to interpret the detailed IEC61499 semantics, for example the operational semantics of composite function blocks. Another important difference is the way the functionality of basic function blocks is defined by automata in IEC61499 but directly by code in ProCom.

We have identified a number of ideas and techniques related to analysis and synthesis in ProCom, that would be feasible to migrate to a IEC61499 context with various effort in terms of required adjustments. The plan for the remaining

project is to take a closer look at one of them, namely compositional model-level execution time analysis. This analysis is suitable because it is not too complex to be addressed within this project, while still involving some of the key challenges such as dealing with the ECCs of basic function blocks.

Another direction of future work is to identify restrictions to the way the IEC6149 constructs are used, or in the interpretation of semantic ambiguities, that would significally improve the analysability of IEC61499 models.

Acknowledgement

This work described in this report has been performed in the ASSIST project at Mälardalen University, funded by the ABB Software Research Grant Program. In particular, we would like to thank Kristian Sandström and Magnus Larsson at ABB Corporate Research Center in Västerås, Sweden, for providing valuable inputs.

References

[1] Mikael Åkerholm, Jan Carlson, Johan Fredriksson, Hans Hansson, John Håkansson, Anders Möller, Paul Pettersson, and Massimo Tivoli. The SAVE approach to component-based development of vehicular systems. *Journal of Systems and Software*, 80(5):655–667, May 2007.

[2] Hüseyin Aysan, Sasikumar Punnekkat, and Radu Dobrin. Error modeling in dependable component-based systems. In *IEEE International Workshop on Component-Based Design of Resource-Constrained Systems (CORCS'08)*, July 2008.

[3] Etienne Borde and Jan Carlson. Automatic synthesis and adaption of gray-box components for embedded systems - reuse vs. optimization. In *3rd IEEE International Workshop on Component-Based Design of Resource-Constrained Systems (CORCS)*. IEEE Computer Society, July 2011.

[4] Etienne Borde and Jan Carlson. Towards verified synthesis of procom, a component model for real-time embedded systems. In *14th International ACM SIGSOFT Symposium on Component Based Software Engineering (CBSE)*. ACM, June 2011.

[5] Etienne Borde, Jan Carlson, Juraj Feljan, Luka Lednicki, Thomas Leveque, Josip Maras, Ana Petricic, and Séverine Sentilles. Pride - an environment for component-based development of distributed real-time embedded systems. In *9th Working IEEE/IFIP Conference on Software Architecture*. IEEE, June 2011.

[6] Tomáš Bureš, Jan Carlson, Ivica Crnković, Séverine Sentilles, and Aneta Vulgarakis. ProCom – the Progress Component Model Reference Manual,

version 1.0. Technical Report MDH-MRTC-230/2008-1-SE, Mälardalen University, June 2008.

[7] Tomáš Bureš, Jan Carlson, Séverine Sentilles, and Aneta Vulgarakis. A component model family for vehicular embedded systems. In *Proceedings of ICSEA*. IEEE, 2008.

[8] Jan Carlson. Timing analysis of component-based embedded systems. In *15th International ACM SIGSOFT Symposium on Component Based Software Engineering*. ACM, June 2012.

[9] Jan Carlson, Juraj Feljan, Jukka Mäki-Turja, and Mikael Sjödin. Deployment modelling and synthesis in a component model for distributed embedded systems. In *Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 74–82. IEEE Computer Society, 2010.

[10] G. Cengic and K. Akesson. On Formal Analysis of IEC 61499 Applications, Part A: Modeling. *Industrial Informatics, IEEE Transactions on*, 6(2):136 –144, may 2010.

[11] G. Cengic and K. Akesson. On formal analysis of IEC 61499 applications, Part B: Execution semantics. *Industrial Informatics, IEEE Transactions on*, 6(2):145 –154, may 2010.

[12] Lars Grunske. Towards an integration of standard component-based safety evaluation techniques with SaveCCM. In *Quality of Software Architectures, Second International Conference on Quality of Software Architectures*, volume 4214 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2006.

[13] Holobloc Inc. Function block development kit (FBDK), May 2012. http://www.holobloc.org/.

[14] ISaGRAF, May 2012. http://www.isagraf.com/.

[15] M.M.Y. Kuo, Li Hsien Yoong, S. Andalam, and P.S. Roop. Determining the worst-case reaction time of IEC 61499 function blocks. In *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*, pages 1104 –1109, july 2010.

[16] Thomas Leveque, Etienne Borde, Amine Marref, and Jan Carlson. Hierarchical composition of parametric WCET in a component based approach. In *14th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing*. IEEE, March 2011.

[17] Thomas Leveque, Jan Carlson, Séverine Sentilles, and Etienne Borde. Flexible semantic-preserving flattening of hierarchical component models. In *37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE Computer Society, August 2011.

[18] Cristina Seceleanu, Aneta Vulgarakis, and Paul Pettersson. REMES: A resource model for embedded systems. In *In Proc. of the 14th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2009)*. IEEE CS, June 2009.

[19] Séverine Sentilles, Petr Štěpán, Jan Carlson, and Ivica Crnković. Integration of Extra-Functional Properties in Component Models. In *12th International Symposium on Component Based Software Engineering*. Springer, 2009.

[20] Séverine Sentilles, Aneta Vulgarakis, Tomáš Bureš, Jan Carlson, and Ivica Crnković. A component model for control-intensive distributed embedded systems. In *11th Int. Symposium on Component Based Software Engineering*. Springer, 2008.

[21] T. Strasser, M. Rooker, G. Ebenhofer, A. Zoitl, C. Sunder, A. Valentini, and A. Martel. Framework for Distributed Industrial Automation and Control (4DIAC). In *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, pages 283 –288, july 2008.

[22] T. Strasser, A. Zoitl, J.H. Christensen, and C. Su andnder. Design and execution issues in IEC 61499 distributed automation and control systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 41(1):41 –51, jan. 2011.

[23] C. Sunder, A. Zoitl, J.H. Christensen, M. Colla, and T. Strasser. Execution Models for the IEC 61499 elements Composite Function Block and Subapplication. In *Industrial Informatics, 2007 5th IEEE International Conference on*, volume 2, pages 1169 –1175, june 2007.

[24] Aneta Vulgarakis, Jagadish Suryadevara, Jan Carlson, Cristina Seceleanu, and Paul Pettersson. Formal semantics of the ProCom real-time component model. In *35th Euromicro Conference on Software Engineering and Advanced Applications*, 2009.

[25] V. Vyatkin. The IEC 61499 standard and its semantics. *Industrial Electronics Magazine, IEEE*, 3(4):40 –48, dec. 2009.

[26] V. Vyatkin. IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review. *Industrial Informatics, IEEE Transactions on*, 7(4):768 –781, nov. 2011.

[27] Li Hsien Yoong, P.S. Roop, and Z. Salcic. Efficient implementation of IEC 61499 function blocks. In *Industrial Technology, 2009. ICIT 2009. IEEE International Conference on*, pages 1 –6, feb. 2009.

[28] A. Zoitl, T. Strasser, K. Hall, R. Staron, C. Sünder, and B. Favre-Bulle. The past, present, and future of IEC 61499. *Holonic and Multi-Agent Systems for Manufacturing*, pages 1–14, 2007.