Chapter 40
Exploring Options for Modeling of Real-Time Network Communication in an Industrial Component Model for Distributed Embedded Systems

Saad Mubeen, Jukka Mäki-Turja and Mikael Sjödin

Abstract In this paper we investigate various options for modeling real-time network communication in an existing industrial component model, the rubus component model (RCM). RCM is used to develop resource-constrained real-time and embedded systems in many domains, especially automotive. Our goal is to extend RCM for the development of distributed embedded and real-time systems that employ real-time networks for communication among nodes (processors). The aim of exploring modeling options is to develop generic component types for RCM capable of modeling real-time networks used in the industry today. The selection of new component types is based on many factors including compliance with the industrial modeling standards, compatibility with the existing modeling objects in RCM, capability of modeling legacy systems and legacy communications, ability to model and specify timing related information (properties, requirements and constraints), ease of implementation and automatic generation of new components, and ability of the modeled application to render itself to early timing analysis.

Keywords Distributed embedded systems · Real-time systems · Real-time network · Component-based development · Automotive embedded systems

S. Mubeen (✉) · J. Mäki-Turja · M. Sjödin
Mälardalen Real-Time Research Centre (MRTC), Mälardalen University,
Västerâs, Sweden
e-mail: saad.mubeen@mdh.se

J. Mäki-Turja
e-mail: jukka.maki-turja@mdh.se

M. Sjödin
e-mail: mikael.sjodin@mdh.se

J. Mäki-Turja
Arcticus Systems, Järfälla, Sweden

J. J. Park et al. (eds.), *Proceedings of the International Conference on Human-centric Computing 2011 and Embedded and Multimedia Computing 2011*, Lecture Notes in Electrical Engineering 102,
DOI: 10.1007/978-94-007-2105-0_40,

40.1 Introduction

Embedded systems are found in many domains and their applications range from simple consumer products to sophisticated robotic systems. Approximately 98% of the processors produced today are embedded processors [1]. The software that runs on these processors (embedded software) has drastically increased in size and complexity in the recent years. For example, a modern premium car contains approximately 100 embedded processors and the embedded software may consist of nearly 100 million lines of code [2]. In order to deal with such complexity, the research community proposed model- and component-based development of embedded systems [3, 4].

In distributed real-time and embedded systems, the functionality of an application is distributed over many nodes (processors) and the nodes communicate with each other by sending and receiving messages over a real-time network. A component model for the development of these systems is required to support resource efficient development, abstraction of the application software from communication infrastructure, modeling of real-time network communication, development of nodes to be deployed in legacy systems with predefined rules of communication, early analysis during the development, and efficient development tools.

In this paper, we explore various options for the development of special purpose components capable of modeling real-time network communication in distributed embedded systems. These components are to be added in the rubus component model (RCM) which is an industrial model used for component-based development of resource-constrained real-time and embedded systems in many domains. In this paper, we focus on component-based development of embedded systems in automotive domain. RCM has evolved over the years based on the industrial needs and implementation of the state-of-the-art research results. At present, RCM is able to develop only single-node real-time and embedded systems. Our aim is to enable RCM to also model distributed embedded systems that employ real-time networks for inter-node (inter-processor) communication.

The purpose of the new component types is to encapsulate and abstract the communications protocol and configuration in a component-based and model-based software engineering setting. The new components should support state-of-the-practice development processes of distributed embedded systems where communications rules are defined early in the development process. The components should allow the newly developed nodes to be deployed in legacy systems (with predefined communication rules) and enable the nodes to adapt themselves to changes in the communication rules (e.g. due to re-deployment in a new system or due to upgrades in the communication system) without affecting the internal component design. Moreover, the components should enable the modeled application to render itself to early timing analysis during the development.

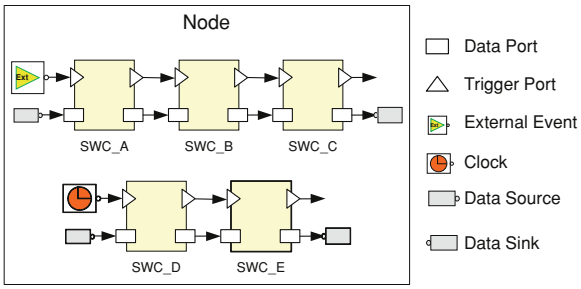

Fig. 40.1 An example system modeled in RCM

40.2 The Rubus Concept

The Rubus concept is based around the Rubus Component Model [5] and its development environment Rubus-ICE (Integrated Component development Environment) [6] providing modeling tools, code generators (also generating glue-code), analysis tools and run-time infrastructure. The overall goal of Rubus is to be aggressively resource efficient and to provide means for developing predictable and analyzable control functions in resource-constrained real-time embedded systems.

40.2.1 The Rubus Component Model

The purpose of RCM is to express the infrastructure for software functions, i.e., the interaction between the software functions in terms of data and control flow. One important principle in RCM is to separate code and infrastructure, i.e., explicit synchronization or data access should all be visible at the modeling level. In RCM, the basic component is called a software circuit (SWC). By separating code and infrastructure RCM facilitates analysis and reuse of components in different contexts (an SWC has no knowledge how it connects to other components). The execution semantics of software components (functions) is simply: upon triggering, read data on data in-ports; execute the function; write data on data out-ports; and activate the output trigger. Figure 40.1 shows how components interact with external events and sinks with respect to both data and triggering in an example real-time system modeled with RCM. The triggering events can be interrupts, internal periodic clocks, internal and external events. RCM has the possibility to encapsulate SWCs into software assemblies enabling the designer to construct the system at various abstraction levels.

40.2.2 The Rubus Code Generator and Run-Time System

From the resulting architecture of connected SWCs, functions are mapped to tasks (run-time entities). Each external event trigger defines a task and SWCs connected through the triggering chain are allocated to the corresponding task. All clock triggered "chains'' are allocated to an automatically generated static schedule that fulfills the precedence order and temporal requirements. Within trigger-chains, inter SWC communication is aggressively optimized to use the most efficient means of communication possible for each communication link. For example, there is no use of semaphores in point-to-point communications within a trigger chain. Another example is the sharing of memory-buffers between ports when there are no overlapping activation periods. The Rubus tool suite facilitates the optimization of the construction of schedule and allocation of SWCs to tasks, e.g., optimization of response times for different types of tasks or memory usage. The run-time system executes all tasks on a shared stack, thus eliminating the need for static allocation of stack memory to each individual task.

40.2.3 The Rubus Analysis Framework

RCM supports the end-to-end timing analysis and resource requirement estimations. It facilitates the expression of real-time requirements and properties on the architectural level, e.g., it is possible to declare real-time requirements from a generated event and an arbitrary output trigger along the trigger chain. For this purpose, the designer has to express real-time properties of SWCs, such as worst-case execution times and stack usage. The scheduler will take these real-time constraints into consideration when producing a schedule. For event-triggered tasks, response-time analysis is performed and the results are compared to the requirements.

40.3 Related Component Models

There are a number of models that support the model-based and component-based development of distributed systems. For example, distributed component object model (DCOM) [7], Common object request broker architecture (CORBA) [8], Enterprise JavaBeans (EJB) [9], etc. These models in their original form are not suitable for the development of distributed embedded systems which are often resource constrained and have real-time requirements. This is because these models require excessive amount of computing resources, have large memory foot print and have inadequate support for modeling of real-time network communication. There are very few commercial component models for the development of distributed embedded and real-time systems in the automotive domain. In the past few years, the research community and industry focused more on the

component-based development of automotive embedded systems which led to the development of various models.

40.3.1 AUTOSAR

AUTOSAR [10] is standardized software architecture for the development of software in automotive domain. It can be considered as a distributed component model [11] in which the application software is defined by means of software components (SWCs). At design time, the virtual function bus (VFB) is responsible for the distribution of SWCs in a single node or on several nodes (according to the requirements of the application). VFB also provides virtual integration of SWCs and communication among them. The run-time representation of VFB for each electronic control unit (ECU) is defined by the run-time environment (RTE). The communication services for SWCs are provided by the basic software (BSW) via RTE.

When AUTOSAR was developed, there was no focus put on the ability to specify and handle timing related information (real-time requirements, properties and constraints) during the process of system development. On the other hand, such requirements and capabilities were strictly taken into account right from the beginning during the development of RCM. AUTOSAR enables the development of embedded software at a relatively higher level of abstraction as compared to RCM. A SWC in RCM resembles more to a runnable entity in AUTOSAR instead of AUTOSAR SWC. A runnable entity is a schedulable part of AUTOSAR SWC. As compared to AUTOSAR, RCM clearly distinguishes between the control flow and the data flow among SWCs in a node. AUTOSAR hides the modeling of execution environment. On the other hand, RCM explicitly allows the modeling of execution requirements, e.g., jitter, deadlines, etc., at an abstraction level that is very close to the functional modeling and at the same time, abstracts the implementation details.

In RCM, special purpose objects and components are used if SWCs require inter-ECU communication otherwise SWCs communicate via data and trigger ports by using connectors. On the other hand, AUTOSAR does not differentiate between intra-node and inter-node communication at modeling level. Unlike RCM, there are no special components in AUTOSAR for inter-node communication. AUTOSAR SWCs use interfaces for all types of communications which can be of two types, i.e. Sender–Receiver and Client–Server. The Sender–Receiver communication mechanism in AUTOSAR is very similar to the pipe-and-filter communication mechanism in RCM.

40.3.2 TIMMO and TADL

TIMing Model (TIMMO) [12] is an initiative to provide AUTOSAR with a timing model. TIMMO provides a predictable methodology and a language called Timing Augmented Description Language (TADL) [13]. TADL can express

timing requirements and timing constraints during all design phases in the development of automotive embedded systems. TADL is inspired by Modeling and Analysis of Real Time and Embedded systems (MARTE) [14] which is a UML profile for model driven development of real-time and embedded systems. TIMMO uses the structural modeling provided by EAST-ADL [15] (a standard domain-specific architecture description language for automotive embedded systems). TIMMO methodology and its model structure abstract the modeling of communication at implementation level where they propose to use AUTOSAR. Both TIMMO methodology and TADL have been evaluated on prototype validators. To the best of our knowledge there is no concrete industrial implementation of TIMMO. In TIMMO-2-USE project [16], the results of TIMMO will be brought to the industry.

40.3.3 ProCom

ProCom [17] is a component model for the development of distributed embedded systems. It is composed of two layers. At the upper layer, called ProSys, it models the system by means of concurrent subsystems that communicate with each other by passing messages via explicit message channels. Unlike an SWC in RCM, a subsystem is active (it has its own thread of execution). At the lower layer, called ProSave, a subsystem is internally modeled in terms of functional components. These components are implemented as a piece of code, e.g., a C function. Like SWCs in RCM, ProSave components are passive (they cannot trigger themselves and require an external trigger for activation). ProCom gets inspiration from RCM. There are a number of similarities between the ProSave modeling layer and RCM. For example, components in both ProSave and RCM are passive. Similarly, both the models separate the data flow from the control flow. Moreover, both the models use pipe-and-filter mechanism for communication. At modeling level, ProCom does not differentiate between inter-node and intra-node communication. On the other hand, RCM clearly distinguishes modeling of inter-node and intra-node communication. ProCom uses two step deployment modeling, i.e., virtual node modeling and physical node modeling [18]. At present, physical node modeling is a work in progress. The validation of a distributed embedded system is yet to be done with ProCom. The development environment and tools accompanying ProCom are still evolving.

40.3.4 COMDES-II

COMDES-II [19] provides a component-based framework for the development of distributed embedded control systems. To some extent, it is similar to ProCom as it models the architecture of a system at two levels. At upper level, an application is modeled as a network of actors (active components) which communicate with each

other by sending labeled messages. At the lower level, the functionality of an actor is modeled by means of function blocks (FBs). Similar to the SWCs in RCM, FBs are passive. The operating system (OS) employed by COMDES-II implements fixed-priority timed multitasking scheduling. On the other hand, Rubus-OS implements hybrid scheduling that combines both static–cyclic scheduling and fixed-priority preemptive scheduling [20]. COMDES-II is a relatively new research project and the support for development tools and run-time environment was provided recently [21]. On the other hand, RCM and its tool suite are relatively mature as they are being used in the industry for the development of embedded systems for more than 10 years [6].

40.3.5 RT-CORBA, Minimum CORBA and CORBA Lightweight Services

There are a number of middleware technologies introduced by the object management group (OMG) such as real-time CORBA, minimum CORBA and CORBA lightweight services. These middleware technologies can be used for the development of real-time and distributed embedded systems [22]. In some projects [23, 24], Real-time CORBA has been used to develop distributed embedded and real-time systems. Because of higher resource requirements, these models may not be suitable for the development of distributed embedded systems that are resource constrained and have hard real-time requirements.

40.3.6 Discussion

By comparing the models discussed in this section with RCM, we propose that RCM can be considered a suitable choice for the development of resource-constraint distributed embedded systems for many reasons. Some of the reasons are: its ability to completely handle and specify the timing related information (i.e., real-time require-ments, properties and constraints during all the stages of system development); it has a small run-time footprint (timing and memory overhead); it implements state-of-the-art research results; it has a strong support for development and analysis tools, etc.

40.4 Modeling of Real-Time Network Communication

A component model for the development of distributed real-time and embedded systems is required to automatically generate network communication for any type of protocol. However, this is often not the practice in the industry because of many

factors including the dependencies due to existing communications and network protocols, early design decisions about the network communication, requirement for early analysis, deployment of the developed system in a legacy system having predefined rules for communication, etc. In this section, we will introduce the model representation of a physical network. We will also explore different options to build new components in RCM capable of modeling real-time network communication.

40.4.1 Network Specification: Modeling Object for a Physical Network

In order to represent the model of communication in a physical network, we propose the addition of a new object type, i.e., the Network Specification (NS) in RCM. There will be one NS for each network protocol. It is composed of two parts, i.e., a protocol independent part and a protocol dependent part. The protocol independent part defines a message and its properties. A message is an entity that is used to send information from one node to another via a network. The properties of a message that are defined by NS include message ID, sender node ID, list of receiver nodes IDs, list of RCM signals included in the message, etc. A signal in RCM has a name, RCM data-type, resolution, real-time properties, etc.

The protocol dependent part of NS is uniquely defined for each protocol. For example, it will be different for different protocols such as CAN (Controller Area Network) [26], CANopen [27], HCAN (Hägglunds Controller Area Network) [28], MilCAN (CAN for Military Lands System domain) [29], Flexray, etc. Therefore, there is one NS per bus. The protocol dependent part of NS contains the complete information of all the frames which are sent to or received from the bus. Moreover, it describes the frame properties. A frame is a formatted sequence of bits that is actually transmitted on the physical bus. In RCM, a frame is a collection of RCM signals. The properties of a frame described by NS include an identifier, a priority, a frame type, a transmission type, a sender node ID, a list of receiver nodes IDs, period or minimum inter-arrival time, deadline, whether a frame is IN or OUT frame, real-time requirements, etc. The transmission type of a frame can be periodic, event or mixed (transmitted periodically as well as on arrival of an event).

RCM clearly distinguishes the data flow from the control flow. The components inside a single node communicate with each other by using data and control signals separately. However, if a component on one node communicates with a component on another node via a network then the signals are packed into frames. The frames are then transmitted over the network. Here, some questions arise regarding the network communication. How are the signals packed into the frames? How are the signals encoded into the frames at the sender node? How are the signals decoded from the frames and sent to the respective SWCs at the

receiver node? All the rules concerning the answers to these questions are specified in the Signal Mapping. The signal mapping is a unique object for each protocol for network communication and is an integral part of the protocol-dependent part of NS. The Signal Mapping describes the length of each signal in a frame, the type of signal encoding in a frame (e.g., signed or unsigned two's complement), maximum age of a signal guaranteed by the sender, etc.

40.4.2 Modeling Objects for Network Communication in a Node

There is a need to introduce special objects for modeling network communication in a node. These objects should be able to hide the software architecture in a node from the network protocol and vice versa. If SWCs located on different nodes intend to communicate with each other, they should only communicate (using the existing intra-node communication in RCM) with the special objects on their own node. Hence, the new objects should encapsulate and abstract the communication protocols and configuration in a node. These special objects along with NS should be able to facilitate the inter-node communication in a distributed embedded application.

To support the abstraction of the implementation of network communications in a node (processor), we will explore different options to develop new object types in the next subsection. After selecting the most suitable object types (based on the selection criteria to be discussed in the next section), we will add them in RCM. This will enable RCM to support state-of-the-practice development processes of distributed embedded systems where communication rules are defined early in the development process. The proposed extension of the model will also allow model-based and component-based development of new nodes that are deployed in the legacy systems that use predefined rules for network communication.

40.4.3 Options for Modeling Network Communication in a Node

Option 1: r*euse of existing intra-node communication*. The first option is to reuse the model of existing intra-node communication in RCM for modeling of inter-node communication (network communication). The intra-node communication in RCM is modeled by means of connectors as shown in Fig. 40.1. A connector in RCM links out-ports of one SWC with in-ports of the other in the same node. RCM allows port-to-port connections for data and trigger flow separately. In a nutshell, the idea is to use the same connectors for modeling both inter-node and intra-node communications by attaching, for example, a Boolean Specifier to it. If the connector is used for intra-node communication, then it is assigned a value "0". On the other hand, if an

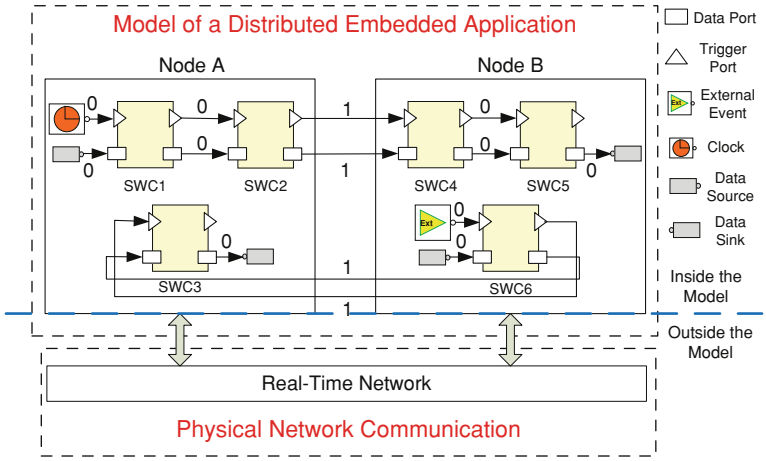

Fig. 40.2 Option1: model of a distributed real-time and embedded application

SWC on one node communicates with an SWC on another node then the specifier is assigned value equal to "1". We assume that a tool will automagically generate the run-time architecture for all communications in the modeled application. Moreover, the tool will perform the deployment of the distributed embedded application.

Figure 40.2 shows a distributed real-time application modeled with this technique. The application consists of two nodes which are connected to each other via a real-time network. There are three SWCs in each node. There are two distributed transactions (Event Chains) in the system. By distributed transaction, we mean that SWCs are in a sequence (chain) and have one single triggering ancestor (e.g., a clock, interrupt, external or internal event, etc.). In Fig. 40.2, the first distributed transaction is composed of SWC1, SWC2, SWC4 and SWC5 whereas the second is composed of only two SWCs, i.e., SWC6 and SWC3. The first distributed transaction is activated (triggered) by a clock while the second is activated by an external event.

The value of Boolean Specifier associated with each connector is "0" in case of intra-node communication, for example, a connector between SWC1 and SWC2. Similarly, the value of Boolean Specifier is equal to "1" in case of network communication, for example, a connector between SWC2 and SWC4. Although, the communication between SWC2 and SWC4 takes place via the network, a designer models the system as indicated by the upper portion of Fig. 40.2. The deployment and synthesis tools along with the run-time support will be responsible for generating intra-node and inter-node communications.

Option 2: *out- and in- SWCs for out and in frames*. The second option for modeling network communication in RCM is to introduce special purpose software circuits i.e. out software circuit (OSWC) and in software circuit (ISWC) for each frame that is to be sent or received by a node, connected to a network, respectively.

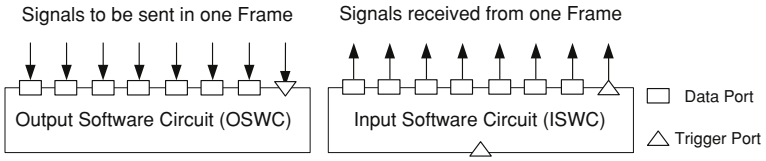

Fig. 40.3 Graphical illustration of OSWC and ISWC

Out Software Circuit (*OSWC*). it is a software circuit which denotes the data that leaves the model. An OSWC is associated with a LAN object. In RCM, LAN is an object to represent the connection between two or more nodes in the system. Formally, a LAN is defined by its name, a list of the connected nodes and NS. There is exactly one OSWC in a node for every outgoing frame on the network. Each OSWC describes all the signals that can be sent in a particular frame. As discussed earlier, a frame constains zero or more signals. An OSWC has only one trigger in-port and at least one data in-port. Each data in-port is associated with one signal in NS. Therefore, the number of data in-ports may vary depending upon the number of signals to be packed in the frame. An OSWC has no data and trigger output-ports. The OSWC component uses protocol specific rules, specified in the protocol-specific part of NS, while maping signals to frames and encoding data in the frames. In this way, OSWC provides a clear abstraction to the SWCs that send signals to one of its data in-ports. Thus, SWCs are kept unaware of the protocol-specific details such as signal-to-frame mapping, data-type encoding and transmission patterns of frames.

In Software Circuit (*ISWC*). it is a software circuit which denotes the data that enters the model. An ISWC is associated with a LAN object defined in RCM. There is exactly one ISWC component in a node for every frame received from the network. Each ISWC describes all the signals that are contained in a particular received frame. An ISWC has one unconditional trigger out-port. An unconditional trigger port produces a trigger signal every time the SWC is executed. There is at least one data out-port in the ISWC component. Each data out-port is associated with one signal in NS of the LAN object. Therefore, the number of data out-ports may vary depending upon the number of signals contained in the received frame. An ISWC has no data out-ports. There is one trigger in-port in every ISWC component which can be trig-gered when a frame arrives at a node. Figure 40.3 graphical illustrates OSWC and ISWC.

Consider an example of a node in a distributed embedded application modeled with OSWC and ISWC as shown in Fig. 40.4. Although, CAN is used for real-time network communication, this modeling approach can be applied to any real-time network protocol. Note that the figure is divided into two halves: the upper half represents the model of a node whereas the lower half depicts the physical communication including CAN controller and CAN network. There are two grey boxes outside the model called CAN SEND and CAN RECEIVE that are placed just below the sets of OSWCs and ISWCs, respectively. These gray boxed are specific for each network protocol. The frames that leave the model are denoted by

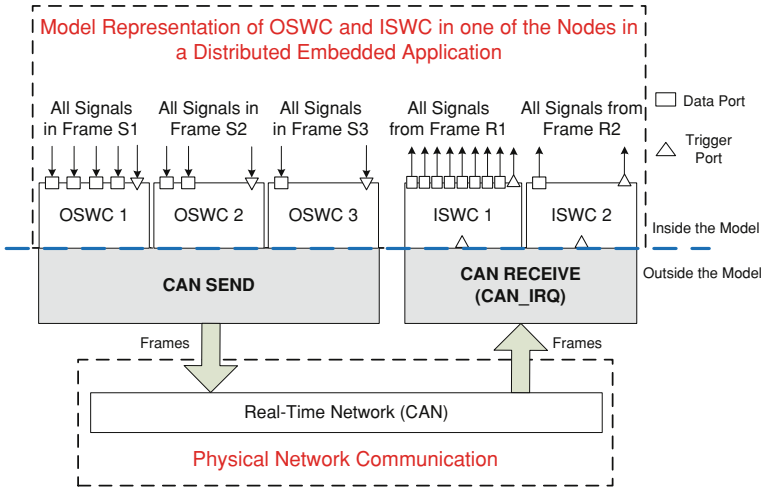

Fig. 40.4 Option2: model of a distributed real-time and embedded application

S e.g., *S1*, *S2* and *S3*. Similarly all the frames that enter the model are denoted by *R* e.g., *R1* and *R2* as shown in Fig. 40.4.

All the signals sent in frame *S1* are provided at the data in-ports of OSWC1. These signals are mapped and encoded into *S1* by OSWC1 according to the protocol-specific information available in NS. Once the frame is ready, it leaves the model as it is sent to the grey box CAN SEND. In this example, this grey box represents a CAN controller in the node which is responsible for the physical transmission of this frame. When a frame arrives at the receiving node, it is transferred by the physical network drivers to a grey box CAN RECEIVE that produces an interrupt. The frame enters the model and is transferred to the destination ISWC which extracts the signals, decodes the data and encodes it to RCM data-type. The data is placed on the data out-port of ISWC connected to the data in-port of the destination SWC and the corresponding trigger out-port is triggered (the tracing information is provided in the NS).

Option 3: n*etwork interface components for each node*. The third option is to introduce two special purpose modeling component types in RCM: the network input interface (NII) and the network output interface (NOI) as shown in Fig. 40.5.

Network input interface (*NII*) c*omponent*: it is the model representation of incoming signals from the network. Each node connected to a network will have one NII. Each NII contains one data-port and one trigger-port for each signal in every frame. When a frame arrives at the node, the physical bus driver and protocol-specific implementation of NII extract the signals (zero or more signals per frame) and encode their data in the RCM data-type. When the signal(s) is delivered, the data is placed on the data-port which is connected to data in-port of the destination SWC (the tracing information is provided in NS), and the corresponding trigger-port is triggered.

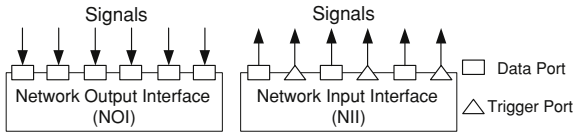

Fig. 40.5 Graphical illustration of NOI and NII Components

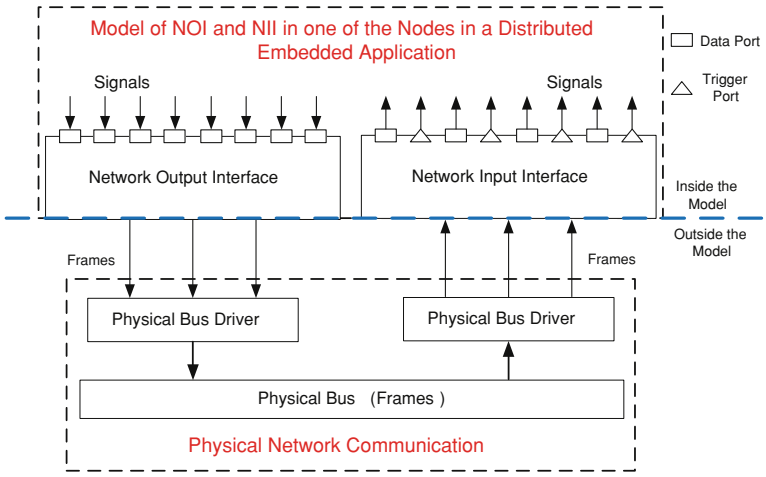

Fig. 40.6 Option3: model of a distributed real-time and embedded application

Network output interface (*NOI*) *component*: it is the model representation of the outgoing signals on a network. Each node connected to a network will have one NOI component. The major difference from the NII is that the NOI component does not have any trigger ports. Conceptually, the NOI has an implicit trigger port for each data port–however, to decrease the burden for the modeler, these ports are omitted from the model. The NII uses protocol-specific rules on how to map signals to frames and encode data in the frames specified in the protocol-specific part of NS. The NII also uses protocol specific rules to decide when to send each frame. Thus, the SWCs that use the NII are kept unaware about details such as signal-to-frame mapping, data-type encoding, and transmission patterns (common transmission patterns are: periodic, on-change, on-change with minimum distance between messages, etc.).

Consider the model representation of NOI and NII in one of the nodes connected to a real-time network in a distributed embedded application as shown in Fig. 40.6. The figure is divided into two halves: the upper half represents the model of a node in RCM whereas the lower half depicts the physical network. All SWCs in a node, requiring remote communication, send signals to the data in-ports of the NOI component. The NOI maps these signals into frames and sends the frames according to the protocol-specific rules defined in NS. When a frame arrives at a node, NII decodes the frame and extracts signals out of it according to

the information available in NS. Then, it places the extracted data on data output-port which is connected to the data in-port of the destination SWC and triggers the corresponding trigger out-port.

40.5 Components for Modeling Real-Time Network Communication

40.5.1 Selection Criteria

The selection of new component types is based on many factors including compliance with the existing modeling standards in the industry, compatibility with the existing modeling concepts and compoents in RCM, ability to abstract the implementation and configurations in distributed embedded sysems, dependencies due to legacy (already developed) systems and legacy communications, ability to model real-time properties, ease of implementation of new components, automatic generation of the new components, ability of the modeled application to render itself to early analysis, etc. One important factor in the selection is the ease to model timing information (properties, requirements, constraints) in new components that enable the modeled application to render itself to end-to-end timing analysis early during the developemnt.

40.5.2 Components Selection

Although, the modeling approach in option 1 appears to be very simple and easy to model, it may not be practical in an industrial setup because of the requirements of modeling legacy systems and legacy communications, deployment of newly developed nodes in the existing systems and requirement for early analysis of the modeled application. Moreover, this approach requires very complex tools capable of automagically deploying and synthesizing the modeled system and generating the run-time implementation of all communications in the modeled application. This approach is not very expressive to perform the response-time analysis which is one of the most important requirements during the development of real-time systems.

The modeling approach in option 2 appears to be the most suitable candidate for the development of special components. One reason is that OSWC and ISWC component types are consistent with the existing modeling elements in RCM. These components are very similar to regular SWCs and hence, it is very easy to add them in the component model. The run-time implementation of these components can be easily generated. Another advantage is the generation of OSWC and ISWC automatically from NS. Moreover, it complies with the existing modeling standards and network communication protocols used in the automotive

industry. For any network protocol such as CANopen, HCAN, MilCAN, Flexray, etc., all the modeling objects will remain the same except the protocol-dependent part of NS. This modeling approach may remain applicable if an application modeled with RCM uses the communication mechanism of AUTOSAR. In this case the grey boxes in Fig. 40.4 will represent the AUTOSAR run-time environment. A node developed with this approach can be easily deployed in a legacy distributed embedded system. In this case, same OSWC and ISWC components can be used and the rules of communication defined by the legacy system are encapsulated in NS. Since the network communication is modeled at frame level in a node, this approach provides an effective means to specify and trace the timing information for early timing analysis during the development.

The third modeling approach is a very general approach to model real-time network communication. It can be used for any type of real-time network protocol. Similar to the second approach, the protocol specific information is specified in NS while the implementation of NOI and NII components remains the same for all protocols including the rules of communication in legacy systems. NOI and NII can also be automatically generated from NS. This approach is very complex to implement mainly because these components are very different from the existing SWCs in RCM. Moreover, if these components are not implemented carefully then there is a risk of getting redundant rules for sending frames over the network. For example, assume CAN network in Fig. 40.6. The NOI sends a frame according to the protocol-specific information in NS where as the frame is physically transmitted by the CAN controller according to the CAN protocol. Although this approach facilitates to completely specify the timing related information, it is very difficult to extract the tracing information of event chains from the modeled application. Therefore an application modeled with this approach does not easily render itself to end-to-end timing analysis as compared to the second modeling approach. In future, this approach may be added in RCM because of being concise and general.

Based on the above discussion, we select the second modeling approach. Hence, we add special-purpose components OSWC and ISWC along with NS, in RCM to support modeling of real-time network communication.

40.5.3 Automatic Generation of the Selected Components

Both OSWC and ISWC components can be automatically generated from NS by a network configuration tool. The input to this tool is the protocol-specific information about the network communication and the tracing information of tasks in all the distributed transactions (event-based and periodic chains) present in the application. This information is made available from the configuration files that correspond to NS. The output of this tool is the automatically generated OSWC and ISWC components for each node in the network. This tool also carries out

mapping from NS to components and vice versa. All OSWC and ISWC components are translated into a set of SWCs to execute the protocol at run-time.

40.6 Conclusion

We explored various options to develop special purpose component types in the rubus component model (RCM). The purpose of the special components is to model network communication in distributed real-time and embedded systems. RCM is an industrial component model for the development of resource-constrained real-time and embedded systems. We introduced the capability of modeling real-time network communication in RCM. We investigated three approaches for modeling of real-time network communication. We presented the selection criteria to select the most suitable modeling technique from the three candidates. Accordingly, we added the following objects in RCM: NS (model representation of physical network); and output and input software circuit (OSWC and ISWC) component types. These components make the communications capabilities of a node in distributed embedded system very explicit, but efficiently hide the implementation or protocol details.

The criteria for the selection of new components is based on many factors, such as, compliance with the industrial modeling standards, compatibility with the existing modeling concepts and objects in RCM, ability to abstract the implementation and configurations in distributed embedded sysems, capability of modeling legacy (already developed) systems and legacy communications, ability to model and specify timing related information (properties, requirements, constraints, etc.), ease of implementation and automatic generation of new components, ability of the modeled application to render itself to early timing analysis, etc. With the introduction of new modeling capabilities in RCM, it can be considered a suitable choice for the industrial development of distributed real-time and embedded systems. There are many reasons to support this proposition such as: ability to model and specify real-time requirements, properties and constraints during all the stages of system development; small run-time footprint (timing and memory overhead); implementation of state-of-the-art research results; ability to model real-time communication (both intra-node and inter-node); strong support for development and analysis tools, etc.

In future, we will extract an end-to-end timing model from the modeled application to perform the holistic response-time analysis. We will also demonstrate how the periodic and event chains can be traced in a distributed transaction modeled with RCM. We also plan to automatically generate the implementation of OSWC and ISWC from configuration files of other specialized protocols for real-time communication such as CANopen, HCAN, MilCAN, subsets of J1939, etc. Furthermore, we also plan to build a validator in an industrial setup with the extended RCM.

Acknowledgments This work is supported by Swedish Knowledge Foundation (KKS) within the project EEMDEF. The authors would like to thank the industrial partners Arcticus Systems Sweden and Hägglunds BAE Systems for cooperation.

References

1. Broy M (2005) Automotive software and systems engineering. Formal methods and models for co-design, MEMCODE'05
2. Charette RN (2009) This car runs on code. IEEE Spectrum 46(2). http://spectrum.ieee.org/green-tech/advanced-cars/this-carruns-on-code
3. Henzinger TA, Sifakis J (2006) The embedded systems design challenge. Proceedings of the 14th international symposium on formal methods (FM), pp 1–15
4. Crnkovic I, Larsson M (2002) Building reliable component-based software systems. Artech House, Norwood
5. Hänninen K, Mäki-Turja J, Nolin M, Lindberg M, Lundbäck J, Lundbäck K-L (2008) The rubus component model for resource constrained real-time systems. 3rd IEEE international symposium on industrial embedded systems
6. Arcticus systems. http://www.arcticus-systems.com
7. Microsoft, Distributed component object model (DCOM). http://msdn.microsoft.com/enus/library/Aa286561
8. OMG. Common object request broker architecture (CORBA), Version 3.1, January 2008. http://www.omg.org/spec/CORBA/3.1
9. DeMichiel L (2002) Sun microsystems, Enterprise JavaBeans specification, Version 2.1. Sun microsystems
10. AUTOSAR Techincal overview, Version 2.2.2. AUTOSAR—AUTomotive Open System ARchitecture, Release 3.1. The AUTOSAR Consortium, August 2008. http://www.autosar.org
11. Heinecke H et al (2006) AUTOSAR—Current results and preparations for exploitation. In: Proceedings of the 7th euroforum conference, ser. EUROFORUM'06
12. TIMMO Methodology, Version 2, TIMMO (TIMing MOdel), Deliverable 7, October 2009. The TIMMO Consortium
13. TADL: Timing augmented description language, Version 2, TIMMO (TIMing MOdel), Deliverable 6, October 2009. The TIMMO Consortium
14. The UML profile for MARTE, January 2010. http://www.omgmarte.org/
15. EAST-ADL domain model specification, Deliverable D4.1.1. http://www.atesst.org/home/liblocal/docs/ATESST2-D4.1.1
16. TIMMO-2-USE. http://www.itea2.org/projects/step/2
17. Sentilles S, Vulgarakis A, Bures T, Carlson J, Crnkovic I (2008) A component model for control-intensive distributed embedded systems. In: Proceedings of the 11th international symposium on component based software engineering, pp 310–317
18. Carlson J, Feljan J, Mäki-Turja J, Sjödin M Deployment modeling and synthesis in a component model for distributed embedded systems. 36th EUROMICRO Conference on Software Engineering and Advanced Applications, pp 74–82
19. Ke X, Sierszecki K, Angelov C (2007) COMDES-II: a component-based framework for generative development of distributed real-time control systems. 13th IEEE international conference on embedded and real-time computing systems and applications, RTCSA, pp 199–208
20. Mäki-Turja J, Hänninen K, Nolin M (2007) Efficient development of real-time systems using hybrid scheduling. 9th real-time in Sweden (RTiS'07), pp 157–163
21. Guo Y, Sierszecki K, Angelov C (2008) COMDES development toolset. 5th international workshop on formal aspects of component software FACS 08, Spain

22. Catalog of specialized CORBA specifications. OMG group. http://www.omg.org/technology/documents/
23. Lankes S, Jabs A, Bernmerl T (2003) Integration of a CAN-based connection-oriented communication model into real-time CORBA. Parallel and distributed processing symposium
24. Finocchiaro R, Lankes S, Jabs A (2004) Design of a real-time CORBA event service customised for the CAN bus. In: Proceedings of the 18th international parallel and distributed processing symposium
25. GmbH RB (1991) CAN specification Version 2.0. September
26. CANopen high-level protocol for CAN-bus, Ver. 3.0. NIKHEF, Amsterdam, March 2000
27. Westerlund J (2009) Hägglunds controller area network. Network implementation specification. BAE Systems, Hägglunds, Sweden
28. MilCAN (CAN for Military Land Systems domain). http://www.milcan.org/