2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications

Analyzable Modeling of Legacy Communication in Component-Based Distributed Embedded Systems

Saad Mubeen*, Jukka Mäki-Turja*†, Mikael Sjödin* and Jan Carlson*
* *Mälardalen Real-Time Research Centre (MRTC), Mälardalen University, Västerås, Sweden*
† *Arcticus Systems, Järfälla, Sweden*
{*saad.mubeen, jukka.maki-turja, mikael.sjodin, jan.carlson*}*@mdh.se*

***Abstract*—We present extensions to the existing industrial component model Rubus Component Model (RCM). By introducing special purpose components to encapsulate and abstract the communication protocols in distributed embedded systems we allow use of legacy nodes and legacy protocols in a component-based and model-based software engineering environment. With the addition of these components, RCM will be able 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 also allows model-based and component-based development of new nodes that are deployed in the legacy systems that use predefined communication rules. We also demonstrate how an end-to-end timing model can be extracted from a distributed embedded system modeled with extended RCM. The extracted model is then used to perform an end-to-end timing analysis that we implemented in the Rubus Analysis Framework.**

***Keywords*-Model-Based Software Engineering; Component-Based Software Engineering; distributed embedded systems; end-to-end timing analysis;**

I. Introduction

Embedded systems are found in almost all electronic products around us. Their applications span over many domains including automotive, aerospace, consumer electronics, biomedical, military applications, business applications, industrial control, etc. It is claimed in [1] that more than 98 percent of the processors produced today are embedded processors. Not only the number of embedded processors has increased in the past few years but the software which runs on them, i.e., the embedded software has also drastically increased in size and complexity. In automotive domain, for example, a modern premium car contains nearly 100 million lines of code that run on about 70 to 100 embedded processors [2]. Because of the continuously increasing trend in size and complexity of embedded software, the development of embedded systems has become very complex.

Often, embedded systems are resource-constrained and have hard real-time requirements. In order to capture such requirements as early as possible during the process of system development, handle complexity of embedded software, lower development cost, reduce time-to-market and time-to-test, allow reusability and modeling at higher level of abstraction, etc., the research community proposed the use of Model-Based Engineering (MBE) and Component-Based Software Engineering (CBSE) for the development of embedded systems [3], [4]. MBE provides the means to use models throughout the process of system development while CBSE facilitates the development of large software systems by integration of software components. CBSE raises the level of abstraction for software development and makes it possible to reuse software components and their architectures. There is a great need for bringing these development techniques in the embedded systems industry.

In distributed embedded systems, the functionality is distributed over many nodes and the nodes communicate with each other through a bus or a network. Software development of distributed embedded systems is more complex compared to single processor embedded systems. When MBE and CBSE are used for the development of resource-constrained and hard real-time distributed embedded systems, modeling of communication infrastructure arises as another challenge. In the industry, embedded systems are often deployed in legacy systems (previously developed) which use predefined rules for communication. Furthermore, distributed embedded systems are often expected to use legacy network protocols for real-time communication. A component model for the development of distributed embedded systems should not only be resource-efficient, but also abstract the application software from the communication infrastructure. Moreover, it should support the modeling of legacy communications and legacy systems.

In this paper we propose an extension to a commercially available component model, the Rubus Component Model (RCM) [5], used for the development of resource-constrained real-time embedded systems in many domains especially automotive. It supports glue code generation, end-to-end response-time analysis, and resource requirement estimations. Over the years, RCM has evolved based on the industrial needs and the state-of-the-art research results. At present, RCM is able to model only single-node embedded systems. We extend RCM by adding special purpose components to it. The purpose of new components is to encapsulate and abstract the communication protocols and configuration in a component-based and model-based software engineering setting. The motivation for the extension of RCM comes

978-0-7695-4488-5/11 $26.00 © 2011 IEEE
DOI 10.1109/SEAA.2011.43

from the industrial demand to model distributed embedded systems, real-time network communication, legacy communications and legacy systems.

A. Goals and Paper Contributions

We present an extension to an existing industrial component model by introducing new components to it. Our main goals in introducing new components are:

1) Allow model-based and component-based development of new nodes that are deployed in legacy systems that use predefined communication rules.
2) Support state-of-the-practice development processes where communication rules are defined early in the development process.
3) Enable adaptation of a node when communication rules change (e.g. due to re-deployment in a new system or due to upgrades in the communication system) without affecting the internal component design.
4) Generate these special components from the information about legacy communication or from early design decisions about network communication. The generated components should be compatible with the existing entities defining functionality and communication in RCM.

These goals are to be realized in RCM. The scope of this paper is PSMs (Platform Specific Models) for distributed embedded systems. With PSM we mean that the software components have been allocated to nodes and any adaptation to specific node characteristics (e.g., device drivers and memory layouts) has been added to the model. Using our new components, nodes can be developed without explicit knowledge about the communication configuration.

One important objective during the extension of RCM is to enable the developer to specify real-time properties and analyze timing behavior of the modeled distributed embedded system. While making design decisions about the new modeling concepts and components, we placed special focus on how the modeled system will render itself to an end-to-end timing analysis. In this paper, we also show how we extract an end-to-end timing model from a distributed embedded system using the Rubus tool suite. The extracted model is then used to perform an end-to-end timing analysis that we implemented in the Rubus Analysis Framework [6], [7].

B. Paper Layout

The rest of the paper is organized as follows. Section II presents the Rubus concept, the component model and its development environment. In Section III, we present the related research and compare different modeling approaches with ours. Section IV describes the new modeling objects that support modeling of legacy communication. Section V describes the implementation of the end-to-end timing analysis of distributed embedded systems in the Rubus tool suite. Section VI concludes the paper and presents the future work.

II. Background – 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], which includes modeling tools, code generators, 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 embedded systems.

A. The Rubus Component Model

The purpose of the component model 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 is to separate functional code and infrastructure implementing the execution model, 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). It is the lowest-level hierarchical element in RCM and its purpose is to encapsulate basic functions. The SWCs interact with each other through the use of ports. An SWC can be seen as a type, or a class, that can be instantiated an arbitrary number of times. By separating functional code and the 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:

1) Upon triggering, read data on data in-ports.
2) Execute the function.
3) Write data on data out-ports.
4) Activate the output trigger.

An example system modeled with RCM, depicted in Fig. 1, shows how components interact with external events and actuators with regard to both data and triggering. The triggering events can consist of interrupts, internal periodic clocks, internal and external events. Furthermore, the component model has a possibility to encapsulate SWCs into software assemblies enabling the designer to construct the system at different hierarchical levels.

B. The Rubus Code Generator and Run-Time System

From the resulting architecture of connected SWCs, functions are mapped to run-time entities; tasks. Each external event trigger defines a task and SWCs connected through the chain of triggered SWCs (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.

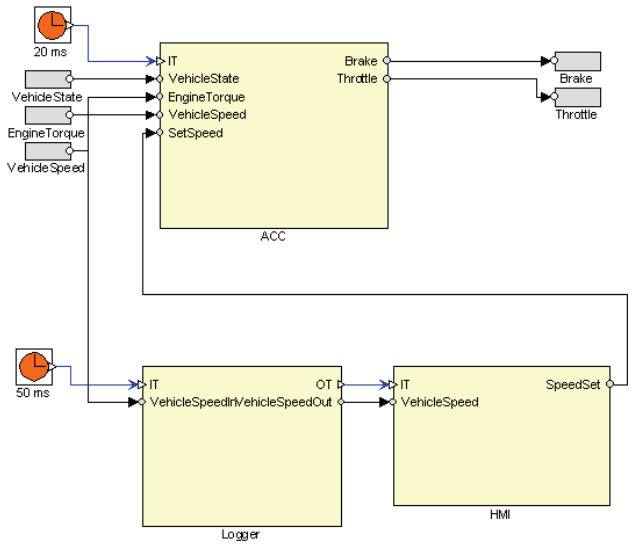

Figure 1. An example system in RCM

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 sharing of memory buffers between ports when there are no overlapping activation periods. This means that a buffer can be shared between two ports belonging to different SWCs if it can be guaranteed that these ports will never use the buffer space at the same time. This is true in the case of a trigger chain because a task early in the chain can never be active at the same time as a task late in the chain (considering the deadlines of tasks are smaller than their respective periods).

Allocation of SWCs to tasks and construction of schedule can be submitted to different optimization criterion to minimize, e.g., 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.

C. The Rubus Analysis Framework

The model also allows expressing real-time requirements and properties on the architectural level. For example, 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 calculations are performed and compared to the requirements.

III. Related Work

There exist many component models for the development of distributed systems, e.g., Distributed Component Object Model (DCOM) [8], Common Object Request Broker Architecture (CORBA) [9], Enterprise JavaBeans (EJB) [10], etc. These models in their original form are not suitable for the development of resource-constrained distributed embedded systems with hard real-time requirements because they require excessive amount of computing resources, have large memory foot print and have inadequate support for modeling of real-time communication.

There are very few commercial component models for the development of distributed embedded and real-time systems especially in automotive domain. In the last decade, automotive research community and industry has focused more on the component-based development of automotive embedded systems which led to the development of various solutions, approaches, methodologies, and models. Some of them are discussed below.

AUTOSAR

AUTOSAR (AUTomotive Open System ARchitecture) [11] is a standardized software architecture for the development of software in automotive domain. It can be viewed as a standardized distributed component model [12]. In AUTOSAR, the application software is defined in terms of Software Components (SWCs). The distribution of SWCs, their virtual integration and communication at design time is handled by the Virtual Function Bus (VFB). The run-time representation of VFB for each Electronic Control Unit (ECU) is defined by the Run-Time Environment (RTE). The communication services are provided by the Basic Software (BSW) via RTE to the AUTOSAR SWCs.

When AUTOSAR was being developed, there was no focus placed on the specification and handling of real-time requirements and properties 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 describes embedded software development at a relatively higher level of abstraction compared to RCM. A Software Circuit in RCM more resembles to a runnable entity compared to 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 close to the functional modeling while abstracting the implementation details.

In RCM, special Software Circuits, which are integral part of our contribution in this paper and will be introduced in the next Section, are used if SWCs require inter-ECU

communication; otherwise, SWCs communicate via data and trigger ports. 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 for component interconnection used in RCM.

TIMMO

TIMMO (TIMing MOdel) [13] is an initiative to provide AUTOSAR with a timing model. It describes a predictable methodology and a language, TADL (Timing Augmented Description Language) [14], to express timing requirements and timing constraints during all design phases in the development of automotive embedded systems. TADL is inspired by MARTE (Modeling and Analysis of Real Time and Embedded systems) [15] which is a UML profile for model-driven development of real-time and embedded systems. TIMMO development methodology makes use of structural modeling provided by EAST-ADL [16] which is a domain specific architecture description language used in automotive domain. TIMMO methodology and its model structure abstract the modeling of communication at implementation level of EAST-ADL 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 [17], the results of TIMMO will be further validated and brought to the industry.

ProCom

ProCom [18] is a two-layer component model for the development of distributed embedded systems. At the upper layer, called ProSys, it models a system with concurrent subsystems which communicate by passing messages via explicit message channels. Unlike an RCM SWC, a subsystem is active which means that it has its own thread of execution and hence, it can be triggered periodically or by internal events. At the lower layer, called ProSave, a subsystem is internally modeled in terms of functional components which are implemented as a piece of code, for example, a C function. Like RCM SWCs, ProSave components are passive which means that they cannot trigger themselves and hence, they require an external trigger for activation.

ProCom is inspired by RCM, and 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 clearly separate data flow from control flow among the components. Moreover, the communication mechanism for component interconnection used in both the models is pipe-and-filter. At modeling level, ProCom does not differentiate between inter-node and intra-node communication. ProCom uses two-step deployment modeling, i.e., virtual node modeling and physical node modeling [19]. At present, physical node modeling is a work in progress. The validation of a complete distributed embedded system, modeled with ProCom, is yet to be done. Moreover, the development environment and the tools accompanying ProCom are still evolving.

COMDES-II

COMDES-II (COMponent-based design of software for Distributed Embedded Systems) provides a component-based framework for the development of distributed embedded control systems [20]. It models the architecture of a system at two levels. At upper level, an application is modeled as a network of actors that are active components. Actors communicate with each other by sending labeled messages. At the lower level, the functionality of an actor is modeled in terms of Function Blocks which are passive components similar to the SWCs in RCM. The Operating System (OS) employed by COMDES-II implements fixed-priority timed multitasking scheduling. On the other hand, Rubus OS implements hybrid scheduling policy that combines both static cyclic scheduling and fixed-priority preemptive scheduling [21]. COMDES-II is a relatively new research project and the support for development tools and run-time environment was provided fairly recently [22]. 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].

Real-Time CORBA

Object Management Group (OMG) defined middleware technologies such as Real-Time CORBA, minimum CORBA and CORBA lightweight services for the development of real-time and distributed embedded systems [23]. In some projects, Real-Time CORBA has been used to develop distributed embedded and real-time systems [24], [25]. Because of higher resource requirements, these models may not be suitable for the development of resource-constrained distributed embedded systems with hard real-time requirements.

RCM can be considered a suitable choice for the development of resource-constrained distributed embedded systems for many reasons. For example, it can completely handle timing related information, i.e., real-time requirements, properties and constraints during all the stages of system development; It has a small run-time footprint (timing and memory overhead); it implements the state-of-the-art research results; It has a strong support for development and analysis tools, etc.

IV. Support for Modeling of Legacy Communication

In an ideal scenario, it should be possible to automatically generate the communication from the design model for each distributed embedded application. However, this is often not the practice in the industry because of legacy communications and legacy systems. These systems have their own predefined rules of communication. Our goal is to introduce the support for modeling of legacy communications in RCM.

To support abstraction of the implementation of communications in a node, we propose the introduction of two special purpose modeling elements, i.e., Output Software Circuit (OSWC) and Input Software Circuit (ISWC) for each frame that is to be sent or received by a node (connected to a network) respectively. In order to represent a model of communication in a physical bus, we propose another modeling object called Network Specification (NS).

A. Network Specification (NS)

It is the model representation of a physical bus. It consists of two parts: one is protocol independent and the other is protocol dependent. The protocol-independent part defines messages and the data-elements mapped to these messages. Moreover, it describes message properties, i.e., a message ID, a unique sender node ID, a list of receiver nodes IDs and an ordered set of RCM signals. A signal in RCM has a name, data type and real-time properties.

The protocol-dependent part of NS defines the behavior semantics of each message according to the protocol used for network communication. It is specific to each protocol, e.g., it will be different for CANopen [26], Hägglunds Controller Area Network (HCAN) [27], MilCAN (CAN for Military Land Systems domain) [28], Flexray, etc. The protocol-dependent part of NS contains complete information of all the frames which are sent on the bus. Moreover, it describes the frame properties. In RCM, a frame is a collection of RCM signals.

The frame properties described by the protocol-dependent part of NS (e.g., for a CANopen protocol) include an identifier (a reference to the corresponding message in the protocol-independent part of NS), a priority, a transmission type (type of message transmission in CANopen), a sender node ID, a list of receiver nodes IDs, whether a frame is an IN frame or an OUT frame, a period (period with which a message is sent in case of periodic transmission), an inhibit time (minimum time between successive transmission of a message in case of one of the asynchronous transmission types in CANopen), SYNC period (time between SYNC messages sent by the CANopen SYNC master), and real-time requirements. Moreover, it also specifies the speed of CAN bus. The transmission type of a frame can be periodic, event or mixed (transmitted periodically as well as on arrival of an event) [29].

The components inside a single node communicate with each other by using data and control signals separately. However, if a component on one node intends to communicate with a component on another node via a network then the signals are packed into frames. These frames are then transmitted over the network. Here, some questions arise regarding the network communication. How are signals packed into the frames? How many signals a message contains? How are signals encoded into the frames at the sender node? How are signals decoded from the received frames and sent to the respective SWCs at the receiver node? All the rules that are concerned with the answers to these questions are specified in the Signal Mapping. The Signal Mapping is unique for each network communication protocol and is defined by the protocol-dependent part of NS.

B. Output Software Circuit (OSWC)

OSWC is the model representation of signals in an outgoing message (frame) to the network. Basically, it is a Software Circuit which denotes the data that leaves the model. An OSWC is associated with a LAN (Local Area Network) object. In RCM, LAN is an object to represent a connection between two or more nodes in a system. Formally, a LAN is defined by its name, a list of connected nodes and a Network Specification. 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. A frame contains 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 the Network Specification. 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 out-ports. The OSWC component uses protocol-specific rules, specified in the protocol-specific part of NS, while encoding data and mapping signals to a frame. 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. The OSWC component is graphically illustrated in Fig. 2.

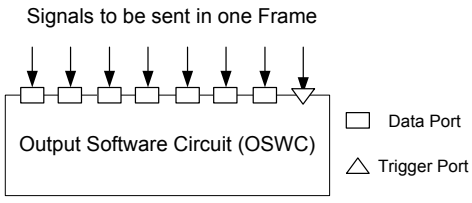

Figure 2. Graphical illustration of OSWC.

C. Input Software Circuit (ISWC)

ISWC is the model representation of signals in an incoming message (frame) from the network. Basically, 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. An ISWC describes all the signals that are contained in a received frame associated to it. 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 the Network Specification 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 is triggered when a frame arrives from the network. When a frame arrives at a node, the physical bus drivers and protocol-specific implementation of ISWC 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 the data in-port of the destination SWC (the tracing information is provided in NS), and the corresponding trigger port is triggered. Fig. 3 graphically illustrates ISWC.

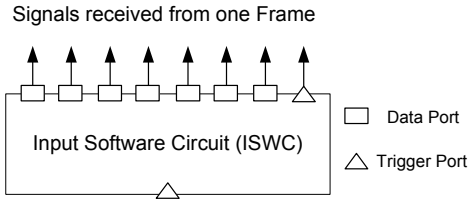

Figure 3. Graphical illustration of ISWC.

Example

Consider an example of a node in a distributed embedded application modeled with newly introduced objects in RCM as shown in Fig. 4. The network protocol considered in this example is CAN. 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 boxes are specific for each network protocol. The frames that leave the model (sent to CAN SEND) are denoted by S (Send), e.g., $S1$, $S2$ and $S3$. Similarly all the frames that enter the model (received from CAN RECEIVE) are denoted by R (Receive), e.g., $R1$ and $R2$ as shown in Fig. 4.

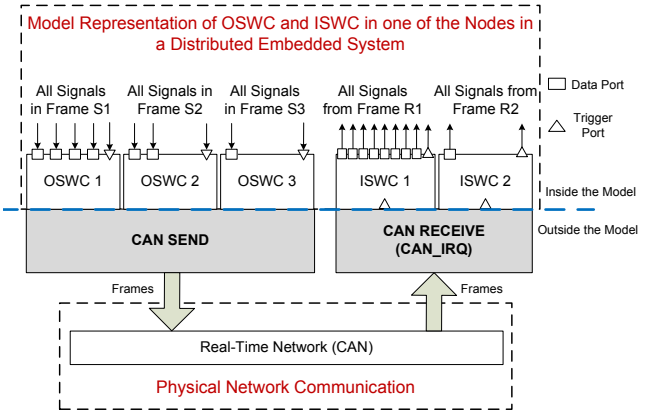

Figure 4. Model of OSWCs and ISWCs in one of the nodes in a distributed embedded system modeled with extended RCM.

All the signals sent in the 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 the Network Specification. 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 on the network according to the communication rules of CAN protocol.

When a frame arrives at the receiving node, it is transferred by the physical network drivers to a grey box (CAN RECEIVE in this example) that produces an interrupt. The frame enters the model and is transferred to the destination ISWC (the tracing information is provided in the Network Specification). ISWC extracts the signals from the frame, decodes the data from the frame and encodes it to RCM data type. The data is placed on the data out-port of ISWC which is 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 Network Specification).

D. Automatic Generation of OSWC and ISWC

Both OSWC and ISWC 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 the NS. The output of this tool is a set of automatically generated OSWCs and ISWCs for each node in the network. This tool also carries out mapping from NS to OSWC and ISWC and vice versa. The Input and Output Software Circuits are translated into a set of SWCs to execute the protocol at run-time.

Let us briefly compare our newly introduced modeling approach for network communication with the existing

modeling approach for intra-node communication in RCM (depicted in Fig. 1 by means of connectors). An alternative to the new modeling approach (presented in this paper) would have been to use the same connectors for modeling both inter-node and intra-node communication by attaching a boolean specifier to it, say, 0 for intra-node and 1 for inter-node communication; and some tool could automagically generate the run-time architecture for all communications and perform the deployment of the distributed embedded application. Similarly, another alternative modeling approach is to have an allocation property on each SWC, describing which node it will run on. However, these modeling approaches 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 early analysis of the developed system.

Analyzability was one important aspect that was kept in mind while introducing new components in RCM. The objective was to enable RCM to not only model the legacy communication but also to analyze the end-to-end timing behavior of the modeled system. In the next Section, we will discuss how the required timing information is extracted from a distributed embedded system, modeled with RCM, to perform an end-to-end timing analysis.

V. Implementation of End-to-End Timing Analysis in Rubus-ICE

In real-time systems, the time at which the result is available is as important as correct value of the result. With the newly introduced modeling elements in RCM, we can model a complete distributed real-time embedded system. In order to ensure that all timing requirements are met, the modeled system should render itself to an end-to-end timing analysis. To perform the timing analysis, an end-to-end timing model of the application should be available. In this Section, we first present the end-to-end timing model used by the Rubus Analysis Framework. Then we demonstrate, by an example, the extraction of the end-to-end timing model. Finally, we describe the support for the end-to-end timing analysis available in Rubus-ICE.

A. System Model for End-to-end Timing Analysis

The scheduling model that we implemented in the Rubus Analysis Framework, to carry out the end-to-end timing analysis, consists of two state-of-the-art scheduling models, i.e., a node analysis model and a network analysis model. The node analysis model was previously implemented in the the analysis framework of Rubus-ICE [7]. The node analysis model is a task model with offsets that is adapted from the scheduling model for the holistic response-time analysis developed by [30] and later on, extended by many researchers, e.g., [31], [32]. This model is used for the response-time analysis of tasks in a node. The network analysis model that we implemented in Rubus-ICE is a communication model [33] which is used for the response-time analysis of CAN messages. The task model and the communication model together comprise the end-to-end timing model of a distributed real-time and embedded system.

*1) **Node Analysis Model**:* The system (node), Γ, consists of a set of k transactions $\Gamma_1, \ldots, \Gamma_k$. Each transaction Γ_i is activated by a periodic sequence of events with a period T_i. In case of sporadic events, T_i denotes the minimum inter-arrival time between two consecutive events. In this model we consider that the activating events are mutually independent, i.e., the phasing between them is arbitrary. There are $|\Gamma_i|$ tasks in a transaction Γ_i and each task may not be activated until a certain time, called an *offset*, elapses after the arrival of the external event. By task activation we mean that the task is released for execution. A task is denoted by τ_{ij}. The first subscript, *i*, specifies the transaction to which this task belongs and the second subscript, *j*, denotes the number of the task within the transaction. A task, τ_{ij}, is defined by the following attributes.

- C_{ij}: It is the worst case execution time of the task.
- O_{ij}: It is an offset of the task.
- D_{ij}: It is the deadline of the task.
- J_{ij}: It is the maximum release jitter.
- B_{ij}: It represents the maximum blocking of the task from lower priority tasks.
- P_{ij}: It represents the priority of the task.
- R_{ij}: It represents the worst-case response time of the task.

In this task model, there are no restrictions placed on offset, deadline or jitter, i.e., they can each be either smaller or greater than the period.

*2) **Network Analysis Model**:* The system (network) consists of a number of nodes that are connected through a CAN bus. If a task on one node intends to communicate with a task on another node, it queues a message in the send queue of its node. The CAN protocol ensures arbitration and transmission of all messages over the bus. There are four different types of CAN frames used for message transmission, i.e., Data frame, Remote Transmit Request (RTR) frame, Overload frame and Error frame. In this model, a message corresponds to a message that uses Data or RTR frames for transmission. Each message m has the following attributes.

- ID_m: It is a unique identifier.
- $FRAME\_TYPE$: It specifies whether the frame is a Standard or an Extended CAN frame.
- $TRANSMISSION\_TYPE$: It specifies whether the frame is periodic or event or mixed (both periodic and event).
- P_m: It is a unique priority.
- C_m: It specifies the transmission time of the message.
- J_m: It is a release jitter that is inherited from the response time of the task queueing the message.

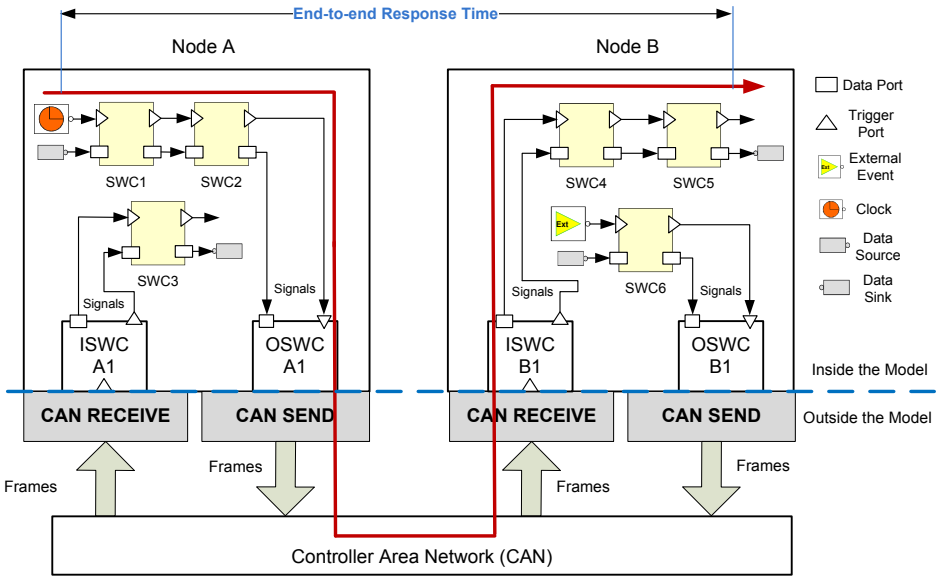

Figure 5. Example distributed system modeled with extended RCM

- DLC_m: Each message can carry a data payload that ranges from 0 to 8 bytes. This number is specified in the header field of the frame called Data Length Code.
- T_m: It specifies the period of a message in case of periodic transmission. In case of an event transmission, T_m refers to the minimum time that should elapse between the transmission of any two messages.
- B_m: Each message has a blocking time which refers to the maximum amount of time this message can be blocked by the lower priority messages.
- R_m: It denotes the worst-case response time of a message m.

B. Extraction of End-to-End Timing Model

We extract an end-to-end timing model from the distributed transactions modeled with extended RCM. The extracted model is used to analyze the end-to-end timing for delays and network utilization. Consider the following example.

Example

An example distributed embedded system modeled with RCM using the new modeling objects is shown in Fig. 5. There are two nodes in the system with three SWCs per node. SWCs communicate with each other by using both inter-node and intra-node communication. For inter-node communication, CAN or any high level protocol of CAN (e.g., CANopen, HCAN, MilCAN, etc.) can be used. In this example, the nodes are connected to a CAN network. An event chain (distributed transaction) that consists of four Software Circuits, i.e., SWC1, SWC2, SWC4 and SWC5 is identified with bold lines in Fig. 5. In this transaction, a clock triggers SWC1 which in turn triggers SWC2. SWC2 then sends a signal to the OSWC A1 which in turn maps it to a CAN frame. It then sends the frame to the grey box CAN SEND and hence, the data leaves the model. The frame is transmitted over the CAN bus by the CAN controller according to the communication rules of CAN protocol.

When the frame is received at Node B, the grey box CAN RECEIVE raises an interrupt and passes the frame to ISWC B1 and hence, the data enters the model. It should be noted that there can be more than one ISWCs in a node. In that case, a received frame is passed to the desired ISWC by looking at the tracing information in the NS. The ISWC $B1$ decodes the received frame, extracts signal from the frame, places the data on the corresponding data port and triggers the corresponding trigger port. The elapsed time between the arrival of a triggering event (clock trigger) at the input of SWC1 and the response of SWC5 is referred to as an end-to-end response time of the distributed transaction and is indicated in Fig. 5.

The end-to-end timing model, used by the Rubus Analysis Framework, requires the timing related information of all the transactions and messages as discussed in the above subsection. The required timing information about all the transactions in the system is extracted from the compiled and verified design representation of the modeled systems provided in the form of node configuration files in Rubus-ICE. At network level, the timing information is specified in the Signal Mapping that is defined in NS. This information is extracted from the configuration specification files corresponding to NS.

C. Support for End-to-End Timing Analysis

In Rubus-ICE, when the designed model is completed it is compiled to the Intermediate Compiled Component Model (ICCM) file [7]. All the timing information required by the end-to-end timing model is extracted from the ICCM file. From this timing model, the Rubus Analysis Framework performs the response-time analysis of individual tasks [32], the response-time analysis of messages on the network [33], [34] and the end-to-end timing analysis [35]. The analysis framework provides the results, i.e., response times of individual tasks, response times of network messages, end-to-end response times of event chains (distributed transactions), network utilization, etc., back to Rubus-ICE. This whole process is depicted in Fig. 6.

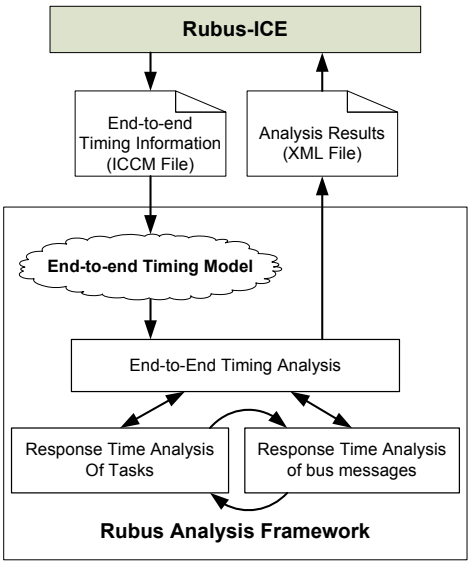

Figure 6. Extraction of end-to-end timing model for timing analysis in Rubus-ICE

VI. Conclusion

We introduced new components to the industrial component model, the Rubus Component Model (RCM), for the development of distributed embedded systems. The purpose of new modeling objects, i.e., Output Software Circuit (OSWC), Input Software Circuit (ISWC) and Network Specification (NS) is to abstract the implementation and configuration of communications in distributed embedded systems. These objects make the communication capabilities of a node very explicit, but efficiently hide the implementation or protocol details. The extended model allows model-based and component-based development of new nodes that are deployed in legacy (previously developed) systems that use predefined communication rules. While making the design decisions about new components, one important objective was to enhance analyzability in the component model. Here, the focus was the ease to extract the end-to-end timing model from a distributed application modeled with RCM. The extracted model is used by the Rubus Analysis Framework to perform the end-to-end timing analysis.

With the addition of new modeling capabilities, RCM can be considered a suitable choice for the industrial development of resource-constrained distributed embedded systems with hard real-time requirements. There are a number of reasons behind this motivation, e.g., it can model real-time communication (both intra-node and inter-node); it can completely handle timing related information (real-time requirements, properties and constraints) during all the stages of system development; it has a small run-time footprint (timing and memory overhead); it implements the state-of-the-art research results; it has a strong support for development and analysis tools, etc.

In the future work, the implementation of OSWC and ISWC will be automatically generated from protocol configuration files of other specialized communication protocols used for real-time network communication such as CANopen, HCAN (Hägglunds Controller Area Network), J1939, etc. For example, the next step will be to generate automatically the implementation of OSWC and ISWC from DCFs (Device Configuration Files) in CANopen or for subsets of J1939.

Acknowledgement

This work is supported by Swedish Knowledge Foundation (KKS) within the project EEMDEF, the Swedish Research Council (VR) within project TiPCES, and the Strategic Research Foundation (SSF) with the centre PROGRESS. The authors would like to thank the industrial partners Arcticus Systems and BAE Systems Hägglunds for the cooperation.

References

[1] M. Broy, "Automotive software and systems engineering," in *Formal Methods and Models for Co-Design, 2005. MEMOCODE '05. Proceedings. Third ACM and IEEE International Conference on*, 2005, pp. 143 – 149.

[2] R. N. Charette, "This Car Runs on Code," *Spectrum, IEEE*, vol. 46, no. 2, 2009, http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code.

[3] T. A. Henzinger and J. Sifakis, "The Embedded Systems Design Challenge," in *Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*. Springer, 2006, pp. 1–15.

[4] I. Crnkovic and M. Larsson, *Building Reliable Component-Based Software Systems*. Norwood, MA, USA: Artech House, Inc., 2002.

[5] K. Hänninen, J. Mäki-Turja, M. Nolin, M. Lindberg, J. Lundbäck, and K.-L. Lundbäck, "The Rubus Component Model for Resource Constrained Real-Time Systems," in *3rd IEEE International Symposium on Industrial Embedded Systems*, June 2008.

[6] "Arcticus Systems," http://www.arcticus-systems.com.

[7] K. Hänninen, J. Mäki-Turja, S. Sandberg, J. Lundbäck, M. Lindberg, M. Nolin, and K.-L. Lundbäck, "Framework for real-time analysis in Rubus-ICE," in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, 2008, pp. 782 –788.

[8] "Microsoft, Distributed Component Object Model (DCOM)," http://msdn.microsoft.com/en-us/library/Aa286561.

[9] "OMG, Common Object Request Broker Architecture (CORBA) , Version 3.1," January 2008. [Online]. Available: http://www.omg.org/spec/CORBA/3.1

[10] L. DeMichiel, "Sun Microsystems, Enterprise JavaBeans Specification, Version 2.1," *Sun Microsystems*, 2002.

[11] "AUTOSAR Techincal Overview, Version 2.2.2," in *AUTOSAR – AUTomotive Open System ARchitecture, Release 3.1*. The AUTOSAR Consortium, August 2008. [Online]. Available: http://autosar.org

[12] H. Heinecke et al., "AUTOSAR – Current results and preparations for exploitation," in *Proceedings of the 7th Euroforum Conference*, ser. EUROFORUM '06, May 2006.

[13] "TIMMO Methodology , Version 2," *TIMMO (TIMing MOdel), Deliverable 7*, October 2009, The TIMMO Consortium.

[14] "TADL: Timing Augmented Description Language, Version 2," *TIMMO (TIMing MOdel), Deliverable 6*, October 2009, The TIMMO Consortium.

[15] "The UML Profile for MARTE: Modeling and Analysis of Real-Time and Embedded Systems," January 2010. [Online]. Available: http://www.omgmarte.org/

[16] "EAST-ADL Domain Model Specification, Deliverable D4.1.1," http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf.

[17] "TIMMO-2-USE," http://www.itea2.org/projects/step/2.

[18] S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic, "A Component Model for Control-Intensive Distributed Embedded Systems," in *Proceedings of the 11th International Symposium on Component Based Software Engineering (CBSE2008)*, October 2008, pp. 310–317.

[19] J. Carlson, J. Feljan, J. Mäki-Turja, and M. Sjödin, "Deployment Modelling and Synthesis in a Component Model for Distributed Embedded Systems," in *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*, 2010, pp. 74 –82.

[20] X. Ke, K. Sierszecki, and C. Angelov, "COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems," in *Embedded and Real-Time Computing Systems and Applications, RTCSA 2007. 13th IEEE International Conference on*, 2007, pp. 199 –208.

[21] J. Mäki-Turja, K. Hänninen, and M. Nolin, "Efficient Development of Real-Time Systems Using Hybrid Scheduling," in *9th Real-Time in Sweden (RTiS'07)*, August 2007, pp. 157–163.

[22] Y. Guo, K. Sierszecki, and C. Angelov, "COMDES Development Toolset," in *5th International Workshop on Formal Aspects of Component Software FACS 08, Malaga, Spain*, 2008.

[23] "Catalog of Specialized CORBA Specifications. OMG Group," http://www.omg.org/technology/documents/.

[24] S. Lankes, A. Jabs, and T. Bernmerl, "Integration of a CAN-based connection-oriented communication model into Real-Time CORBA," in *Parallel and Distributed Processing Symposium*, 2003.

[25] R. Finocchiaro, S. Lankes, and A. Jabs, "Design of a real-time CORBA event service customised for the CAN bus," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004, p. 121.

[26] "CANopen high-level protocol for CAN-bus, Version 3.0," *NIKHEF, Amsterdam*, March 2000.

[27] J. Westerlund, "Hägglunds Controller Area Network (HCAN), Network Implementation Specification," *BAE Systems Hägglunds, Sweden*, April 2009.

[28] "MilCAN (CAN for Military Land Systems domain)," http://www.milcan.org/.

[29] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Extending Schedulability Analysis of Controller Area Network for Mixed (Periodic/Sporadic) Messages," in *(to appear) the 16th IEEE Conference on Emerging Technologies and Factory Automation (ETFA), 2011*, sept. 2011.

[30] K. Tindell, "Adding Time-Offsets to Schedulability Analysis," Department of Computer Science, University of York, England, Tech. Rep., January 1994.

[31] J. Palencia and M. G. Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets," *Real-Time Systems Symposium, IEEE International*, p. 26, 1998.

[32] J. Mäki-Turja and M. Nolin, "Efficient implementation of tight response-times for tasks with offsets," *Real-Time Syst.*, vol. 40, no. 1, pp. 77–116, 2008.

[33] K. Tindell, H. Hansson, and A. Wellings, "Analysing real-time communications: controller area network (CAN)," in *Real-Time Systems Symposium (RTSS) 1994*, pp. 259 –263.

[34] R. Davis, A. Burns, R. Bril, and J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, pp. 239–272, 2007.

[35] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, vol. 40, pp. 117–134, April 1994. [Online]. Available: http://dx.doi.org/10.1016/0165-6074(94)90080-9