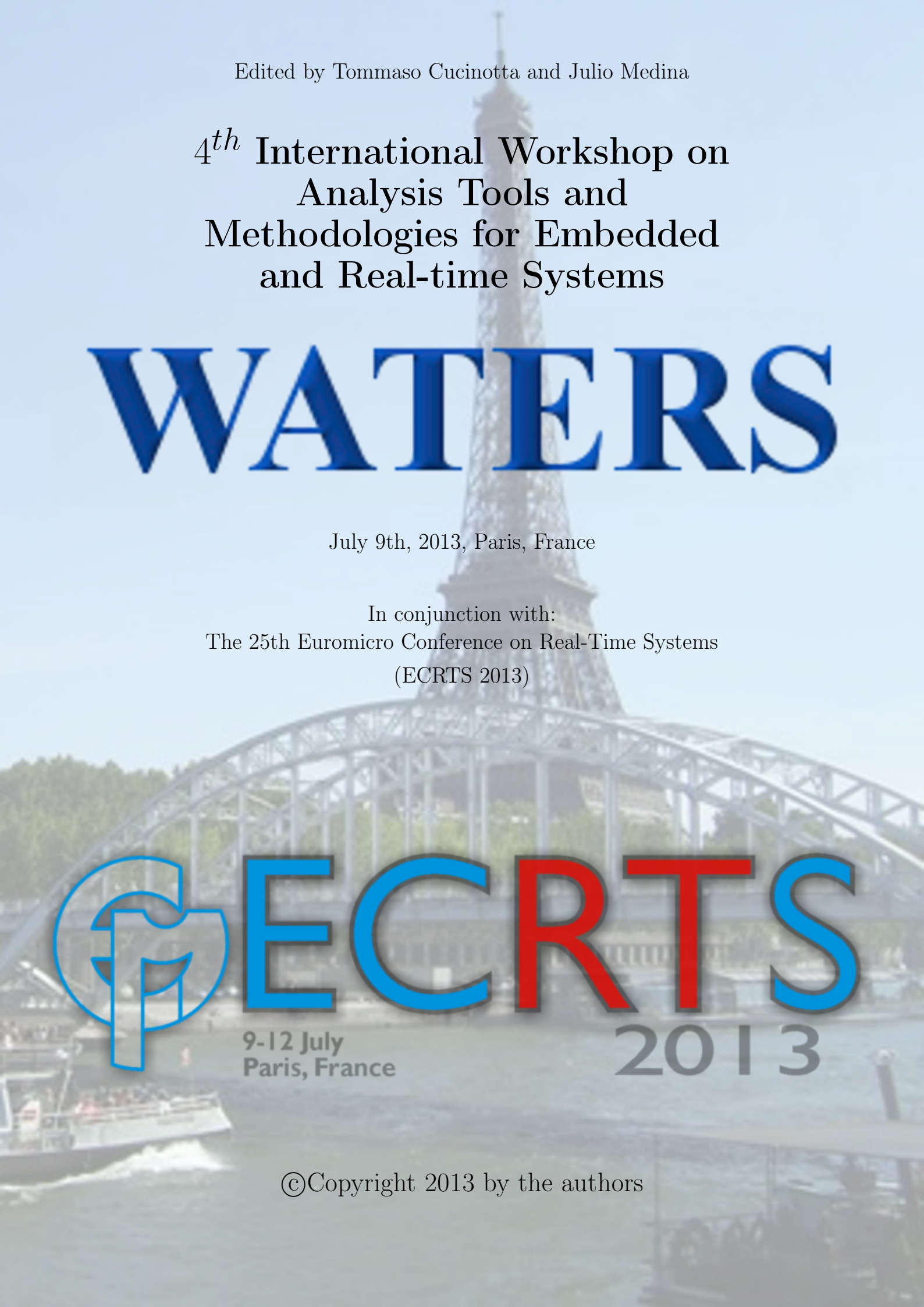
Edited by Tommaso Cucinotta and Julio Medina

4^{th} International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems

WATERS

July 9th, 2013, Paris, France

In conjunction with:
The 25th Euromicro Conference on Real-Time Systems
(ECRTS 2013)



This page has been left intentionally blank.

Edited by Tommaso Cucinotta and Julio Medina

4^{th} International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems

July 9th, 2013, Paris, France

In conjunction with:
The 25th Euromicro Conference on Real-Time Systems
(ECRTS 2013)

This page has been left intentionally blank.

Table of Contents

Message from the Program Chairs

Real-Time schedulability analysis techniques, scheduling algorithms, and synchronisation protocols, have greatly evolved and continue to evolve in our research community along the last 40 years. New hardware platforms and the increasing demand from different application domains, are posing more and more challenges on the predictability and reliability of applications in terms of their timing behaviour. While theory grows and goes rising in complexity, the gap between industrial practice and state-of-the-art research techniques keeps also enlarging, leading to a panoply of strategies, formalisms, abstraction techniques, standards, and business models to deal with the very basic problem: how to assure timing predictability of software, and be able to build dependable systems, applications, or products upon it.

As it happens with all computer science and IT related research areas, the adoption of theoretical research results into industrial practice follows an evolutionary pattern, in which each formalism, paradigm or language fights for its place. As for many other knowledge domains, the effective means for comparing, evaluating and choosing among the available techniques relies on the scientific method; this requires experimentation and practice. Hence, software tools, methodologies, comprehensive use cases data sets, and benchmarks are the effective arms to find a place in this battle against complexity.

Besides, different authors use different algorithms for generating random task sets, different application traces when simulating dynamic real-time systems, different simulation engines when simulating scheduling algorithms. Instead, research in the field of real-time and embedded systems would greatly benefit from the availability of well-engineered, possibly open tools, simulation frameworks and data sets which may constitute a common metrics for evaluating simulation or experimental results in the area. It would be really beneficial to have a possibly wide set of reusable data sets or behavioural models coming from realistic industrial use-cases over which to evaluate the performance of novel algorithms. Availability of such items would increase the possibility to compare novel techniques in dealing with problems already tackled by others from the multifaceted viewpoints of effectiveness, overhead, performance, applicability, etc.

The initial goal of this workshop was to bring visibility and recognise the work of those scientists who write software that is useful for our community; to make these tools more widely known in it; and to create a group of researchers interested in contributing with new software and tools.

In this fourth edition, we have enlarged the spectrum of software tools, OS services and networking resources to domains and information technologies in which soft real-time requirements are dominant. As in former editions, we also ask authors to provide their software (or links to their web page where the software is made available). All this information is available through the workshop web page[1]. Also, our mailing list in Google Group is active to distribute information on the workshop themes[2].

[1] `http://www.ctr.unican.es/waters2013/`

[2] `https://groups.google.com/forum/?fromgroups#!forum/ecrts-waters`

We would like to thank the Euromicro organisation for having allowed us to organise this event, and particularly Laurent George, Gerhard Fohler and Stefan M. Petters for their prompt and ready support. We would like to thank the Real-Time Systems Laboratory of Scuola Superiore Sant'Anna and the Computers and Real-Time Group of the University of Cantabria for their support in hosting the WATERS website. Also, we would like to thank all the authors for having submitted their work to the workshop for selection, the Program Committee members for their effort in reviewing the papers, the presenters for ensuring interesting sessions, and the attendees for participating into this event.

We are sure that interesting ideas and discussions will come out of the presentations, demos and the questions that will alternate along the day. We hope you to find this day interesting and enjoyable and your overall experience with WATERS to bring you back for more our next edition.

The WATERS 2013 Chairs

Tommaso Cucinotta[3] and Julio Medina[4]

[3]Tommaso Cucinotta is with Bell Laboratories, Alcatel-Lucent, Dublin, Ireland
e-mail: tommaso.cucinotta@alcatel-lucent.com

[4]Julio Medina is with Departamento de Electrónica y Computadores Universidad de Cantabria, Santander, Spain
e-mail: julio.medina@unican.es

Program Committee

- Luca Abeni (University of Trento, Italy)
- Ian Broster (Rapita Systems Ltd, York, UK)
- Laura Carnevali (University of Florence, Italy)
- Marisol García Valls (Universidad Carlos III de Madrid, Spain)
- Laurent George (INRIA Paris-Rocquencourt, France)
- Michael Gonzalez (Universidad de Cantabria, Spain)
- Kwei-Jay Lin (University of California, Irvine, US)
- Giuseppe Lipari (École Normale Supèrieure de Cachan, France)
- Thomas Nolte (Mälardalen University, Sweden)
- Martin Quinson (Nancy University, France)
- Simon Schliecker (Symtavision GmbH, Braunschweig, Germany)
- Thomas C. Schmidt (Hamburg University of Applied Sciences, Germany)
- Wang Yi (Uppsala University, Sweden)

Papyrus: an eclipse-based open source initiative for exploiting the full modeling power of UML and DSML's

Camille LETAVERNIER and Florian NOYRIT
CEA LIST/DILS/ Laboratory of model driven engineering for embedded systems (LISE)

The principle of separating concerns is widely used in engineering to address complexity. In Model-Driven Engineering (MDE), this principle notably led to the development of Domain Specific Modeling Languages (DSML). Those languages provide constructs that are directly aligned with the concepts of the domain in question. A specific domain, in the broad sense, can be an application domain (e.g. auto-motive) or a specific concern (e.g. requirement modeling). This coincides nicely with the vision specified in the ISO/IEC/IEEE 42010 standard which suggests that each stakeholder needs dedicated viewpoints to address his or her concerns. It is the responsibility of language designers and methodologists to provide the DSMLs that support these viewpoints. And, it is the responsibility of tool designers to provide appropriate user interfaces to corresponding language authoring tools.

This is precisely what Papyrus project provides: a modeling environment that let language designers define their DSML for a specific viewpoint and let tool designers configure the modeling environment to support the desired viewpoint. However, Papyrus provides those features under the following motto: “Don’t reinvent the wheel, adopt UML instead but tailor it to your needs”. To do that, Papyrus provides a UML profile editor to let language designer define the DSML. But, more important, Papyrus provides means to tool designers to customize the modeling environment in order to tailor the UI to corresponding viewpoint. Papyrus is therefore a generic modeling environment that can be customized to support UML-based DSML.

This talk presents the key concepts and features that are available today in Papyrus to define and implement UML-based DSMLs. It will especially show how UML-based DSML can be much more than just an extension of the UML metamodel. We will also introduce features that are coming soon together with those we plan to develop in the long term.

A Simulation Environment based on OMNeT++ for Automotive CAN–Ethernet Networks

Jun Matsumura, Yutaka Matsubara, Hiroaki Takada
Graduate School of Information Science
Nagoya University, Aichi, Japan

Masaya Oi, Masumi Toyoshima, Akihito Iwai
DENSO CORPORATION
Aichi, Japan

***Abstract*—Due to the rapid increase in the functionality requirements of automotive control networks, mixed CAN (Controller Area Network) and Ethernet networks have recently gained considerable attention. This paper presents a simulation environment for CAN–Ethernet networks based on the open-source network simulator OMNeT++, part of the INET framework. We develop simulation models of CAN and a CAN–Ethernet Gateway (GW). To validate the CAN model, we measure the end-to-end latency of CAN messages and compare its performance with an existing CAN network simulator. We apply the proposed simulation environment to an automotive CAN–Ethernet system, and confirm that it is an effective aid in the design and evaluation of such networks.**

I. Introduction

Controller Area Network (CAN), originally developed by Bosch in 1985, has long been used as the standard protocol for automotive control networks. In recent years, because the functionality demands of such systems have grown rapidly, it has become increasingly difficult to meet network requirements using only the CAN and Local Interconnect Network (LIN) protocols. Moreover, some new functions using Ethernet have been considered, such as Diagnostics on Internet Protocol (DoIP) and reprogramming the firmware of Electronic Control Units (ECUs). Ethernet offers the twin advantages of high communication speed and a flexible network topology. However, its lack of a real-time property means that it has not, to date, been used in a control network. To overcome these problems, there has been some research into mixed CAN–Ethernet networks [4], and some CAN–Ethernet gateway (GW) products have been released [6], [7].

In order to integrate CAN–Ethernet networks early in the development process, we need to explore an appropriate design that can meet certain stated requirements, because existing CANs cannot be reused. The requirements include the end-to-end latency of messages, the number of GWs, and the amount of buffering to be implemented in each GW. The evaluation of CAN–Ethernet networks can be conducted by analytical or simulation methods. The worst-case response time has been considered as an analytical method in CAN networks [3], [12] and Ethernet networks [8], [9]. In addition, the Symta/S analytical tool supports CAN–Ethernet networks [13].

Once we obtain an analytical method for a targeted network, it can be evaluated quickly. However, the communication protocol and network topology involved in developing such analytical methods for complex message sets is not straightforward. Therefore, simulation methods are more effective for designing and evaluating the early states of a network.

To date, many network simulators have been developed, such as CANoe [11], Venet [14], OPNET [2], TrueTime [1], and OMNeT++ [5]. Provided by Vector Inc., CANoe is a well-known development tool for the design of automotive control networks, and supports many network protocols, such as CAN, LIN, MOST, FlexRay, and Ethernet. OPNET (OPNET Technologies Inc.) is a bit-level network simulator; such tools give more accurate simulations than message-level simulators. However, their simulation speed is generally slow. In addition, as OPNET is a commercial tool, we cannot easily add a simulation model of a new network protocol. Therefore, to evaluate the early states of a network, message-level simulators are more suitable than bit-level simulators. Venet, developed by InterDesign Technologies Inc., is a message-level simulator for CANs, but does not support the Ethernet protocol. TrueTime, distributed by Lund University and based on MATLAB/Simulink, is a message-level simulator that does support Ethernet, as well as MAC and CSMA/CD. However, IP and TCP/UDP are not supported. The open-source discrete time event simulator OMNeT++ is well-documented for both users and model developers. This means that we can easily develop a new simulation model. Although OMNeT++ supports many multimedia communication protocols, no automotive network protocols are currently supported.

In this paper, we propose a simulation environment for CAN–Ethernet networks based on OMNeT++ and the INET framework [10]. We develop a CAN simulation model and a CAN–Ethernet GW model. To validate the developed CAN model, we compare the end-to-end latency of CAN messages using the proposed simulator with results from the Venet tool. Furthermore, we perform a case study for an automotive CAN–Ethernet network and confirm the applicability of the proposed simulator.

This paper is organized as follows: Section 2 presents an overview of the proposed simulation environment, and Section 3 describes in detail the simulation models for CAN and CAN–Ethernet GW. Section 4 gives the results of our evaluation of the developed CAN model and a case study. Finally, Section 5 concludes the paper.

II. Simulation Environment

A. Overview

Fig. 1 shows an overview of the proposed simulation environment, with input files on the left, the simulator in thecenter, and the output files on the right. In the following sections, we describe the necessary input files, an overview of

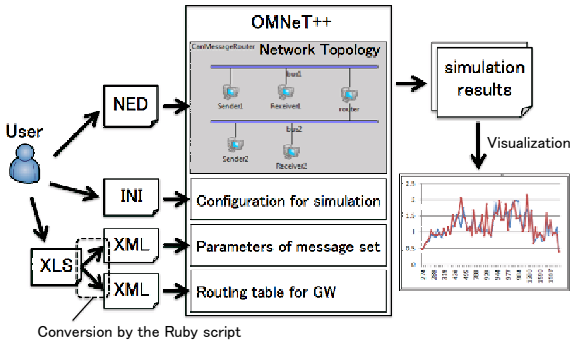


Fig. 1. Overview of the proposed simulation environment

```
<NodeInfo>
   <Node ID="ECU73">
      <SendMessage SendTime="0" Offset="50"
       ID="7cf" DLC="8" SendInterval="10"/>
      <SendMessage SendTime="100" Offset="10"
       ID="8df" DLC="8" SendInterval="0"/>
      ...
   </Node>
</NodeInfo>
```

Fig. 2. Example of a definition file for a message set

the simulation environment, and how to obtain and analyze the simulation results.

B. Input Files

Users create three input files for the simulator, a NED file (.ned), an INI file (.ini), and a spreadsheet file (.xls). The network topology is described in the NED file. The INI file includes initial parameters and the configuration for the simulations, and the spreadsheet file includes the message sets of each node. The spreadsheet file is described in its original format for this environment. We created a script program using the Ruby language to generate two XML files from the spreadsheet file.

An example of an XML file that defines message sets sent by each node is shown in Fig. 2. A `<Node>` element represents a node in the network and includes multiple `<SendMessage>` elements. In this example, the node with ID ECU73 sends two messages. A `<SendMessage>` element defines the message sent by the node in terms of five attributes, `ID`, `Offset`, `DLC`, `SendTime`, and `SendInterval`. `ID` is simply an identifier for the message. `Offset` denotes the relative time between the simulation start time and the first time at which the message was sent. `DLC` represents the data length of the message, and `SendTime` denotes the absolute sending time of the message. `SendInterval` represents the sending period of the message. In this example, as the message with ID 0x7cf is cyclic, `SendInterval` is set to a sending period of 10. For cyclic messages, `SendTime` must be 0. The message with ID 0x8df is an event-driven message and is sent at 100.

The other XML file defines routing maps for each GW. An example of a routing map is shown in Fig. 3. A `<MessageInfo>` element represents a routing configuration for each message. The `<RoutingInfo>` has two attributes,

```
<RoutingMap ID="1">
   <MessageInfo ID="686">
      <RoutingInfo Name="Ethernet" Send="0"/>
      <RoutingInfo Name="CAN1" Send="0"/>
      <RoutingInfo Name="CAN2" Send="1"/>
      <RoutingInfo Name="CAN3" Send="1"/>
      <RoutingInfo Name="CAN4" Send="0"/>
      <RoutingInfo Name="CGW" Send="0"/>
   </MessageInfo>
   ...
</RoutingMap>
```

Fig. 3. Example of a definition file for a routing map

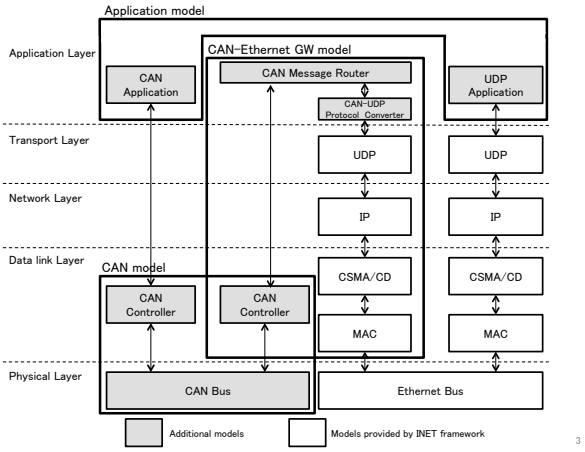


Fig. 4. Protocol structure

`Name` and `Send`. `Name` represents the name of a bus, and `Send` determines whether the message should be transferred in binary for each bus connected to the GW. The example shows that, when the GW receives message 0x686, it is only transferred to the CAN2 bus and the CAN3 bus.

C. Structure of the Simulator

The protocol structure of the proposed simulator is shown in Fig. 4. The simulator is based on OMNeT++ version 4.2.2 and INET framework version 1.99.4, as well as three simulation models, namely a CAN model, a GW model, and one of two application models. The CAN model consists of a CAN bus model and a CAN controller model to simulate the CAN protocol, and the GW model relays messages received from one bus to other buses. A CAN–CAN GW model includes only a CAN message router, whereas a CAN–Ethernet GW model includes a CAN–UDP Protocol Converter, which converts CAN messages to UDP packets, and vice versa, as well as routing CAN messages. The CAN–Ethernet GW is based on existing models of Ethernet bus, MAC, CSMA/CD, IP, and UDP included in the INET framework. We develop a CAN application model that sends and receives CAN messages, and a UDP application model that sends and receives UDP packets.

D. Simulation Parameters

In the proposed simulation environment, the following parameters can be obtained from the simulation results.

- End-to-end latency of each message

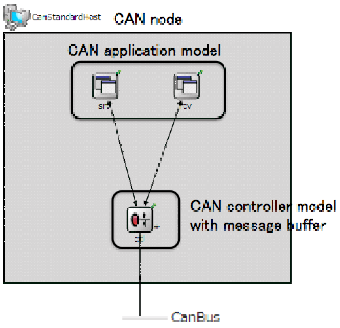


Fig. 5. Structure of CAN model

- Necessary buffer size of each GW
- Average utilization of each bus

To obtain these parameters, we implement some code using the signal mechanism provided in OMNeT++. For the end-to-end latency, signals are generated when the departure node generates a message and when the destination node receives the message. When the number of messages in the GW queue changes, another signal is generated. To obtain the average utilization parameter, the time used for each bus is calculated once per second.

III. Simulation Models

A. CAN Model

As shown in Fig. 5, the CAN model consists of a CAN controller model, a CAN bus model, and a CAN application model.

The CAN controller model stores CAN messages sent from a CAN application in a queue, and sends them one-by-one to the CAN bus. Fig. 6 shows the state transition diagram of the CAN controller model. There are two states, transmission and idle. In the idle state, when the CAN controller model receives a CAN message from a CAN application, it requests the CAN bus to send the message. The CAN controller model then transits from idle to transmission (1). When the CAN controller model receives a CAN message while in the transmission state, the CAN message is stored in the message queue (2). When the CAN controller model is in the transmission state and receives an arbitration completion message from the CAN bus, and if the message queue is empty, the CAN controller model transits to the idle state (3). If the message queue is not empty, the CAN controller model requests the CAN bus to send the next message and remains in the transmission state (4).

We implement three types of message queue in the CAN controller model. These are a FIFO queue, a partial priority-based queue, and a full priority-based queue. Because each queue type has advantages and disadvantages, the optimal type will depend on the network topology, characteristics of the application, cost of hardware, and software driver complexity. Users can specify the type of queue for each CAN controller model in the INI file. With a FIFO queue, CAN messages are sorted in first-in first-out order, regardless of the priority of the messages. The partial priority-based queue sorts CAN

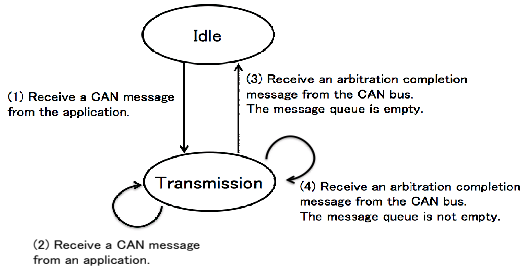


Fig. 6. State transition diagram of the CAN controller model

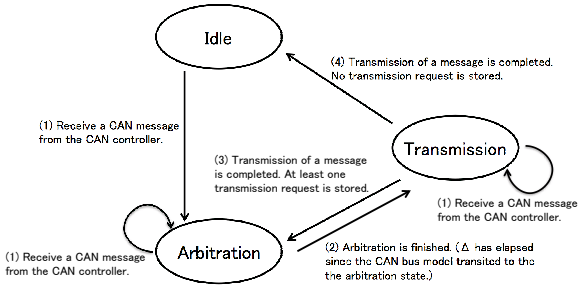


Fig. 7. State transition diagram of the CAN bus model

messages in order of priority, except for those whose send request has already been issued to the CAN bus. With a full priority-based queue, all CAN messages in the queue are sorted in order of priority. As a filtering function, when the controller model receives messages from a CAN bus, only those whose ID is specified in the INI file are relayed to applications in the upper protocol layer.

The CAN bus model handles message transmission and arbitration of conflicting send requests among the CAN nodes connected to a CAN bus. In a real CAN, message arbitration is handled by both CAN controllers and a CAN bus. In our simulation, message arbitration is achieved using only a CAN bus model. Fig. 7 shows the state transition diagram of the CAN bus model. There are three states, idle, arbitration, and transmission. When the CAN bus model receives a request to send a CAN message, it stores the sending request (1). If the CAN bus model is in the idle state, it transits to the arbitration state. When the CAN bus model receives an arbitration completion message from itself, the message with the highest priority is selected from those requesting to be sent at that time. Furthermore, an arbitration completion message is sent to the CAN controller requesting the message with the highest priority to be sent. The CAN bus model transits to the transmission state (2). When the CAN bus model receives a transmission completion message from itself, the message with the highest priority is transmitted to all CAN controllers connected to the CAN bus, except for the node that sent the message. If at least one of the send requests is stored, the CAN bus model transits to the arbitration state (3). If there are no requests, the CAN bus model transits to the idle state (4).

When using a discrete event simulator such as OMNeT++, requests to send messages from each CAN controller model to a CAN bus model are sequentially processed, although these requests may be issued at the same time. Therefore,

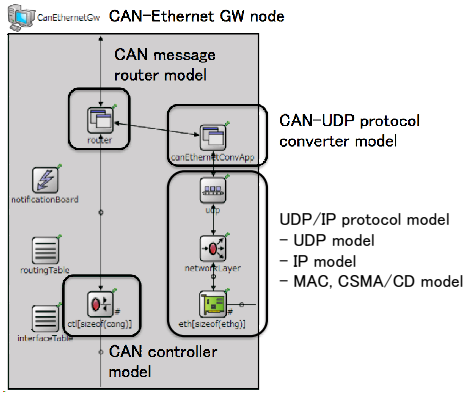


Fig. 8. Structure of CAN–Ethernet GW model

these requests do not conflict. In this case, the first message processed in the CAN bus model can be immediately sent. In our method, send requests made during the period Δ are stored in the CAN bus model to simulate the arbitration mechanism of the CAN protocol. Send requests issued within Δ are considered to give conflicts, and are handled according to the above-mentioned priority-based arbitration mechanism. The default value of Δ is 0.002 ms, corresponding to the transmission time of 1 bit of data in a 500 Kbps CAN bus.

B. Application Models

The CAN application model simulates the control application at an ECU, sending and receiving CAN messages. When the simulator is initialized, each node loads a message set to be sent from an XML file. In real automotive control systems, ECUs are not synchronized because of individual differences in their crystal oscillators. To model this situation, we introduce drift values for the deviation of the message periods of each ECU. Therefore, the sending period for a periodic message is calculated by multiplying the period specified in the XML file by a drift value specified in the INI file.

The UDP application model simulates a multimedia application at an ECU. The model generates UDP packets sent to CANs, and the data field of a UDP packet is an array of CAN messages. The ID and data of each CAN message are generated using random values. The DLC of each CAN message is 8 bytes.

C. CAN–Ethernet GW Model

As shown in Fig. 8, a CAN–Ethernet GW model has two functions, message routing among CAN buses and message conversion from CAN messages to UDP packets (and vice versa). Other types of GW model, such as CAN–TCP, CAN–FlexRay, and CAN–LIN, require only a protocol converter.

The CAN message router model manages the routing table generated at the initialization of the simulator, according to the XML file explained in Section II-B. CAN messages specified in the routing table are only transferred to buses if the value corresponding to the message is 1. If a CAN message needs to be transferred to an Ethernet bus, the message must be sent to the CAN–UDP protocol converter model.

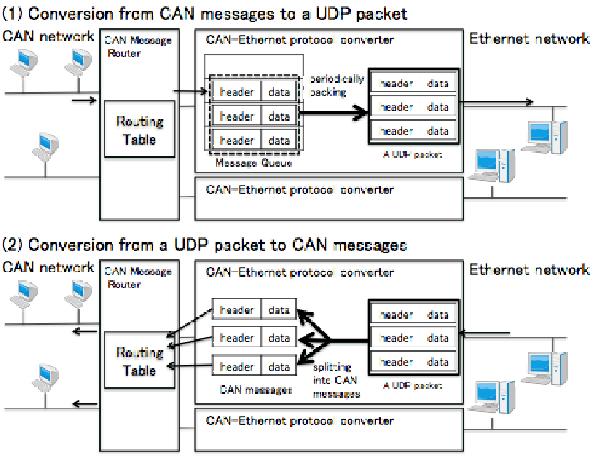


Fig. 9. Behavior of CAN–Ethernet Protocol Converter

TABLE I. CONFIGURATION FOR VALIDATION OF CAN MODEL

CAN Network Speed	500 Kbps
# of CAN nodes	14
# of CAN buses	1
# of CAN messages	65
Queue Type of CAN controller	Full priority-based queue

The CAN–UDP protocol converter model converts CAN messages to UDP packets and vice versa. Fig. 9 illustrates the behavior of the protocol converter model. In converting them to UDP packets, the CAN messages received for a given period are stored in a model buffer. Such stored messages are periodically arranged into one UDP packet and sent to an Ethernet bus (1). Users can specify the packing period (default value is 1 ms) in the INI file. Several algorithms for the conversion of CAN messages to UDP packets were proposed in [4]—which gives the best performance will depend on the specific application. In the conversion back from UDP packets, the CAN messages packed in a UDP packet are split off. Each CAN message is then sent to a CAN message router model (2). Although the internal GW delays (processing time to copy arriving messages into a buffer, check where they are to be routed, packing target messages, calling software drivers) are not considered at this time, a static delay time could easily be configured.

IV. EVALUATION

A. Validation of CAN Model

To validate the CAN model, we compared the average end-to-end latency of each CAN message using the proposed simulator and Venet. The common parameters used in the evaluation are given in Table I. All CAN messages are periodic. We performed one simulation with drift and one without. The simulation time was 60 s.

The simulation by OMNeT++ took only 13 s, which was 11.6 % faster than the simulation by Venet. With no drift, the simulation results from the proposed model correspond almost exactly to those obtained using Venet. The errors of between

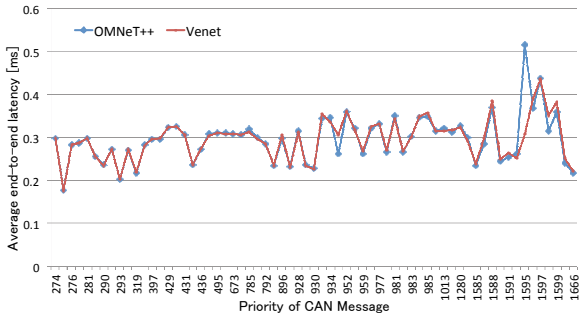


Fig. 10. Average end-to-end latency for CAN messages with drift

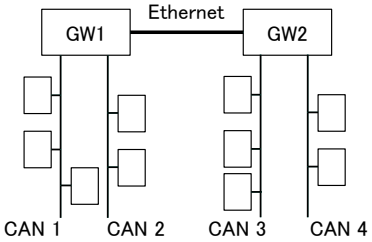


Fig. 11. Network configuration for evaluation

30–40 ns are due to rounding errors in the sending time of CAN messages. Therefore, we consider the CAN model to be valid in the case of no drift. Fig. 10 shows the results of the simulation with drift. The average end-to-end latency error between OMNeT++ and Venet is of the order of a few ms. However, this is greater than the error in the case of no drift. In particular, the error for message ID 1595 is approximately 0.2 ms. We can confirm that the rounding error in the sending time of CAN messages also caused this error. The number of disruptions caused by messages with higher priority differed between the two simulators. We consider the above results to again validate the CAN model.

B. Case Study for a CAN–Ethernet Network

We performed a case study of a CAN–Ethernet network to confirm the applicability of the proposed simulator. Table II shows the configuration of the network in this evaluation, and the network topology is shown in Fig. 11. The network consists of four CAN buses and an Ethernet bus. These are connected to CAN–Ethernet GW nodes. The parameters measured in this evaluation are the maximum end-to-end latency of each message, the maximum latency rate, the buffer usage in each CAN controller, and the utilization of each CAN bus. The maximum latency rate is defined as the ratio of the maximum end-to-end latency to the sending period of the message, and is only calculated for periodic messages. If the maximum latency rate is greater than 1, the message may not arrive at the destination node prior to the next sending time of the message.

In addition to the algorithms proposed in [4], we propose a heuristic deadline-aware algorithm. For this evaluation, we assumed that the deadline of each message corresponded to its sending period. We assumed two different maximum delay times in each CAN network, 5 ms (i.e., a constant value) and

TABLE II. CONFIGURATION OF CAN–ETHERNET NETWORK FOR CASE STUDY

CAN Network Speed	500 kbps
Ethernet Network Speed	100 Mbps
# of CAN nodes	90
# of Ethernet buses	1
# of CAN–Ethernet GWs	2
# of CAN buses	4
# of CAN messages	216 (periodic) 194 (aperiodic)
Drift of Each Node	± 1%
Queue Type of CAN controller	Full priority-based queue

TABLE III. PARAMETERS FOR CONVERSION ALGORITHMS

	Buffer size	Conversion period [ms]	Maximum shortened time [ms]	Ratio of urgent messages [%]	Assumed delay time in CAN network
Basic	1	-	-	-	-
Buffer	40	20	-	-	-
Time	40	20	20	-	-
Urgency	40	20	20	20	-
deadline (5 ms)	20	5	-	-	5ms
deadline (50% of periods)	20	5	-	-	50% of periods

50% of the period of each CAN message. In the deadline-aware algorithm, when a CAN message reaches a GW, it and the messages stored in the GW buffer are packed into the Ethernet packet if the remaining time to the deadline of the message is less than 5 ms or 50% of the period of the message. In this experiment, the value of the assumed delay time was heuristically determined, because there is no analysis method for the worst-case end-to-end delay in CAN–Ethernet networks. Therefore, we found the appropriate value by changing the parameter in 10% increments, from 10% to 90%. The simulation time was 7200 s, which corresponds to one driving cycle.

The maximum delay ratios of each algorithm are shown in Fig. 12. In the case of the deadline-aware algorithm (50% of periods), all messages met their timing constraints. From Fig. 12, we can see that the maximum delay ratio of some messages is greater than 1 in the basic, buffer, and time algorithms. This means that the deadline was missed in these simulations. For messages with high priorities (i.e., from 1 to 300), the conversion period was insufficient to meet the timing constraints. For messages with lower priorities, the sending period was shorter than for messages with similar priorities. Therefore, the influence of these messages was large.

Table IV presents the average utilization of CAN buses and the number of conversions from CAN to Ethernet. In the table, *empty* refers to the number of conversions to an Ethernet packet without any CAN messages, and *not empty* refers to the number of conversions to an Ethernet packet with CAN messages. The average utilization of each bus is similar for all algorithms, whereas the difference in average Ethernet utilization is small. The fewest conversions from CAN messages to Ethernet packets for the buffer-type algorithm occur at GW1. In the deadline-aware algorithm (50% of periods), the number of conversions from CAN to Ethernet was also fewer than for the basic and urgency algorithms. Therefore, the deadline-type algorithm (50% of periods) gave

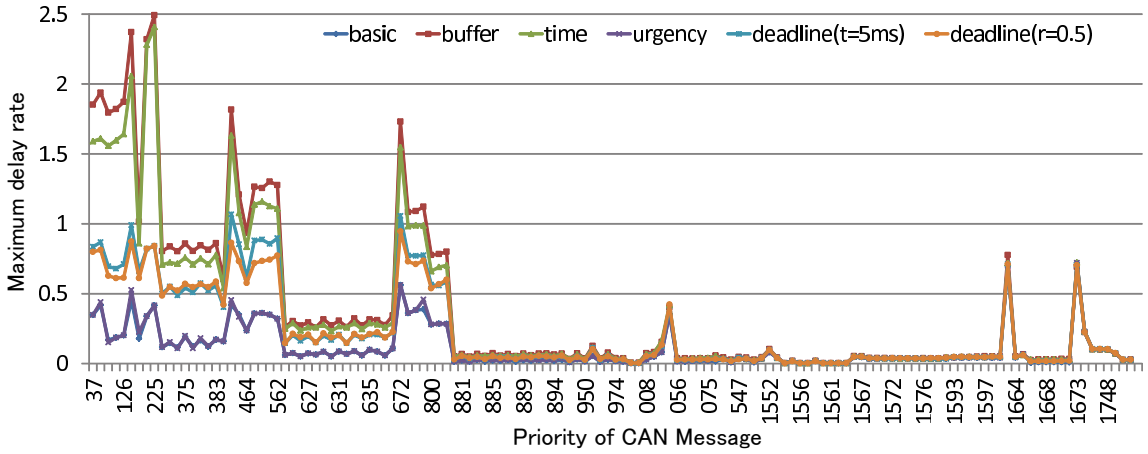


Fig. 12. Maximum delay ratio of each CAN message

TABLE IV. AVERAGE WORKLOAD OF EACH BUS AND THE NUMBER OF CONVERSIONS TO ETHERNET PACKETS

	Average workload [%]					# of conversions from CAN to Ethernet			
						GW1		GW2	
	CAN1	CAN2	CAN3	CAN4	Ethernet	empty	not empty	empty	not empty
Basic	32.78	69.61	45.31	50.81	2.14	1	1753806	1	11453219
Buffer	32.78	69.61	45.31	50.81	0.34	1	359999	1	363375
Time	32.78	69.61	45.31	50.81	0.35	1	366819	1	426005
Urgency	32.78	69.61	45.31	50.81	2.01	1	1667816	1	10637389
Deadline(5 ms)	32.78	69.61	45.31	50.81	0.45	1	718992	1	824190
Deadline(50% of periods)	32.78	69.61	45.31	50.81	0.48	1	718992	1	978810

the best performance in this case study.

V. CONCLUSION

We presented a simulation environment for CAN–Ethernet networks based on OMNeT++ and the INET framework, and discussed details of a CAN simulation model and a CAN–Ethernet GW model. To validate the developed CAN model, we measured the end-to-end latency of CAN messages using both the proposed simulator and Venet. Furthermore, we performed a case study for a CAN–Ethernet network, and confirmed the applicability of the proposed simulator. We showed that the proposed simulation environment is useful for the design and evaluation of CAN–Ethernet GWs and the topology of CAN–Ethernet networks. In future work, we will improve the conversion algorithm for UDP packets to CAN messages, and develop a flow control algorithm for use in a CAN–Ethernet GW. Finally we are planning to distribute the developed CAN model for OMNeT++ from our website.

ACKNOWLEDGMENT

Part of this work was supported by KAKENHI (24700027) and the Monbukagakusho scholarship.

REFERENCES

[1] Anton Cervin and Karl-Erik Årzén, “TrueTime: Simulation tool for performance analysis of real-time embedded systems”, Model-Based Design for Embedded Systems, 2009.

[2] OPNET, website:http://www.opnet.com

[3] Thomas Herpel, Kai-Steffen Hielscher, Ulrich Klehmet, and Reinhard German, “Stochastic and deterministic performance evaluation of automotive can communication”, *Computer Networks*, Vol. 53, No. 8, pp. 1171–1185, 2009.

[4] Andreas Kern, Dominik Reinhard, Thilo Streichert, and Jurgen Teich, “Gateway strategies for embedding of automotive can-frames into ethernet-packets and vice versa”, *Architecture of Computing Systems*, Vol. 6566, pp. 259–270, 2011.

[5] OMNeT++, website:http://www.omnetpp.org

[6] ADFweb, website:http://www.adfweb.com

[7] PHYTEC, website:http://www.phytec.com

[8] Jean-Luc Scharbarg, Frédéric Ridouard and Christian Fraboul, “A probabilistic analysis of end-to-end delays on an AFDX avionic network”, *IEEE Trans. on Industrial Informatics*, Vol.5, No.1, pp.38–49, 2009.

[9] Jonas Diemer, Jonas Rox, and Rolf Ernst, “Modeling of Ethernet AVB Networks for Worst-Case Timing Analysis”, *7th Vienna International Conference on Mathematical Modelling*, 2012.

[10] Klaus Wehrle, Jochen Reber, Dirk Holzhausen, Volker Boehm, Verena Kahmann, and Ulrich Kaage, “INET Framework for OMNeT++”, http://inet.omnetpp.org/doc/INET/inet-manual-draft.pdf.

[11] CANoe. website:http://www.vector.com/vi_canoe_en.html

[12] Robert Davis, Alan Burns, Reinder Bril, and Johan Lukkien, “Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised”, Real-Time Systems, Vol.35, No.3, pp.239–272, 2007.

[13] SymTA/S. website:http://www.symtavision.com/symtas.html

[14] Venet. http://www.kumikomi.net/archives/2004/11/31et04/ interd/vs.pdf (in Japanese)

CloudNetSim - Simulation of Real-Time Cloud Computing Applications

Tommaso Cucinotta, Aram Santogidis
Bell Laboratories, Alcatel-Lucent, Dublin

In this paper, we describe CloudNetSim, a project aiming to realise a simulation platform supporting our ongoing and planned research activities in the area of resource management and scheduling for distributed QoS-sensitive and soft real-time applications. It is based on OMNeT++, integrating in the platform a set of modules for the simulation of CPU scheduling, including hierarchical scheduling at both levels of the hypervisor and guest Operating System, as needed when simulating cloud infrastructures. Thanks to the modularity of OMNeT++, CloudNetSim may easily leverage many existing simulation models already available for networking, including standard network components and protocols, such as TCP/IP. After a brief overview of related simulation tools found in the literature, and the discussion of their limitations, we provide a detailed description of the internals of our simulator. Then, we show results gathered from a few representative scenarios demonstrating how its behaviour matches with the one of simple real applications.

1 Introduction

Cloud Computing is gaining momentum as one of the key innovations disrupting the world of computing, constituting a page turn from the old ages of Personal Computing to a new era of massively distributed cloud applications and services accessed from a plethora of devices with increased mobility support. Cloud Computing is also generating a continuous pressure towards the research community, for introducing innovations and novel mechanisms promising to support better the nowadays and future computing scenarios. As connectivity evolves towards higher bandwidth and lower latency, more and more soft real-time (RT) and interactive on-line applications are becoming increasingly used and popular [17]. These include many on-line interactive cloud applications, such as office suites (e.g., Google Docs) or e-Learning platforms [7], virtual desktop, and online massively parallel games.

When working at the lowest layers of the cloud infrastructure, and specifically at the hypervisor, Operating System (OS), and CPU scheduling levels, it is often difficult if not impossible to gain access to realistic test-beds over which to carry out research activities in the field. Often, it is very handy and convenient to have available tools that may assist researchers in simulating cloud deployments and end-to-end distributed applications, with a sufficient level of abstraction depending on the research purposes and scope.

However, simulation of distributed soft real-time applications over general-purpose computing platforms and networks is troublesome due to the lack of proper tools. Various simulators exist in the areas of networked systems and real-time systems, and recently a few simulation tools have become available in the area of Cloud Computing. In general, the existing tools were lacking the fundamental ability to integrate the various cross-domain simulation aspects that affect the end-to-end performance (see Section 2).

In this paper, we introduce CloudNetSim, a project aiming to provide a simulation platform to assist the experimentation with resource management and scheduling in cloud computing. At a glance, its main features comprise: packet-level simulation of end-to-end network communications between clients and servers distributed throughout a cloud infrastructure; simulation of computing resources including but not limited to CPU scheduling both at the hypervisor and at the guest OS levels; support for virtual machine (VM) deployment strategies; modularity and extensibility, with the possibility to introduce additional scheduling policies, VM deployment strategies and application models as needed. We aim to keep an abstraction level that allows for simulation of thousands of nodes and applications, gathering the necessary QoS metrics, within an affordable time.

Even though CloudNetSim targets simulation of cloud applications, the presented work may also be used for simulating networked soft real-time and embedded systems.

2 Related Work

In this section, the simulation tools mostly related to the presented work are briefly introduced. They fall roughly in the categories of real-time systems simulators, network protocols simulators and cloud computing simulators.

In the area of RT and embedded systems, many simulation tools deal with simulation of CPU scheduling, including *RTSim* [18], *MAST* [10], *MAST2* [9], *SimTrOS* [19]

and others [1], just to mention a few. However, either they are exclusively focused on hard RT and embedded systems and they do not support general-purpose schedulers and related technologies, or they neglect entirely the networking aspects. Some tools integrate simulation of CPU scheduling and technologies/protocols for CAN busses or Wireless Sensor Networks [6], however these tools are hardly reusable in the context of general purpose technologies.

In the area of networking and distributed systems, many tools provide an accurate simulation of networking technologies and packet-level simulation of networking protocols [21, 11, 20]. For example, *NS2*[1] is probably one of the most widely known open-source simulators used in research about network protocols, whose development started in 1996. It supports packet-level simulation of many Internet protocols and technologies, including TCP/IP and wireless networks. However, the simulator derives from a quite old code base, where functionality has been evolving over years, split around C/C++ and Object Tcl code. This resulted in the lack of modularity and clean interfaces for extending its functionality. It is not a case that, from 2004, a new *NS3*[2] project was born with the intention of a complete redesign of the internals of the simulator, and ultimately dropping compatibility with NS2. Unfortunately, this resulted in a set of features not (yet) as complete as in NS2. A completely alternative project is the open-source *OMNeT++*[3], free for academic use, and commercially licensed as *OMNEST*[4]. OMNeT++/OMNEST is a simulation platform with a completely modular design where generic modules can be connected in arbitrarily complex topologies and communicate with each other, all integrated with an Eclipse-based development environment including a visual topology editor. One of the main uses of OMNeT++ is in connection with the INET Framework[5], an open-source communication networks simulation package including a set of modules for simulation of Internet technologies and protocols, including TCP/IP, IPv6, Ethernet, PPP, 802.11, MPLS, and others.

However, all of these network simulators simply do not include any CPU scheduling infrastructure. In a cloud environment, where multiple VMs may be multiplexed on the same physical host, processor and core, it is important to simulate CPU scheduling, to get a comprehensive picture of the end-to-end response-time and performance. Especially when dealing with low-latency cloud applications deployed in future scenarios with fine-grained cloud data centres, tools are needed to support a *comprehensive* and *integrated* simulation of multiple resources, such as CPU, network and storage, that allow for modelling distributed applications, particularly those with QoS requirements, developed in the context of general-purpose technologies.

Recently, a few simulation tools have become available [2, 14, 4, 12] targeting the specific simulation needs arising in the area of Cloud Computing. *CloudSim* [2] is a Java-based simulation platform modelling various aspects of cloud computing infrastructures such as high-level simulation of data centres with virtualized hosts, energy consumption models and federated clouds. Versions prior to 2.0 have a very simple networking model at the flow level, with statically configured latency and bandwidth values among locations, while from version 2.0 a better network simulation functionality was added.

CloudSim is derived from *GridSim* [3], thus its architecture is still strongly tied to the modelling and simulation of GRID scenarios, with a focus on load balancing within the data centre, rather than gathering performance metrics over end-to-end deployments of general-purpose cloud computing applications, as in our proposed CloudNetSim.

iCanCloud [14, 4] is a simulator platform for cloud computing based on OMNeT++, with the capability to configure various resource management policies for the hypervisor, virtual machine models aiming to simulate the behaviour of real world CPUs, data centre topologies that mimic the architecture of state of the art cloud computing infrastructures (e.g. Amazon EC2[6]) and data storage emulation. Still, this tool is lacking the essential capability to simulate the variety of heterogeneous networks involved in the end-to-end cloud service supply chain. However, being based on OMNeT++ as our framework, iCanCloud has interesting modules that we might re-use, such as the storage models inherited from *SIMCAN* [16, 15], a simulator of local and remote storage systems, including NFS and parallel file systems.

GreenCloud [12] is an NS2-based C++ simulator aiming to model the energy consumption of data center IT equipment (e.g. computers, network switches and communication links), to help the design of energy efficient architectures. However, GreenCloud needs merely a rough estimate of the expected computing workload on the nodes, for its power consumption estimates.

Overall, some of the mentioned simulators targeting cloud computing focus specifically on aspects of the infrastructure related to computing within the data centre, neglecting the important aspects of communications over the Internet or the access network. Others try to enrich an accurate simulation of the network by adding rough computing models which cannot capture a similar level of detail, when addressing QoS and responsiveness. However, consideration of the whole end-to-end chain is very important for the overall QoS delivered to remote customers/users.

As a consequence, we could not find in existing tools

[1] More information at: `http://www.isi.edu/nsnam/ns/`.
[2] More information is available at: `http://www.nsnam.org/`.
[3] More information is available at: `http://omnetpp.org/`.
[4] More information is available at: `http://www.omnest.com`.
[5] More information at: `http://inet.omnetpp.org/index.php`.
[6] More information is available at: `aws.amazon.com/ec2/`.

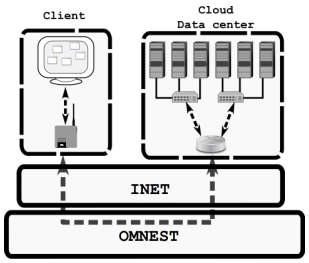


Figure 1. High-level software architecture

features properly matching with the particular needs of research on the topics of resource management and scheduling for interactive, real-time and low-latency cloud and distributed applications.

3 Proposed Approach

One of the primary goals of the overall ongoing CloudNetSim project is to integrate within a single simulation platform the major factors contributing to end-to-end latency of low-latency cloud applications, namely networking, computing and disk access, including overheads due to virtualization (both machine and network virtualization).

We opted to implement the computing part of the simulation on top of OMNEST (see Figure 1), due to its relative maturity, modular design and extensibility. We realized a set of OMNEST modules in order to model computing and CPU scheduling within physical hosts and VMs, as happening within a cloud computing data centre; these inter-mix with the already available network communication modules, resulting in a more comprehensive emulation of the major contributions to end-to-end response-times. At this preliminary stage, disk access has been greatly simplified, but we plan to consider it more thoughtfully later.

Simulating large infrastructures with such a fine-grained level of detail for computing and networking resources may present performance and scalability challenges. However, it has been shown [13] that parallelisation techniques can be effectively applied to OMNeT++ simulations in a seamless fashion, without requiring changes in the code. These will certainly be useful for our planned future investigations.

Overall Design. The core component for modelling computing elements is *CloudNode*. It is built on top of the *NodeBase* compound module of INET which models a network host, and provides interface and network layer functionality. CloudNode additionally incorporates a number of modules that emulate the computing part of the module (see Figure 2), i.e., the CPU Scheduler, modules for applications and for data storage emulation. Moreover, the CloudNode

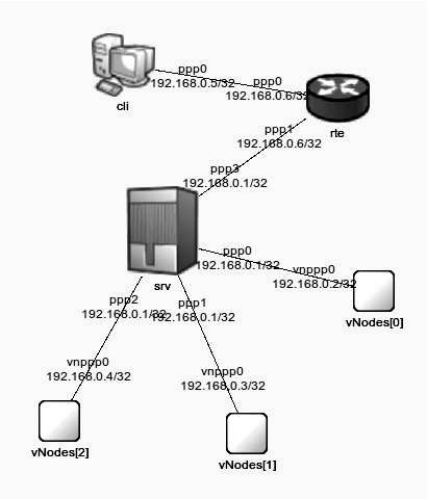


Figure 2. OMNEST representation of a simple topology.

modules can be interconnected with each other in a hierarchical fashion, effectively modelling VMs running within a physical host. Figure 2 illustrates the topology of a client connected through a router to a host running 3 VMs.

In OMNEST terminology, CloudNode is a *compound module* that extends NodeBase (see Figure 3 for an overview of its inner design). It is an aggregation of *simple modules* that allow for modelling various aspects of the software stack typical of virtualized infrastructures. As depicted in Figure 3, CloudNode includes simulation of network capabilities as inherited from NodeBase, data storage and CPU scheduling. The networking capabilities have been customised by adding *SchedPPP*, a module extending the INET PPP interface which is controllable from the CPU scheduler. This is necessary in order to "suspend" the network connectivity of a VM, when it is preempted from execution by the CPU scheduler. A similar *SchedEth* module has been realised for Ethernet.

The *Scheduler* within a CloudNode is able to schedule an arbitrary number of *Schedulable* entities over a configurable number of available CPUs. Also, these can be connected to a *data storage model* in order to model suspension on I/O. Interestingly, a CloudNode is schedulable on its own. This allows VMs to be modelled as CloudNode instances connected to the Scheduler of the outer CloudNode representing the host they are deployed within.

Scheduler Design. Schedulable entities represent software running in the system, including both applications or components at the hypervisor level, and those within guest VMs. The *Schedulable* interface permits the Scheduler to

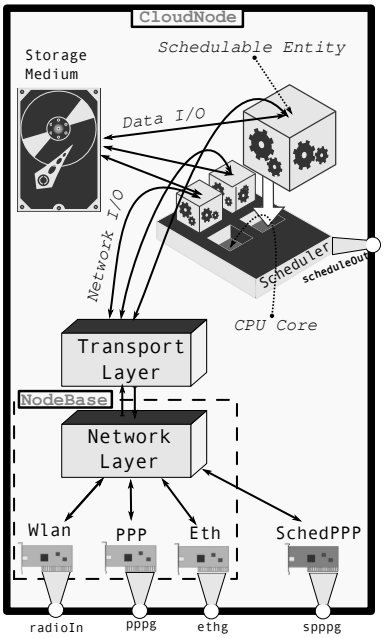


Figure 3. CloudNode design.

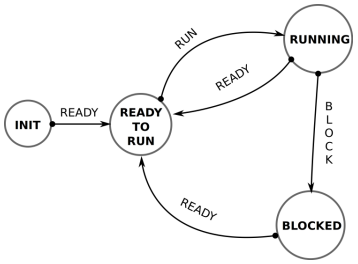


Figure 4. Schedulable entities FSM.

manage their execution state. All these entities extend the *BaseSchedulable* class that implements the well-known Finite State Machine (FSM) in Figure 4. The logical communications between the Scheduler and its managed entities, necessary to realise the mentioned FSM behaviour, is conducted through the *scheduleIn/scheduleOut* ports of the Schedulable interface, by exchanging custom defined OMNEST messages. Specifically, the Scheduler notifies ready-to-run entities whenever a CPU is assigned or revoked to them, according to the scheduling algorithm in use within the Scheduler. The entities, on their own, notify the Scheduler whenever they need to suspend for data I/O, networking, or timer operations.

The CPU is modelled in the scheduler with a few configurable parameters controlling its power-saving capabilities, including the frequency at which it is running, and whether it is in a deep-idle state. Therefore, messages from the Scheduler to the entities also include the frequency change information, needed to allow applications to modulate their execution time behaviours accordingly. This allows for simulation of multi-processor and multi-core hosts with CPU power-saving capabilities. However, an exact strategy to switch among the available CPU frequencies (i.e., mimicking the behaviour of the `cpufreq` governors in Linux) is still work in progress. Also, we only modelled a single idle-state of the CPU with a configurable wake-up latency, as at the moment there is no interest in modelling the multitude of idle states in modern CPUs.

We realised 3 scheduling algorithms: Fixed Priority (FP), Round-Robin, Linux Completely Fair Scheduler (CFS). These can be hierarchically composed with each other. This is shown through the example in Figure 5.(a), where a typical Linux set-up is shown with 6 applications running under various scheduling policies, as detailed in Figure 5.(b). With the proposed architecture, multiple real-time tasks at the same priority under the POSIX SCHED_RR policy are represented as connected to an instance of the Round-Robin Scheduler, which in turn is connected to the FP Scheduler at the needed priority level. Also, SCHED_OTHER tasks are connected to a CFS Scheduler connected to the FP Scheduler at priority 0.

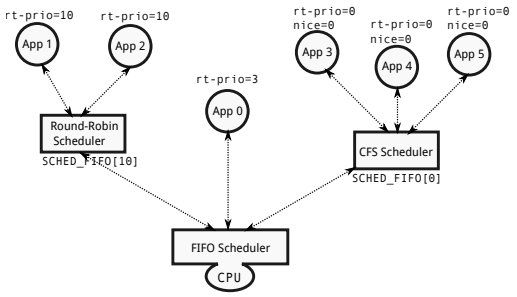
Configuration of the Scheduler(s) topology is simplified by specifying for each application the desired scheduling parameters (including the nice level, in case of SCHED_OTHER entities), and the CloudNode instantiates the required Scheduler modules and interconnections as needed. Note that the overall Scheduler design allows for an easy introduction of new algorithms.

Application Model. Applications are modelled in CloudNetSim as Schedulable entities, executing sequentially a list of instructions. Following the steps of RTSim [18], the purpose of the simulation is not functional simulation, but rather performance evaluation. Therefore, allowed instructions are for now: computing for a fixed amount of time (scaled linearly with the CPU frequency); wait for the transfer of a fixed number of blocks to/from the storage medium; change dynamically the scheduling parameters of the application. Also, a few instructions are being realised allowing for modelling (the impact on performance of) communications among various parts of a distributed cloud application.

A convenience scripting syntax has been defined, so that simple application models may easily be provided through text-based input files to the simulation.

4 Calibration and Simulation Experiments

In this section we report results from a few experiments we ran in order to show how the parameters of the simulated models may be calibrated so that its outcome matches with the behaviour measured from a real simple scenario.



(a)

App	Priority	Policy
0	3	SCHED_FIFO
1	10	SCHED_RR
2	10	SCHED_RR
3	0	SCHED_OTHER
4	0	SCHED_OTHER
5	0	SCHED_OTHER

(b)

Figure 5. Hierarchical scheduling of processes based on policy

	Real world	Simulation
Host (idle)	0.384 +/- 0.040 ms	0.388 ms
Host (hog)	0.322 +/- 0.034 ms	0.333 ms
VM1 (idle)	0.482 +/- 0.034 ms	0.462 ms
VM1 (hog)	0.377 ms +/- 0.036 ms	0.399 ms

Table 1. Ping times statistics for the real-world (left) and simulated (right) scenarios.

Ping Test. We consider a simple topology with a physical host running two VMs connected through a network router to a client that pings the physical host and the VMs (see Figure 2). After calibration of the simulation model parameters, its results are compared with numbers obtained by the corresponding real-world example. In the latter, we used an Intel i5-2520M @ 2.50GHz laptop client pinging a Linux machine equipped with an Intel Xeon E5-2687W CPU whose frequency was fixed at 3.1 GHz, and in which all cores except one have been put offline, and hyper-threading has been disabled, to create the simple scenario reproduced in simulation. Also, a guest KVM Linux OS has been run on the server machine. The host and the VM have been continuously pinged for one minute every half second, when the host was idle, and when it was loaded. The obtained ping times average and standard deviation are shown in Table 1, in both real-world and simulated cases.

The overall ping latency towards the host is the result of summing up delay contributions due to network delay to reach the server, CPU wake-up from idle,networking stack execution for replying to the ping, then back to the client. When pinging the VM, further contributions are due to the context switch to schedule the VM and guest OS networking stack execution for replying to the ping.

CFS Test. We ran another ping experiment using the CFS as the hypervisor scheduler. We verified that, despite a 2nd VM hogging the CPU, the pinged idle VM was responding immediately to the ping, preempting the other one. This behaviour is in sync with the CFS algorithm since the pinged VM, waking up from a blocked state, runs immediately, since its virtual run-time is much lower than the one of the CPU-bound VM continuously executing.

Then, to verify the behaviour of the CFS in presence of different nice values, we considered another simple scenario with three applications running CPU bound tasks on a host and we compared the obtained simulated versus real figures.

We use a `load.sh` shell script realising a simple `for` loop for the number of iterations provided as argument. When running on an Intel i5-2520M CPU at fixed 2.50 GHz frequency, `load.sh` takes 1 second to complete with an argument of 177000. In single-core mode, we run three tasks with the default nice value (0), however the third one is reniced to (10) at half execution. This is obtained as:

```
time ./load.sh 177000 &
time ./load.sh 177000 &
time ./load.sh 88500
time nice ./load.sh 88500
```

The obtained results show that the first two processes completed in less than 2.6 seconds, whilst the reniced process completed after $1.56 + 1.49 = 3.05$ seconds:

```
0.47u 0.02s 1.56r   ./load.sh 88500
0.93u 0.03s 2.55r   ./load.sh 177000
0.95u 0.03s 2.58r   ./load.sh 177000
0.51u 0.00s 1.49r   nice ./load.sh 88500
```

The same experiment has been arranged in the **simulated** model, using the renice instruction explained in the previous section for changing dynamically the third process nice level at half of its execution. This resulted in the following output, gathered from the OMNeT++ logs:

```
T=3.004008  TestCloudNode.srv.tcpApp[0]
T=2.560008  TestCloudNode.srv.tcpApp[1]
T=2.554008  TestCloudNode.srv.tcpApp[2]
```

These results validate the correct behaviour of the CFS Scheduler model, in the mentioned scenario.

5 Conclusions and Future Work

In this paper we presented CloudNetSim, a simulation platform suitable for capturing the behaviour of end-to-end

time-sensitive and particularly low-latency distributed applications. The platform exploits the native OMNEST and INET capabilities for network simulation, integrating simulation of computing and storage access in virtualized environments. We plan to use this platform for our ongoing and planned research in the area of resource management and scheduling for soft real-time cloud computing applications. The presented simulation models are very important to simulate the impact on performance of sharing physical computing resources within the infrastructure, as often done by cloud providers trying to achieve high consolidation levels. However, CloudNetSim may also be useful for simulation of soft real-time distributed embedded systems.

The presented work may be extended along various lines of action: the CPU scheduling models may be refined by adding further scheduling policies, e.g., one mimicking the Xen scheduler [5, 8]; the storage access model is very simple, but re-usable modules from other projects such as SIMCAN might be integrated; the performance achievable with the integrated multi-resource simulation on large scale systems has to be checked, an area where parallelisation techniques such as [13] might be useful.

References

[1] M. Ashjaei, M. Behnam, and T. Nolte. The design and implementation of a simulator for switched ethernet networks. In *Proc. of the 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, pages 57–62, Pisa, Italy, July 2012.

[2] R. Calheiros et al. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[3] R. N. Calheiros, R. Ranjan, C. A. F. D. Rose, and R. Buyya. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *CoRR*, abs/0903.2525, 2009.

[4] G. Castane, A. Núñez, and J. Carretero. iCanCloud: A brief architecture overview. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 853–854, 2012.

[5] L. Cherkasova, D. Gupta, and A. Vahdat. Comparison of the three cpu schedulers in xen. *SIGMETRICS Perform. Eval. Rev.*, 35(2):42–51, Sept. 2007.

[6] M. Chitnis et al. Impact Of The Operating System on the QoS offered by an IEEE 802.15.4-compliant Sensor Network. In *Proceedings of the 7th IFAC International Conference on Fieldbuses and networks in industrial and embedded systems*, Tolouse, France, Nov 2007.

[7] T. Cucinotta et al. Virtualised e-learning with real-time guarantees on the irmos platform. In *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2010)*, pages 1–8, Perth, Australia, December 2010.

[8] G. Dunlap. Scheduler development update. Xen Summit Asia 2009, Shanghai, 11 2009.

[9] M. G. Harbour et al. Modeling real-time networks with mast2. In *Proc. of the 2nd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, pages 51–56, Porto, Portugal, July 2011.

[10] M. G. Harbour, J. J. G. García, J. C. P. Gutiérrez, and J. M. D. Moyano. Mast: Modeling and analysis suite for real time applications. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, ECRTS '01, pages 125–, Washington, DC, USA, 2001. IEEE Computer Society.

[11] T. Issariyakul and E. Hossain. *Introduction to Network Simulator NS2*. Springer, 2009.

[12] D. Kliazovich, P. Bouvry, and S. U. Khan. A packet-level simulator of energy- aware cloud computing data centers. *Journal of Supercomputing*, 62(3):1263–1283, 2012.

[13] D. Lugones, K. Katrinis, M. Collier, and G. Theodoropoulos. Parallel simulation models for the evaluation of future large-scale datacenter networks. In *Proc. of the 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, DS-RT '12, pages 85–92, Washington, DC, USA, 2012.

[14] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente. iCanCloud: A flexible and scalable cloud infrastructure simulator. *J. Grid Comput.*, 10(1):185–209, Mar. 2012.

[15] A. Nunez et al. Design of a flexible and scalable hypervisor module for simulating cloud computing environments. In *Performance Evaluation of Computer Telecommunication Systems, International Symp. on*, pages 265–270, 2011.

[16] A. Nunez, J. Fernandez, J. Garcia, and J. Carretero. New techniques for simulating high performance MPI applications on large storage networks. In *Cluster Computing, 2008 IEEE International Conference on*, pages 444–452, 2008.

[17] E. Oliveros, A. Mazzetti, W. Huther, and A. Menychtas. Irmos deliverable d2.1.3 - final version of requirements analysis report. Technical report, IRMOS Consortium, Nov 2010.

[18] L. Palopoli, G. Lipari, G. Lamastra, L. Abeni, G. Bolognini, and P. Ancilotti. An object-oriented tool for simulating distributed real-time control systems. *Softw. Pract. Exper.*, 32(9):907–932, July 2002.

[19] J. Schneider, M. Bohn, and C. Eltges. SimTrOS: A Heterogenous Abstraction Level Simulator for Multicore Synchronization in Real-Time Systems. In *Proc. of the 2nd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, pages 39–44, Porto, Portugal, July 2011.

[20] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, Simutools '08, pages 60:1–60:10, Brussels, Belgium, 2008. ICST.

[21] E. Weingartner, H. vom Lehn, and K. Wehrle. A performance comparison of recent network simulators. In *Proceedings of the IEEE International Conference on Communications 2009 (ICC 2009)*, Dresden, Germany, 2009. IEEE.

A Simulation Tool for Optimal Phasing of Nodes Distributed over Industrial Real-Time Networks

Sang-Hun Lee Hyun-Wook Jin
Dept. of Computer Science and Engineering
Konkuk University
Seoul, Korea
{mir1004, jinh}@konkuk.ac.kr

Kanghee Kim
School of Electronic Engineering
Soongsil University
Seoul, Korea
khkim@ssu.ac.kr

***Abstract*—Emerging industrial real-time networks, such as EtherCAT and PROFINET, provide highly accurate clock synchronization. Thus, this feature opens a new chance to adjust phasing across distributed nodes aiming for better synchronization of dependent tasks. However, obtaining an optimal node phasing across distributed nodes has not been given enough attention while the worst-case task phasing on each node assuming no global clock has been studied in many ways. In this paper, we suggest a simulation tool that searches for an optimal phasing on distributed nodes with respect to less end-to-end delays and less actuation jitters. The proposed tool holistically simulates task scheduling on each node, transmission of network messages, DMA, and I/O event handling. It tries to reduce the time to find an optimal node phasing by skipping uninteresting phase combinations. Through a case study, we show that the simulator can efficiently suggest an optimal node phasing across distributed nodes and provide distribution of possible end-to-end delays and actuation jitters for given task sets.**

Keywords—node phasing; industrial network; end-to-end delay; actuation jitter; clock synchronization

I. INTRODUCTION

Emerging industrial real-time networks, such as EtherCAT [1] and PROFINET [2], provide highly accurate clock synchronization between distributed nodes. For instance, the distributed clock of EtherCAT can provide accurate clock synchronization with errors less than few tens of nanoseconds [3][4]. PROFINET uses Precision Transparent Clock Protocol (PTCP) for clock synchronization. Such accurate clock synchronization opens a new chance to adjust phasing across distributed nodes aiming for better synchronization of dependent tasks (e.g., multi-axis motion controls).

However, obtaining an optimal phasing across distributed nodes has not been given enough attention while the worst-case task phasing on each node has been studied in many ways [5][6]. This is mainly because there has been limited support for clock synchronization on legacy industrial networks and thus researchers have focused on analysis of the worst-case end-to-end delays between tasks assuming no global clock. In order to find the worst-case end-to-end delays, existing studies suggest an analytical model based on a worst-case task phasing on every node, but they introduce too much pessimism into the analysis [7][8]. However, once we make an assumption of a precise global clock for all the nodes, we can investigate into the problem of finding a phase combination in order to reduce the end-to-end delays and the actuation jitters. The problem can be dealt with by a simulation approach or an analytical approach.

In this paper, we propose a simulation framework that searches for an optimal node phasing on distributed nodes with respect to the end-to-end delays and the actuation jitters. The proposed framework considers behavior of several components composing the target system in a holistic manner to analyze the impact of node phasing on the metrics. The target system is assumed to interconnect one master and multiple slave nodes with a real-time network such as EtherCAT. In our framework, we model each node as a set of periodic tasks scheduled by a fixed-priority scheduling algorithm. In addition, we consider interactions between the nodes and the network, which include network event handling in either polling or interrupt mode, message queuing on network device, and direct memory access (DMA). In the simulation, we also take into account the communication delays in the real-time network, which are affected by the packet forwarding scheme at each node and the network topology.

One novelty of the proposed framework is that we try to reduce the time to find an optimal node phasing by skipping uninteresting phase combinations. Usually, an exhaustive search needs to consider a huge number of phase combinations, which is not tractable. We search for a limited set of the node phase combinations based on the observation that only a few specific states produce an unpredictable trend of end-to-end delays and actuation jitters. That is, our framework runs simulation only for phase combinations that generate such interesting states, while the target metric for other predictable states are calculated by using simple mathematical equations. Through a case study, we show that the proposed simulation framework can efficiently suggest an optimal node phasing across distributed nodes and provide distributions of the end-to-end delays and the actuation jitters for given task sets.

The rest of the paper is organized as follows. Section II describes the system model we assume. In Section III, we suggest our simulation framework for optimal node phasing. Section IV shows a case of finding an optimal node phasing for

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (#2012R1A2A2A02015266 and # 2011-0020905).

a given task set, and Section V summarizes the related work. Finally, in Section VI, we conclude the paper.

II. System Model

We consider a distributed real-time system that interconnects one master node and *N* slave nodes with a real-time fieldbus such as EtherCAT, which is common in many industrial applications. For example, a motion control system has such a system configuration where the master node generates position commands to describe the motion trajectory and the slave nodes are motor drives to respond to the commands. In the following, we explain the node model and the network model.

A. Node Model

We consider a set of periodic tasks for both the master and the slave nodes. All the slave nodes are assumed to have the same set of tasks for the sake of brevity, which is not necessary for our simulation framework. The master node may have a different set of tasks than that of the slave nodes. The tasks are denoted by $T_{1,m}, \dots, T_{n,m}$ for node m, where n can be different for the master ($m = 0$) and the slaves ($1 \le m \le N$). We denote each task by $T_{i,m} = (e_{i,m}^{min}, e_{i,m}^{max}, p_{i,m}, \emptyset_{i,m})$, where $e_{i,m}^{min}$ and $e_{i,m}^{max}$ are its minimum and maximum execution time, respectively, and $p_{i,m}$ is its period. Each task gives a rise of an infinite sequence of jobs. $J_{i,m}^{j}$ denotes the j^{th} job of $T_{i,m}$. The release time of $J_{i,m}^{j}$is denoted by $r_{i,m}^{j}$, and the release time of the first job, i.e., $r_{i,m}^{1}$, is called the phase of $T_{i,m}$ and denoted by $\emptyset_{i,m}, (0 \le \emptyset_{i,m} < p_{i,m})$. In our simulation framework, we assume the critical instant (i.e., $\emptyset_{1,m} = \emptyset_{2,m} = \cdots = \emptyset_{n,m} = 0$) in each node with a single-processor. In the paper, we focus on finding an optimal combination of the node phases, which are represented by $(\Delta_1, \Delta_2, \dots, \Delta_N)$ in Fig. 1. The node phase Δ_m can be varied between 0 and p_m^H. (p_m^H is the least common multiple of the periods of all tasks in node *m*.) We assume that the deadline of each job is the same with the end of its period.

For the scheduling algorithm, we assume a fixed-priority scheduling algorithm such as Rate Monotonic [9]. Since each task is assigned a static priority, the execution of a lower priority task can be preempted by a higher priority task. Thus, the response time of $J_{i,m}^{j}$ reflects the delay due to such preemption and is denoted by $R_{i,m}^{j}$.

In our task model, to address a realistic scenario commonly found in many industrial networks, we assume that only one task can access the network device for each node. The rationale behind this is that many industrial network protocols do not support multiplexing and de-multiplexing between tasks [10][11]. Moreover, as the name implies, only the master node is assumed to initiate a message transmission. Each slave node may only piggyback some data with the sender task on the message passing through the node on the fly, but cannot initiate a message transmission. Once the sender task at the master creates a message and passes it to the network device, the message experiences a queuing delay in the device, denoted by D_q in Fig. 1, and then is transmitted through the network. Likewise, when the message arrives at a slave node, it requires a message handling time, denoted by D_e, before it becomes available to the receiver task.

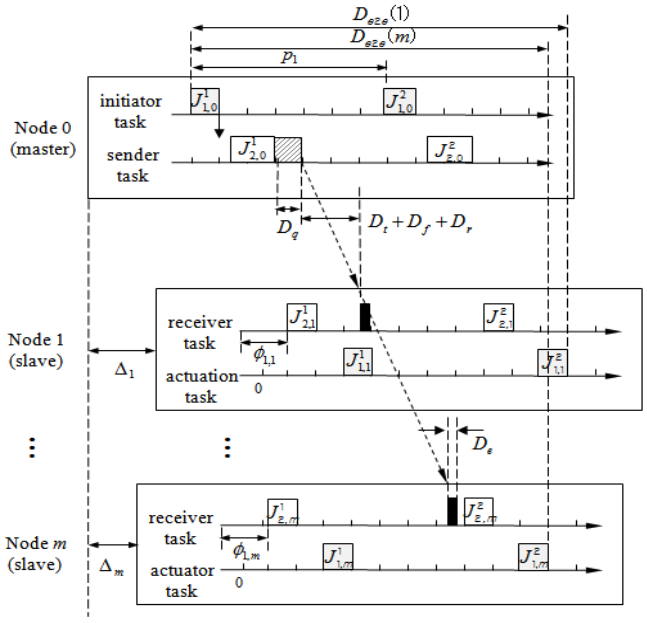


Fig. 1. System model

B. Network Model

We assume a real-time fieldbus, such as EtherCAT, for real-time communications between the nodes. EtherCAT guarantees deterministic communication delays between any two nodes by the following two design choices. The first one is that it interconnects any two adjacent nodes with a dedicated link in a daisy-chain manner. That is, there are no interfering nodes between nodes *m* and *m*+1, and thus no packet collisions on the link. The second one is that each slave node *m* conducts wormhole switching to relay the packets between nodes *m*-1 and to *m*+1, not store-and-forward switching. This switching scheme implemented at the hardware level eliminates the chances that internal software operations of each slave contribute to the end-to-end communication delay. As a result, EtherCAT guarantees deterministic communication delays from the master to any slave *m*, which is analyzed by Pritz et al. [12] as follows:

$$D_{comm}(m) = D_t(s) + (m-1) \times D_f + D_r(s), \quad (1)$$

where $D_{comm}(m)$ is the communication delay from the master's memory to slave *m*'s memory. $D_t(s)$ is the total transmission time of a message of size *s* on the master network device. D_f is the total forwarding time of a message on the slave side, which is a constant of 1 μs, irrespective of the message size. $D_r(s)$ is the total reception time of a message of size *s* on the slave network controller.

In our simulation framework, we adopt the above delay model expressed by Eq. 1. Therefore, the end-to-end delay from the release time of an initiator task at the master to the completion time of an actuator task at slave *m* can be described as follows:

$$D_{e2e}(m) = D_{master} + D_{comm}(m) + D_{slave}(m), \quad (2)$$

where D_{master} is the delay between the release time of the initiator task at the master and the completion time of the sender task plus the queuing time D_q, and $D_{slave}(m)$ is the delay between the arrival time of a message at slave *m* and the completion time of the actuator task. It is important to understand that, in Equation 2, only $D_{slave}(m)$ may be affected by the node phase Δ_m, thus by changing the value of Δ_m, we may be able to minimize the end-to-end delays and the actuation jitters.

In our simulation framework, the target metrics we want to analyze are the end-to-end delays, the actuation jitters, and the minimum possible queue size q at the network interface that does not cause any packet to be dropped for the entire simulation time, that is, $\forall t, max(q_t) \leq q$. In the next section, we will describe the proposed simulation framework that gives the target metrics by finding an optimal combination of the node phases $(\Delta_1, \Delta_2, \ldots, \Delta_N)$.

III. The Proposed Framework

In this section, we describe the proposed simulation framework that finds an optimal phasing for distributed nodes. The framework is based on discrete-event simulation to simulate task scheduling, message transmission/forwarding, DMA, and I/O event handling.

A. Overall Design

The simulator is based on the system model described in Section II. Fig. 2 shows the overall design of the proposed simulator. It consists of four components: input file parser, simulation kernel, node objects, and log manager. Configuration parameters, such as number of nodes, task sets, network bandwidth, message size, and event handling mode, etc., are defined in the configuration file in XML format. The *file parser* reads this configuration file at the initialization phase and passes them as environment variables to the *simulation kernel*. Then the kernel actually performs simulation loops changing node phasing. The *node objects* store the run-time snapshot of each node. The *log manager* stores every event generated during simulation to output files so that user can replay the simulation manually.

B. Node Objects

As mentioned above, the node objects hold a snapshot of run-time image of each node, such as state information of run queue, message queue, and network controller.

The *run queue* stores task control blocks, each of which has $e_{i,m}^{min}$, $e_{i,m}^{max}$, $p_{i,m}$, mission, and the amount of time that the corresponding task executed, called `etime` in this paper. The mission is classified into four categories in the current implementation: *send*, *receive*, *actuation*, and *other*. This information decides the behavior of the task. More details of this information are described in the next subsection. The `etime` variable represents time units consumed by the task in the current period.

The *message queue* stores network messages, which are yet to be sent to the network or yet to be consumed by a receive task. The queue traces its length q_t at time *t* and reports the maximum queue length at the end of simulation.

The *network controller* decides the message transmission speed based on the bandwidth information. It maintains the remaining bytes of a message being sent.

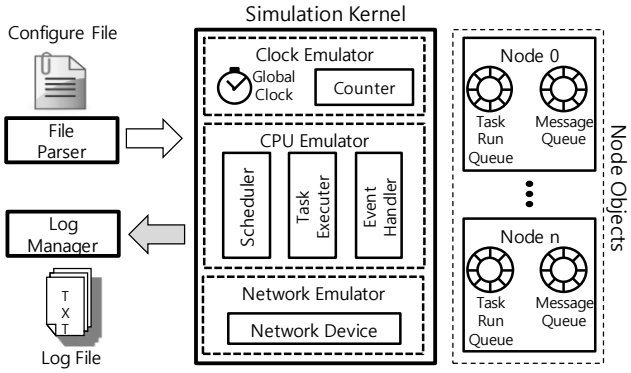


Fig. 2. Overall design of simulator

C. Simulaton Kernel

The simulation kernel comprises clock emulator, CPU emulator, and network emulator. The *clock emulator* simulates synchronized global clock (e.g., distributed clock of EtherCAT) and increases its counter for every simulation loop.

The *CPU emulator* runs task(s) on each node for every time unit. The scheduler selects a task to run, and the task executer emulates the behavior of the selected task based on its mission information as mentioned in the previous subsection. If the mission is *send*, the task executer posts a send request to the network device at the end of execution time of the job. On the other hand, in the *receive* case, the task executer consumes a message from the message queue at the starting point of the job. When the mission is *actuation*, the task executer reports that an actuation has taken place at the completion time of the task. The task executer simply increases the `etime` value for tasks of *other* mission. Our current implementation only supports RM scheduling algorithm but our design is general enough to add other scheduling algorithms.

The *network emulator* simulates operations of the network device (e.g., message transmission, DMA, and raising an interrupt). Event handling can be done in either polling or interrupt mode. If the network device is configured as a polling-based device, the event is handled when a receive task is released. Thus, in this case, the network emulator silently inserts a received message into the receive queue without any notification. When the network device is configured as an interrupt-based device, the network emulator sends an interrupt signal to the CPU emulator so that the interrupt handler is invoked as soon as a message arrives preempting a running task.

To get results for a given phase case, the kernel runs simulation loops emulating operations described above on each node for every time unit.

D. Optimizing Search Time

If the simulator exhaustively repeats the steps described in the previous subsection for all possible combinations of phases, the search time for an optimal node phasing would be very

significant. This search time increases linearly or even exponentially as number of nodes or possible phasing range becomes larger. These also affect the volume of disk I/O performed by the log manager, which is one of dominant overheads of the simulator.

> ***S1***. *A message arrives on idle time, and the event handler does not affect execution of any tasks.*
>
> ***S2***. *A message arrives on idle time, but the event handler delays execution of a task.*
>
> ***S3***. *A message arrives at a task's release time, and the event handler preempts the task.*
>
> ***S4***. *A message arrives while a task runs, and thus the event handler preempts the task.*

Fig. 3. Possible correlation states

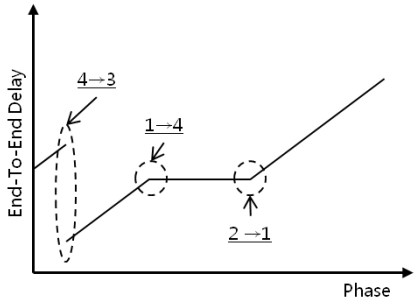


Fig. 4. End-to-end delay variation by changing phase

In order to tackle this issue, we try to reduce the number of phases we investigate while we still gather information for all possible phases. It is important to understand the correlation between phase and end-to-end delay to find interesting phase points. We can easily notice that the phase of a slave node does not affect other slaves' end-to-end delay by the system model described in Section II. This means that we do not need to consider all combinations of node phases for simulation but can concentrate on phases for each node independently. Due to this characteristic, we can reduce the number of simulation loops.

During simulations, we also observed that in most cases the end-to-end delay increased as phase became larger. Moreover, we found that at some points the end-to-end delay started showing a constant value for a while or reduced suddenly to a very small value as phase became larger. By analyzing these results further, we noticed that a phase resulted in one of states shown in Fig. 3, and the trend of end-to-end delay was changed when a state is changed from one to another by a new phase. Thus, by simulating only for phase points that changes the state, we can significantly reduce the number of simulation loops.

Fig. 4 shows an example of end-to-end delay variation as the discrepancy between phases of the master and a slave increases. Numbers in the figure present state transitions. As we can see in the figure, the end-to-end delay increases in most cases, while the slope of the line is changed (when the state changes from S1 to S4 and from S2 to S1) or the line is disconnected (when the state changes from S4 to S3).

IV. Simulation Example

In this section, we run a case to show that the proposed simulation framework can efficiently suggest an optimal phasing across distributed nodes and provide distributions of the end-to-end delays and the actuation jitters for given task sets.

A. Configuration

Table I shows the task set we run on slave nodes. We refer the task set defined in Kim et al. [13] for a motor drive. Each slave node schedules these tasks using the RM algorithm. Table II shows the configuration parameters. We borrow network parameter values, such as queue size, bandwidth, and forwarding delay, from EtherCAT.

TABLE I. Task Set of Slave Node

Task Name	Period (us)	Execution Time (us)	Description
MotorAct	250	25~35	Controls a motor, and is assigned the highest priority
RtMsg	250	10~15	Receives real-time messages from the network, and shares it with the MotorAct task
NrtMsg	250	7~10	Handles non-real-time messages, and is assigned a lower priority than the RtMsg task
HealthMon	500	6~9	Performs health monitoring, and is assigned the lowest priority

TABLE II. Configuration Parameters

Parameters	Value
Number of slave nodes	50
Network queue size	3
Network bandwidth	100Mbps
Event handling mode	Interrupt
Event handling overhead	5us
Simulation time unit	1us
Network forwarding delay (D_f)	1us
Packet size	50byte
DMA overhead for a 50byte packet	1us

B. Optimal Phase Combination

We ran simulator with tasks defined in Table I and got the node phase combinations to achieve the best end-to-end delay and actuation jitter as shown in Fig. 5 and Fig. 6, respectively. To see the impact of task sets on the optimal node phasing, we had three more cases in addition to the default task set case (i.e., case 1). In case 2, we assigned the highest priority to RtMsg and second to MotorAct. In case 3, we increased the execution time of RtMsg to 21~30us. In case 4, we increase the execution time of MotorAct to 49~70us. Tables III and IV show the end-to-end delays and the actuation jitters for node phase combinations presented in Fig. 5 and Fig. 6, respectively. Since we consider maximum and minimum execution time of tasks,

end-to-end delay and actuation jitter are also represented as pairs of maximum and minimum values.

As shown in Fig. 5 and Fig. 6, we can observe that providing minimum end-to-end delay does not guarantee minimum actuation jitters. Since actuation jitter only considers the response time of the MotorAct task on all slave nodes, the minimum actuation jitter is achieved when the message from the master node is arrived before the actuation task is released while slaves have the (almost) same phase. However, this increases the end-to-end delay.

C. Search Time

As described in Subsection III.D, we tried to reduce the time to find an optimal node phasing by skipping uninteresting phase points. Fig. 7 shows simulation results only for interesting points, where the correlation state is changed, on 35th slave node. In this case study, two packets are received in p_m^H; thus, each phase point reports two end-to-end delay values, though these are very close. Fig. 8 shows that we can obtain rest of values by connecting the points in Fig. 7. In this way, we can reduce the number of simulation loops significantly and save the search time.

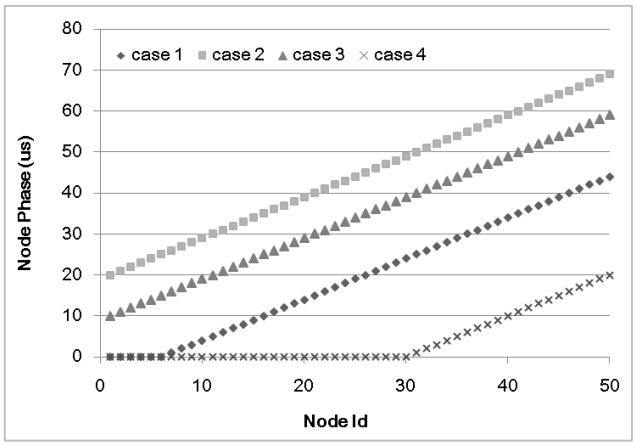


Fig. 5. Node phase combination for minimum worst-case end-to-end delay

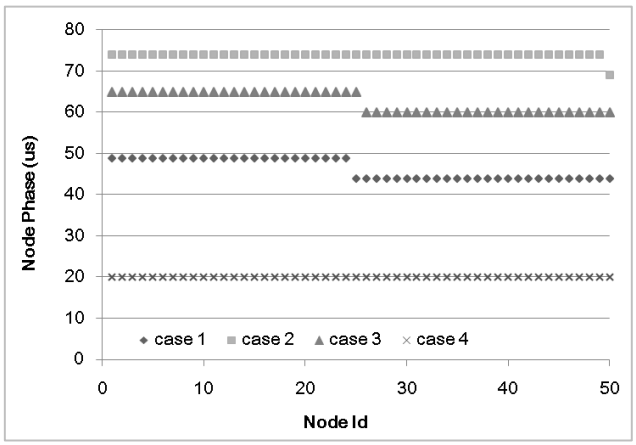


Fig. 6. Node phase combination for minimum worst-case actuation jitter

TABLE III. ACTUATION JITTERS WITH MINIMUM END-TO-END DELAY

Case	Packet	Min (us)	Max (us)
1	1st Packet	29	59
	2nd Packet	29	59
2	1st Packet	34	64
	2nd Packet	34	64
3	1st Packet	34	64
	2nd Packet	34	64
4	1st Packet	0	46
	2nd Packet	0	46

TABLE IV. ACTUATION JITTERS WITH MINIMUM ACTUATION JITTER

Case	Packet	Min (us)	Max (us)
1	1st Packet	5	15
	2nd Packet	5	15
2	1st Packet	0	15
	2nd Packet	0	15
3	1st Packet	0	10
	2nd Packet	0	10
4	1st Packet	5	26
	2nd Packet	5	26

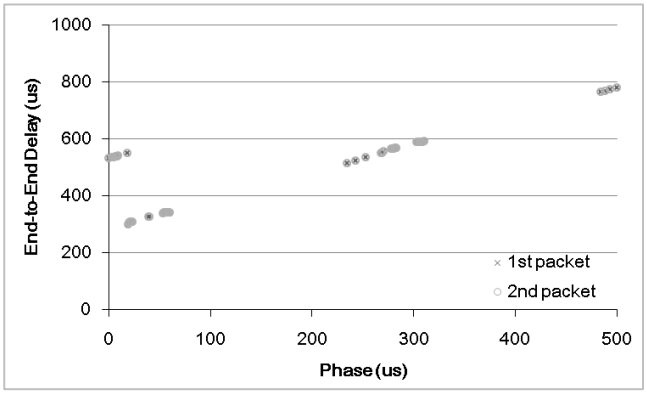


Fig. 7. End-to-end delay of interesting points on 35th slave node

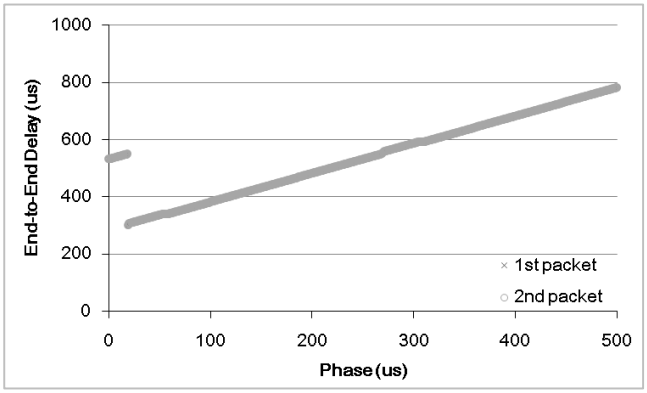


Fig. 8. End-to-end delay for 35th slave node

V. RELATED WORK

There have been several researches on simulators for industrial real-time network. Steinbach et. al. [14] have presented a simulation model for TTEthernet-based in-vehicle backbone network. They have also modified OMNet++ and the

INET framework to implement a simulator. Garner et. al. [15] have provided an overview of IEEE 802.1AS and presented simulation of AVB. They have used 8 nodes to measure the time synchronization error between a master and slave nodes. Harbour et. al. [16] have suggested a model for switched real-time Ethernet. Zeng et. al. [5] have presented a stochastic analysis framework for end-to-end latency in CAN-based distributed real-time systems. However, previous researches did not consider phasing on distributed nodes.

There are also several simulation tools for hard real-time scheduling algorithms [17][18]. We can utilize such simulators for the scheduler of the CPU emulator described in Subsection III.C to support more scheduling algorithms. To make use of these, the tools have to provide open APIs and support network task models.

It is important to reduce simulation time to find an optimal solution. For example, Garc á-Valls et. al. [19][20] proposed to selectively check for dynamic systems in real-time reconfiguration. They have used the deadline of real-time requirement to reduce the time for find optimal solution. In this paper, we try to reduce the simulation time by skipping uninteresting/predictable phasing points.

VI. Conclusions and Future Work

In this paper, we proposed a simulation framework that searched for an optimal phasing on distributed nodes with respect to the end-to-end delays and the actuation jitters, thanks to precise global synchronization of emerging real-time industrial networks, such as EtherCAT. The proposed framework considered behavior of several components composing the target system in a holistic manner to analyze the impact of node phasing on the metrics. One novelty of the proposed framework is that we try to reduce the time to find an optimal node phasing by investigating only interesting phase points where the correlation state is changed. We carried out a case study to show that the proposed simulation framework can efficiently suggest an optimal node phasing across distributed nodes in terms of end-to-end delays and the actuation jitters.

As future work, we intend to consider task phasing within a node so that we can achieve the minimum end-to-end delay or minimum actuation jitter. We also plan to apply this simulation results to a real system. Especially, we plan to target a partitioned system by extending the simulator. To do this, the simulator has to allow several tasks can access (i.e., share) the network device on the master node.

References

[1] D. Jansen and H. Buttner, "Real-time Ethernet: The EtherCAT Solution," Computing and Control Engineering, Vol. 15, No. 1, pp. 16–21, February 2004.

[2] J. Feld, "PROFINET - Scalable Factory Communication for All Applications," In Proc. of IEEE International Workshop on Factory Communication Systems (WFCS'04), September 2004.

[3] G. Cena, IC. Bertolotti, and S. Scanzio, "Evaluation of EtherCAT Distributed Clock Performance," IEEE Transactions on Industrial Informatics, Vol.8, No.1, pp. 20-29, February 2012.

[4] G. Cena, IC. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "On the Accuracy of the Distributed Clock Mechanism in EtherCAT," In Proc. of IEEE International Workshop on Factory Communication Systems (WFCS'10), pp. 43-52, May 2010.

[5] H. Zeng, M. Di Natale, P. Giusto, and A. Sangiovanni-Vincentelli, "Stochastic Analysis of CAN-Based Real-Time Automotive Systems," IEEE Transactions on Industrial Informatics, Vol. 5, No. 4, pp. 388-401, November 2009.

[6] M. Ashjaei, M. Behnam, and T. Nolte, "The Design and Implementation of a Simulator for Switched Ethernet Networks," In Proc. of International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems (WATERS'12), July 2012.

[7] K. Tindell, A. Burns, and A. J. Wellings, "Calculating Controller Area Network (CAN) message response times," Control Engineering Practice, Vol. 3, No. 8, pp. 1163-1169, August 1995.

[8] K. Tindell and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems," Microprocessing and Microprogramming, Vol. 40, No. 2-3, pp. 117-134, April 1994.

[9] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment," Journal of the ACM, Vol. 20, No. 1, pp. 46-61, January 1973.

[10] M. Farsi, K. Ratcliff, and M. Barbosa, "An introduction to CANopen," Computing & Control Engineering Journal, Vol. 10, No. 4, pp. 161-168, August 1999.

[11] S. Potra and G. Sebestyen, "EtherCAT Protocol Implementation Issues on an Embedded Linux Platform," IEEE International Conference on Automation, Quality and Testing, Robotics, May 2006.

[12] G. Prytz, "A Performance Analysis of EtherCAT and PROFINET IRT," In Proc. of IEEE international Conferrence on Emerging Technologies and Factory Automation (ETFA'08), pp. 408-415, September 2008.

[13] K. Kim, M. Sung, and H.-W. Jin, "Design and Implementation of a Delay-Guaranteed Motor Drive for Precision Motion Control," IEEE Transactions on Industrial Informatics, Vol. 8, No. 2, pp. 351-365, May 2012.

[14] T. Steinbach, H. D. Kenfack, F. Korf, and T. C. Schmidt, "An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy," In Proc. of of the 4th International ICST Conference on Simulation Tools and Techniques (SIMUTools 2011), pp. 375-382, March 2011.

[15] G. Garner, A. Gelter, and M. Teener, "New Simulation and Test Results for IEEE 802.1AS Timing Performance," In Proc. of IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, pp. 1–7, October 2009.

[16] M. G. Harbour, J. J. Gutierrez, J. M. Drake, P. Lopez, and J. C. Palencia, "Modeling Real-Time Networks with MAST2," In Proc. of 2nd International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems (WATERS'11), July 2011.

[17] Y. Lu, J. Kraft, T. Nolte, and C. Norstrom, "A Statistical Approach to Simulation Model Validation in Timing Analysis of Complex Real-Time Embedded Systems," In Proc. of 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems (WATERS'10), July 2010.

[18] P. Courbin, and L. George, "FORTAS : Framework fOr Real-Time Analysis and Simulation," In Proc. of 2nd International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems (WATERS'11), July 2011.

[19] M. Garc á-Valls, I. R. Lopez, and L. F. Villar, "iLand: An Enhanced Middleware for Real-Time Reconfiguration of Service Oriented Distributed Real-Time Systems," IEEE Transactions on Industrial Informatics, vol. 9, No. 1, pp. 228-236. February 2013.

[20] M. Garc á-Valls, A. Alonso, and J. A. de la Puente, "A Dual-Band Priority Assignment Algorithm for Dynamic QoS Resource Management," Future Generation Computer Systems, Vol. 28, No 6, pp. 902-912, June 2012.

Many-in-one Response-Time Analyzer for Controller Area Network

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

***Abstract*—The existing tools for the response-time analysis of Controller Area Network (CAN) support only periodic and sporadic messages. They do not analyze mixed messages which are implemented by several higher-level protocols based on CAN that are used in the automotive industry. We present a new response-time analyzer for CAN that supports periodic and sporadic as well as mixed messages. Moreover, it supports the analysis of the system where periodic and mixed messages are scheduled with offsets. It will support the analysis of all types of messages while taking into account several queueing policies and buffer limitations in the CAN controllers.**

I. Introduction

The Controller Area Network (CAN) [1] is one of the largely used real-time network protocols in the automotive domain. In 2003, it was standardized by the International Organization for Standardization in ISO 11898-1. It is a multi-master, event-triggered, serial communication protocol supporting bus speeds of up to 1 mega bits per second. Over 850 million CAN enabled controllers were sold in 2011 according to the CAN in Automation (CiA) [2] estimate. Over 2 billion controllers have been sold to date and most of them have been used in the automotive industry. The CAN protocol also finds its applications in other domains, e.g., industrial control, medical equipments, maritime electronics, and production machinery. There are several higher-level protocols for CAN that are developed for many industrial applications such as CAN Application Layer (CAL), CANopen, J1939, Hägglunds Controller Area Network (HCAN), and CAN for Military Land Systems domain (MilCAN).

Often, CAN is used in hard real-time systems. The system providers are required to ensure that the systems meet their deadlines. In order to provide evidence that each action by the system will be provided in a timely manner, *a priori* analysis techniques, such as schedulability analysis, have been developed by the research community. Response-Time Analysis (RTA) [3] is a powerful, mature and well established schedulability analysis technique. It is a method to calculate upper bounds on the response times of tasks or messages in a real-time system or a network respectively. RTA for CAN was developed by Tindell et al. [4] and later revised by Davis et. al [5]. This analysis and its extensions have been implemented in several tools that are used in the automotive industry, e.g., Volcano Network Architect (VNA) [6] and Rubus-ICE [7], [8]. The analysis has also served as the basis for many research projects and has been extended in a number of ways.

A. Paper contribution

There is a limitation with RTA for CAN [4], i.e., it only supports periodic and sporadic messages. It does not support the analysis of mixed messages which are simultaneously time- and event-triggered and are implemented by several higher-level protocols based on CAN that are used in the automotive industry. To the best our knowledge, there is no freely-available tool that implements the analysis of mixed messages (a commercial tool Rubus-ICE implements basic analysis of mixed messages in CAN). In this paper we present a new response-time analyzer for CAN namely MPS-CAN Analyzer (MPS stands for Mixed, Periodic and Sporadic). It supports the analysis of periodic, sporadic and mixed messages. We use the term "many-in-one" because we implement and continue to implement several extensions of RTA for CAN taking into account the following aspects:

- analysis of mixed messages;
- analysis of messages scheduled with or without offsets;
- analysis of messages having arbitrary jitter and deadlines;
- analysis of network with CAN controllers implementing different queueing policies, e.g., priority-based, FIFO-based, or both;
- analysis of network with limitations in CAN controllers, e.g., the controllers implementing abortable or non-abortable transmit buffers.

B. Paper layout

The remainder of the paper is organized as follows. In Section II, we discuss mixed transmission patterns supported by several higher-level protocols. Section III describes the related work, related tools and implemented analysis. Section IV presents the tool layout and its usability. Finally, Section V concludes the paper.

II. Mixed transmission patterns supported by higher-level protocols

Traditionally, it is assumed that the tasks queueing CAN messages are invoked either by periodic or sporadic events. If a message is queued for transmission at periodic intervals, we use the term "Period" to refer to its periodicity. A sporadic message is queued for transmission as soon as an event occurs that changes the value of one or more signals contained in

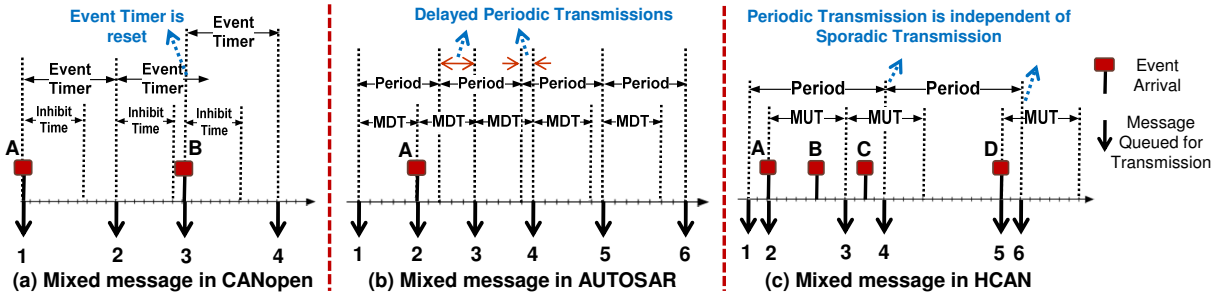


Fig. 1. Mixed transmission pattern in higher-level protocols for CAN

the message provided the Minimum Update Time (MUT[1]) between the queueing of two successive sporadic messages has elapsed. However, there are some higher-level protocols and commercial extensions of CAN in which the task that queues the messages can be invoked periodically as well as sporadically. If a message can be queued for transmission periodically as well as sporadically, it is said to be mixed. In other words, a mixed message is simultaneously time- and event-triggered. We identified three types of implementations of mixed messages used in the industry.

A. Method 1: Implementation in CANopen

The CANopen protocol [9] supports mixed transmission that corresponds to the Asynchronous Transmission Mode coupled with the Event Timer. The Event Timer is used to transmit an asynchronous message cyclically. A mixed message can be queued for transmission at the arrival of an event provided the Inhibit Time has expired. The Inhibit Time is the minimum time that must be allowed to elapse between the queueing of two consecutive messages. A mixed message can also be queued periodically at the expiry of the Event Timer. The Event Timer is reset every time the message is queued. Once a mixed message is queued, any additional queueing of it will not take place during the Inhibit Time [9].

The transmission pattern of a mixed message in CANopen is illustrated in Fig. 1(a). The down-pointing arrows symbolize the queueing of messages while the upward lines (labeled with alphabetic characters) represent arrival of the events. Message 1 is queued as soon as the event A arrives. Both the Event Timer and Inhibit Time are reset. As soon as the Event Timer expires, message 2 is queued due to periodicity and both the Event Timer and Inhibit Time are reset again. When the event B arrives, message 3 is immediately queued because the Inhibit Time has already expired. Note that the Event Timer is also reset at the same time when message 3 is queued as shown in Fig. 1(a). Message 4 is queued because of the expiry of the Event Timer. There exists a dependency relationship between the Inhibit Time and the Event Timer, i.e., the Event Timer is reset with every sporadic transmission.

[1]We overload the term "MUT" to refer to the Inhibit Time in CANopen protocol and Minimum Delay Time (MDT) in AUTOSAR communication.

B. Method 2: Implementation in AUTOSAR

AUTOSAR (AUTomotive Open System ARchitecture) [10] can be viewed as a higher-level protocol if it uses CAN for network communication. Mixed transmission mode in AUTOSAR is widely used in practice. In AUTOSAR, a mixed message can be queued for transmission repeatedly with a period equal to the mixed transmission mode time period. The mixed message can also be queued at the arrival of an event provided the Minimum Delay Time (MDT) has been expired. However, each transmission of a mixed message, regardless of being periodic or sporadic, is limited by the MDT. This means that both periodic and sporadic transmissions are delayed until the MDT expires. The transmission pattern of a mixed message implemented by AUTOSAR is illustrated in Fig. 1(b). Message 1 is queued (the MDT is started) because of partly periodic nature of a mixed message. When the event A arrives, message 2 is queued immediately because the MDT has already expired. The next periodic transmission is scheduled 2 time units after the transmission of message 2. However, the next two periodic transmissions corresponding to messages 3 and 4 are delayed because the MDT is not expired. This is indicated by the text "Delayed Periodic Transmissions" in Fig. 1(b). The periodic transmissions corresponding to messages 5 and 6 take place at the scheduled times because the MDT is already expired in both cases.

C. Method 3: Implementation in HCAN

A mixed message in the HCAN protocol [11] contains signals out of which some are periodic and some are sporadic. A mixed message is queued for transmission not only periodically, but also as soon as an event occurs that changes the value of one or more event signals, provided the MUT between the queueing of two successive sporadic instances of the mixed message has elapsed. Hence, the transmission of a mixed message due to arrival of events is constrained by the MUT. The transmission pattern of a mixed message is illustrated in Fig. 1(c). Message 1 is queued because of periodicity. As soon as event A arrives, message 2 is queued. When event B arrives it is not queued immediately because MUT is not expired yet. As soon as the MUT expires, message 3 is queued. Message 3 contains the signal changes that correspond to event B. Similarly, a message is not immediately queued when the event C arrives because the MUT is not expired.

Message 4 is queued because of the periodicity. Although, the MUT was not expired, the event signal corresponding to event C was packed in message 4 and queued as part of the periodic message. Hence, there is no need to queue an additional sporadic message when the MUT expires. This indicates that the periodic transmission of a mixed message cannot be interfered by its sporadic transmission (a unique property of the HCAN protocol). When the event D arrives, a sporadic instance of the mixed message is immediately queued as message 5 because the MUT has already expired. Message 6 is queued due to periodicity.

D. Discussion

In the first method, the Event Timer is reset every time a mixed message is queued for transmission. The implementation of a mixed message in method 2 is similar to method 1 to some extent. The main difference is that the periodic transmission can be delayed until the expiry of the MDT in method 2. Whereas in method 1, the periodic transmission is not delayed, in fact, the Event Timer is restarted with every sporadic transmission. The MDT timer is started with every periodic or sporadic transmission of a mixed message. Hence, the worst-case periodicity of a mixed message in methods 1 and 2 can never be higher than the Inhibit Timer and MDT respectively. Therefore, the existing analyses hold intact. However, the periodic transmission is independent of the sporadic transmission in the third method. The periodic timer is not reset with every sporadic transmission. A mixed message can be queued for transmission even if the MUT is not expired. The worst-case periodicity of a mixed message is neither bounded by the period nor by the MUT. Therefore, the existing analyses cannot be applied to mixed messages in the third implementation method. Further, there is no free tool that is able to analyze mixed messages that are implemented using the third method. Our main goal is to develop a free tool that analyzes periodic, sporadic and mixed messages in CAN.

III. Related Work and Implemented Analysis

A. Related work

The schedulability analysis of CAN was developed by Tindell et al. [4]. Davis et al. [5] refuted, revisited and revised the analysis by Tindell et al. In [12], Davis et al. extended the analysis in [4], [5] which is applicable to the CAN network where some nodes implement priority queues and some implement FIFO queues. The message deadlines in [12] are assumed to be smaller than or equal to the corresponding periods. In [13], Davis et al. lifted this assumption by supporting the analysis of CAN messages with arbitrary deadlines. Furthermore, they extended their work to support RTA of CAN for FIFO and work-conserving queues.

The analysis in [4], [5] assumes that the CAN controllers have very large transmit buffers. However, most CAN controllers have small number of transmit buffers [14], [13]. If all buffers in the controller are occupied by lower priority messages, a higher priority message released in the same controller may suffer from priority inversion [4], [15], [16], [17]. The analysis in [4], [5] was extended in [15] and [14] to support the analysis of network that contain abortable and non-abortable transmit buffers in the controllers respectively. Most of the CAN enabled Electronic Control Units (ECUs) are capable of aborting transmission requests [15].

All these analyses assume that the messages are queued for transmission periodically or sporadically. Mubeen et al. [18] extended the existing analysis [4], [5] to support mixed messages in CAN where nodes implement priority-based queues. Mubeen et al. [19] further extended their analysis to support mixed messages in the network where some nodes implement priority queues while others implement FIFO queues. Often, there are practical limitations in the transmit buffers of CAN controllers, e.g., buffer size, and support for abort requests [17]. RTA for mixed messages in CAN [18] was extended to support the analysis of network that contain abortable and non-abortable transmit buffers in the controllers in [20] and [21] respectively.

But, none of the analysis discussed above supports messages that are scheduled with offsets i.e., using externally imposed delays between the times when the messages can be queued. In order to avoid deadlines violations due to high transient loads, current automotive embedded systems are often scheduled with offsets [22]. The worst-case response-times of lower priority messages in CAN can be reduced if the messages are scheduled with offsets [23], [24]. A method for the assignment of offsets to improve the overall bandwidth utilization is proposed in [24]. The worst-case RTA for CAN messages with offsets has been developed by several researchers [25], [26], [23], [27], [22].

None of the above analyses with offsets supports mixed messages that are scheduled with offsets. Offset-based analysis [25] was extended in [28] to support worst-case response-time calculations for mixed messages in CAN. However, this analysis is restricted due to limitations regarding message jitter and deadlines. The source of these limitations comes from the base analysis [25]. In [29], Mubeen et al. removed these limitations and extended the analysis for mixed messages [18] with offsets [22].

B. Related tools

The Volcano Network Architect (VNA) [6] is a communication design tool that supports RTA for CAN. It implements RTA of CAN developed by Tindell et al. [4].

Vector [30] is a tools provider for the development of networked electronic systems. CANalyzer [31] supports the simulation, analysis and data logging for the systems that use CAN for network communication. CANoe [32] is a tool for the simulation of functional and extra-functional (e.g., timing) behavior of ECU networks. Network Designer CAN is another tool by Vector that is used to design the architecture and perform timing analysis of CAN.

SymTA/S [33] is a tool by Symtavision for model-based timing analysis and optimization. Among other analyses, it

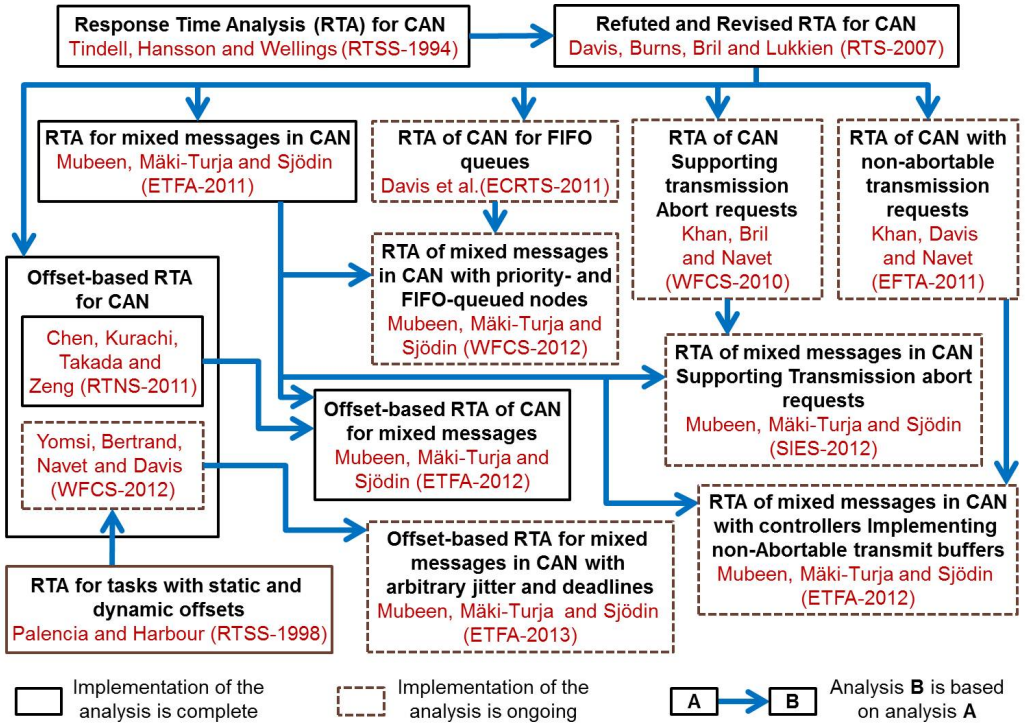


Fig. 2. Graphical representation of Response Time Analysis (RTA) and its extensions implemented in MPS-CAN Analyzer

supports statistical, and worst- and best-case timing analysis for CAN.

RTaW-Sim [34] is a tool for the simulation and performance evaluation of the CAN network.

The Rubus-ICE is a commercial tool suite developed by Arcticus Systems [7] in close collaboration with Mälardalen University Sweden. Among other analyses, it supports RTA of CAN [4], [5] and RTA of CAN for mixed messages[18].

To the best of our knowledge, there is no freely-available tool that implements RTA of CAN for mixed messages. The main purpose of MPS-CAN Analyzer is to support RTA of periodic, sporadic and mixed messages in CAN.

C. Implemented analysis

The analysis that we implemented (and continue to implement) in MPS-CAN Analyzer consists of RTA for CAN and its several extensions as shown in Fig. 2. Solid-line boxes in Fig. 2 represent the analysis that is already implemented. Whereas, the dashed-line boxes represent the analysis whose implementation is ongoing. Fig. 2 also shows the relationship among the analyses.

IV. Tool Layout and Usage

A. Tool layout, inputs and outputs

The Layout of MPS-CAN Analyzer is shown in Fig. 3. There are two windows: (1) main window denoted by "MPS-CAN Analyzer", and (2) window denoted by "New Message". The "MPS-CAN Analyzer" window is the user interface for the tool. The "New Message" window is used to create a new message. It pops up when "New Message" button is clicked on the main window. The "New Message" window and upper portion of the main window including list boxes ("Message List", "Network Speed" and "Number of Nodes") and buttons (radio, check and push buttons) represent the input section of the tool. Whereas, the lower portion of the main window including "Output", "Network Utilization", and "Errors and Warnings" list boxes comprise the output section of the tool.

In the main window, the base analysis must be selected. The base analyses is categorized based on the queueing policy in the CAN controllers, i.e., priority-based, FIFO-based, or both priority- and FIFO-based. Currently, priority-based analysis is implemented while the implementation of other two base analyses is ongoing. Once the base analysis is selected, one or more optional analysis may be selected that include (1) the analysis of messages that are scheduled with offsets, (2) the analysis supporting abortable transmit buffers, and (3) the analysis supporting non-abortable transmit buffers in the CAN controllers. Currently, the offset-aware analysis is implemented while the implementation of other two optional analyses is ongoing. The user can also specify network speed in bits per second (bps) in the main window. There are buttons provided to clear, save and load messages.

In the "New Message" window, message attributes are provided as input. For a mixed message, both period and minimum update time are specified. Whereas for a periodic or sporadic message, only period or minimum update time is specified respectively. The transmission type of a message can be selected from periodic, sporadic, or mixed. There are

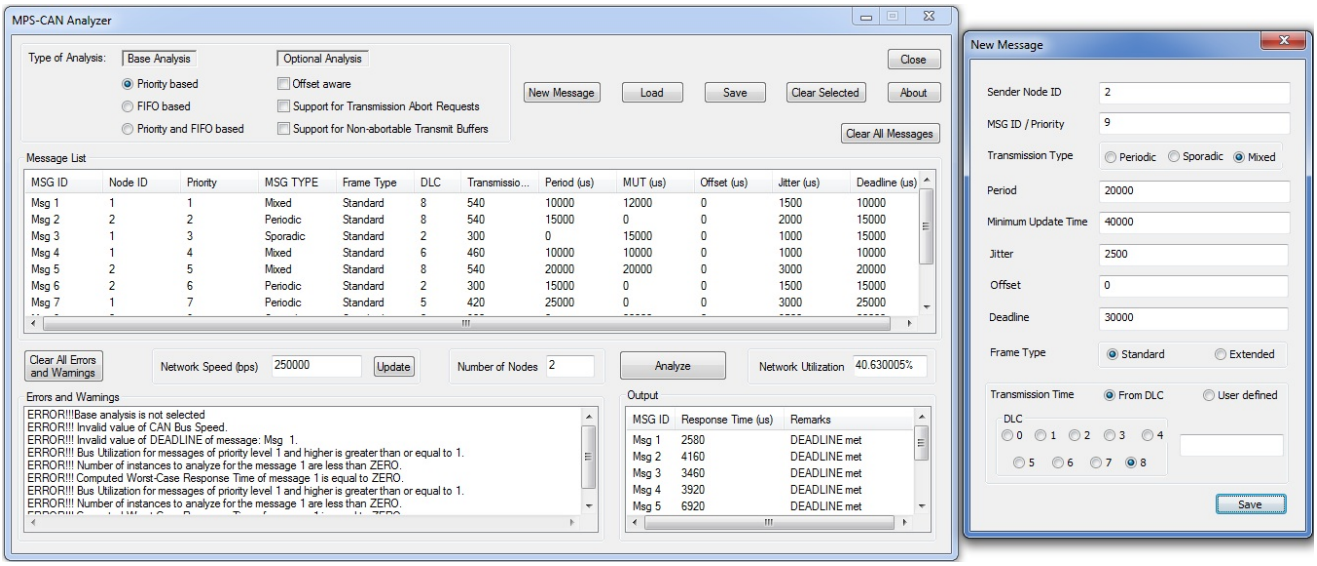


Fig. 3. MPS-CAN Analyzer layout, inputs and outputs

two options for specifying transmission type of a message. First option is based on specifying Data Length Code (DLC), i.e., the number of data bytes present in the CAN message. The second option allows to specify user-defined transmission time. This option may be used for analyzing simplified test cases that are more suitable to research-oriented work.

Any message set can be analyzed by clicking the "Analyze" button. The "Errors and Warnings" list box displays the errors and warnings. Some example errors are shown in Fig. 3. The "Output" list box displays the calculated response times of the messages. It also displays whether a message meets its deadline or not (provided the deadline is specified by the user). The percentage network utilization is also calculated and displayed in the "Network Utilization" list box.

B. Implementation and distribution

The tool is implemented in C language. Each analysis profile supported by the tool is implemented as a separate C file. The tool has a scope for further extensions in the future. The link to the development version of the tool can be found at https://github.com/saadmubeen/MPS-CAN. The implementation of some extensions of RTA is ongoing. Once, the implementation is completed, the executables and source code of the tool will be provided on the above link.

C. Example case study

We perform an example case study to show the usability of the tool. The system in the example case study consists of two nodes that are connected to the CAN network. There are 10 messages in the system out of which four are mixed, two are sporadic and the rest are periodic. All the attributes of these messages are tabulated in the Table I. The attributes of a message m are identified as follows. The priority, transmission type, transmission time, period, minimum update time, offset, jitter, deadline and worst-case response time are represented by $P_m, \xi_m, C_m, T_m, MUT_m, O_m, J_m, D_m$ and R_m respectively. All timing parameters are in microseconds. The selected speed for CAN is 250000 bps. The worst-case response times calculated by MPS-CAN Analyzer are listed in the last column of Table 1. The network utilization calculated by MPS-CAN Analyzer for this message set is equal to 40.630005%.

V. Conclusion

We introduced a new tool MPS-CAN Analyzer to support Response Time Analysis (RTA) of periodic, sporadic and mixed messages in Controller Area Network (CAN). The existing RTA tools for CAN analyze only periodic and sporadic messages. They do not support the analysis of mixed messages that are implemented by several higher-level protocols for CAN that are used in the automotive industry today.

We designated MPS-CAN Analyzer as many-in-one because it implements various extensions of the RTA for CAN taking into account mixed messages, messages scheduled with offsets, messages with arbitrary jitter and deadlines, various queueing policies (e.g., priority- or FIFO-based), and limitations in the transmit buffers in the CAN controllers (e.g., abortable or non-abortable). Some extensions are already implemented while the implementation of the rest of the extensions is ongoing. We also showed the usability of this tool by conducting the analysis of example message set.

With the implementation of these analyses, MPS-CAN Analyzer will be able to analyze network communications in heterogeneous systems (which may consist of different types of ECU's supplied by different Tier 1 suppliers). Once the implementation of all extensions of RTA for CAN is completed, we plan to conduct a detailed case study in which we will consider a heterogeneous system with ECUs implementing various queueing policies and having different buffer limitations. For example, some ECUs use priority-based queueing, some use FIFO-based queueing, and some support

TABLE I
ATTRIBUTES AND RESPONSE TIMES OF PERIODIC, SPORADIC AND MIXED MESSAGES

ID_m	Node ID	P_m	ξ_m	DLC	$C_m(\mu s)$	$T_m(\mu s)$	$MUT_m(\mu s)$	$O_m(\mu s)$	$J_m(\mu s)$	$D_m(\mu s)$	$R_m(\mu s)$
1	1	1	Mixed	8	540	10000	12000	0	1500	10000	2580
2	2	2	Periodic	8	540	15000	-	0	2000	15000	4160
3	1	3	Sporadic	2	300	-	15000	0	1000	15000	3460
4	1	4	Mixed	6	460	10000	10000	0	1000	10000	3920
5	2	5	Mixed	8	540	20000	20000	0	3000	20000	6920
6	2	6	Periodic	2	300	15000	-	0	1500	15000	6260
7	1	7	Periodic	5	420	25000	-	0	3000	25000	8180
8	2	8	Sporadic	2	300	-	20000	0	3500	20000	8980
9	2	9	Mixed	8	540	20000	40000	0	4000	20000	9740
10	2	10	Periodic	1	260	20000	-	0	1800	20000	8080

transmission abort requests while others don't. Since, this tool will be freely available, we believe, it may prove helpful in the research-oriented projects that require the analysis of CAN-based systems.

ACKNOWLEDGEMENT

This work in this paper is supported by the Swedish Knowledge Foundation (KKS) within the projects SythSoft and FEMMVA. The authors thank the industrial partners Arcticus Systems, BAE Systems Hägglunds and Volvo Construction Equipment, Sweden.

REFERENCES

[1] Robert Bosch GmbH, "CAN Specification Version 2.0," postfach 30 02 40, D-70442 Stuttgart, 1991.
[2] "Automotive networks. CAN in Automation (CiA)," http://www.can-cia.org/index.php?id=416.
[3] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings, "Fixed priority pre-emptive scheduling:an historic perspective," *Real-Time Systems*, vol. 8, no. 2/3, pp. 173–198, 1995.
[4] 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.
[5] 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.
[6] "Volcano Network Architect (VNA). Mentor Graphics," http://www.mentor.com/products/vnd/communication-management/vna.
[7] "Arcticus Systems," http://www.arcticus-systems.com.
[8] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study," *Computer Science and Information Systems, ISSN: 1361-1384*, vol. 10, no. 1, 2013.
[9] "CANopen Application Layer and Communication Profile. CiA Draft Standard 301. Version 4.02. February 13, 2002," http://www.can-cia.org/index.php?id=440.
[10] "AUTOSAR Techincal Overview, Version 2.2.2., Release 3.1, The AUTOSAR Consortium, Aug., 2008," http://autosar.org.
[11] "Hägglunds Controller Area Network (HCAN), Network Implementation Specification," *BAE Systems Hägglunds, Sweden (internal document)*, April 2009.
[12] R. I. Davis, S. Kollmann, V. Pollex, and F. Slomka, "Controller Area Network (CAN) Schedulability Analysis with FIFO queues," in *23rd Euromicro Conference on Real-Time Systems*, July 2011.
[13] R. Davis and N. Navet, "Controller Area Network (CAN) Schedulability Analysis for Messages with Arbitrary Deadlines in FIFO and Work-Conserving Queues," in *9th IEEE International Workshop on Factory Communication Systems (WFCS)*, may 2012, pp. 33 –42.
[14] D. Khan, R. Davis, and N. Navet, "Schedulability analysis of CAN with non-abortable transmission requests," in *16th IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, sept. 2011.
[15] D. Khan, R. Bril, and N. Navet, "Integrating hardware limitations in can schedulability analysis," in *8th IEEE International Workshop on Factory Communication Systems (WFCS)*, may 2010, pp. 207 –210.
[16] M. D. Natale, "Evaluating message transmission times in controller area networks without buffer preemption," in *8th Brazilian Workshop on Real-Time Systems*, 2006.
[17] Marco Di Natale, Haibo Zeng, Paolo Giusto, Arkadeb Ghosal, *Understanding and Using the Controller Area Network Communication Protocol*. Springer, 2012.
[18] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Extending schedulability analysis of controller area network (CAN) for mixed (periodic/sporadic) messages," in *16th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, sept. 2011.
[19] S. Mubeen, J. Mäki-Turja and M. Sjödin, "Response-Time Analysis of Mixed Messages in Controller Area Network with Priority- and FIFO-Queued Nodes," in *9th IEEE International Workshop on Factory Communication Systems (WFCS)*, May 2012.
[20] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Response Time Analysis for Mixed Messages in CAN Supporting Transmission Abort Requests," in *7th IEEE International Symposium on Industrial Embedded Systems (SIES)*, June 2012.
[21] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Extending response-time analysis of mixed messages in can with controllers implementing non-abortable transmit buffers," in *17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, sept. 2012.
[22] P. Yomsi, D. Bertrand, N. Navet, and R. Davis, "Controller Area Network (CAN): Response Time Analysis with Offsets," in *9th IEEE International Workshop on Factory Communication Systems*, May 2012.
[23] A. Szakaly, "Response Time Analysis with Offsets for CAN," Master's thesis, Department of Computer Engineering, Chalmers University of Technology, Nov. 2003.
[24] M. Grenier, L. Havet, and N. Navet, "Pushing the limits of can-scheduling frames with offsets provides a major performance boost," in *4th European Congress on Embedded Real Time Software*, 2008.
[25] Y. Chen, R. Kurachi, H. Takada, and G. Zeng, "Schedulability comparison for can message with offset: Priority queue versus fifo queue," in *19th International Conference on Real-Time and Network Systems (RTNS)*, Sep. 2011, pp. 181–192.
[26] L. Du and G. Xu, "Worst case response time analysis for can messages with offsets," in *IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, nov. 2009, pp. 41 –45.
[27] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the symta/s approach," *Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, March 2005.
[28] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Worst-case response-time analysis for mixed messages with offsets in controller area network," in *17th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, sept. 2012.
[29] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Extending offset-aware response-time analysis for mixed messages in controller area network," in *18th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, sept. 2013.
[30] "Vector. http://www.vector.com."
[31] "CANalyzer. http://www.vector.com/vi_canalyzer_en.html."
[32] "CANoe. www.vector.com/portal/medien/cmc/info/canoe_productinformation_en.pdf."
[33] A. Hamann, R. Henia, R. Racu, M. Jersak, K. Richter, and R. Ernst, "Symta/s - symbolic timing analysis for systems," 2004.
[34] "RTaW-Sim. http://www.realtimeatwork.com/software/rtaw-sim."

On the Convergence of Experimental Methodologies for Distributed Systems: Where do we stand?

Maximiliano Geier*†, Lucas Nussbaum† and Martin Quinson†
* Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires (C1428EGA), Argentina
† Inria, Villers-lès-Nancy, F-54600, France
Université de Lorraine, LORIA, F-54500, France
CNRS, LORIA - UMR 7503, F-54500, France

***Abstract*—Understanding distributed systems is a complex task. There are many subsystems involved, such as network equipment, disk and CPU, which effect behavior. In order to analyze this kind of applications, different approaches have been proposed: simulation, emulation and experimentation. Each paradigm has evolved independently, providing their own set of tools and methodologies.**

This paper explores how these tools and methodologies can be combined in practice. Given a simple question on a particular system, we explore how different experimental frameworks can be combined in practice. We use a representative framework for each methodology: Simgrid for simulation, Distem for emulation and Grid'5000 for experimentation. Our experiments are formally described using the workflow logic provided by the XP Flow tool.

Our long term goal is to foster a coherent methodological framework for the study of distributed systems. The contributions of this article to that end are the following: we identify a set of pitfalls in each paradigm that experimenters may encounter regarding models, platform descriptions and others. We propose a set of general guidelines to avoid these pitfalls. We show these guidelines may lead to accurate simulation results. Finally, we provide some insight to framework developers in order to improve the tools and thus facilitate this convergence.

I. Introduction

Distributed systems are pervasive in many areas of computing, ranging from scientific applications to content distribution systems. Many of these systems, such as peer-to-peer networks, comprise millions of nodes, distributed all over the world. These are generally highly heterogeneous systems, in which many different subsystems and technologies interact simultaneously using common protocols. It has been a running effort since decades to assess the properties of these systems, such as reliability, resilience, performance or security. Most often, researchers rely for that on experimentation: they analyze the behavior by running the system under a particular scenario and capturing output data that could be of interest.

Experimental work in distributed systems could be categorized in three different paradigms [1], [2]:

- **Simulation**: a prototype of the application is executed on top of a model of the environment. This approach enables the researcher to analyze questions about the system without having access to the actual environment or the actual application. The reliability of the results depend on the validity of the underlying models.
- **Emulation**: the actual application is executed on a simulated environment. The environment can be controlled through classical virtualization techniques, or a controlled artificial load can be injected onto real resources such as network links and CPUs according to the given experimental scenario.
- **Experimentation**: the actual application is executed on a real environment. Although it might be desirable, it is not always possible to do this, as it requires access to an instrumented platform that matches the real environment. This might be prohibitively expensive or not available. Moreover, testing on different scenarios under these circumstances can turn into an incredibly complex task.

Each paradigm offers its own set of tools and methodologies. Most of these tools have evolved independently from each other. It is then difficult to combine them for an augmented analysis.

This paper constitutes a status report regarding the emergence of a coherent experimental framework combining these different approaches. We opted for a practical evaluation where we conduct an experimental analysis leveraging these methodologies, and report on the difficulties encountered. To that extend, we analyze a file broadcasting application that is widely used in a cluster setting. Our focus remains however on the methodological aspect of this study, not on the results of this study per se.

This paper is organized as follows: Section II first introduces the related work while Section III describes the proposed methodology. Section IV presents several experimental traps arising in the different methodological paradigms and hints on how to avoid these traps to get satisfying results. Section V discusses several considerations that tool designers must address to facilitate the methodologies convergence. Finally, Section VI concludes this paper.

II. Related work

Several works combine differing experimental methodologies in a coherent framework toward augmented analysis. **The Emulab-Planetlab portal** [3] allows to use the interface of the Emulab [4] emulator to access the Planetlab [5] experimental platform, clearly bridging the gap between these instruments. **EMUSIM** [6] is another interesting attempt in this regard. It

integrates emulation and simulation environments in the domain of cloud computing applications, providing a framework for increased understanding of this type of systems. Similarly, **Netbed** [7] is a platform based on Emulab that mixes emulation with simulation to build homogeneous networks using heterogeneous resources.

To the best of our knowledge, there is however not much previous work that compares these methodologies in practice.

The work in [8] analyzes "myths and beliefs" about Planetlab as it stood in 2006. It concentrates on debunking assumptions about the platform that were either never correct or simply no longer true at that point. Moreover, it is clear with regard to stating real limitations of the platform, so as to help users decide if it is reasonable to use it for their objectives. This approach is different to our work in the fact that its conclusions are a set of best practices for users of a single platform. It does not analyze how Planetlab plays with other platforms and does not try to construct an unified methodology for the analysis of distributed systems.

III. Methodology

A. Selected Tools

Although certainly interesting, including in this study every existing experimental tool or framework would be a daunting task. In this work, we preferred focusing on one representative framework per methodological paradigm and focus on the methodological aspects. Additional conclusions would have been reached with other tools, but we believe that this does not reduce the impact of our conclusions.

Concerning **simulation**, we used SimGrid [9]. This is an ever growing simulation framework, with more than 10 years of development and over 100 papers based on its results. It features sound and scalable models of distributed systems. **Direct experiments** were run on Grid'5000 [10]. As of May 2013, this platform consists of 11 sites all over France and Luxembourg, with several clusters on each site, connected by high speed links. This scientific instrument is dedicated to the live study of distributed systems. To this end, it allows full deployment of custom operating system on the reserved nodes and the reservation of isolated network portions. We chose the Distem [11] **emulator**. It is easily deployed on Grid'5000 nodes, and enables the experimenter to simulate network topologies. Nodes are deployed as Linux containers, meaning that many virtual nodes could be instantiated in a single physical node, with a small resource overhead. The platform is simulated by slowing down physical links artificially (network or memory bus).

B. Driving Question

The driver of any experimental analysis is usually an interesting question that researchers are trying to answer. As a consequence, the methodology is often an afterthought, and the contribution quality comes from the results found.

This paper is rather different in that regard. As we focus on the methodology itself, we base our analysis on a simple question. It was not chosen to be innovative but instead to be simple enough to not distract the study while being complex enough to mandate non-trivial experiments. This driving question is to evaluate the relative advantages of different chain propagation algorithms for file broadcasting in a cluster setting.

One such algorithm is already implemented in the Kastafior tool of the Kadeploy suite [12]. It is used in production on Grid'5000 to deploy user OS images to each node.

It works as follows: an efficient communication chain is built to connect all participating nodes to their network neighbor when possible and to reduce the transfer interactions on the chain. This chain can be built semi-automatically since the network topology is well documented on this instrument. The file is then split in fixed size chunks, that are sent sequentially from the broadcaster to the first node of the chain. As soon as the first chunk is received by the first node, it is forwarded to the next one in the chain, concurrently to the reception of the second chunk. This algorithm is intended to minimize the time to send the whole file to all participating nodes by overlapping as much communication as possible while avoiding network interactions. In taking advantage of the network topology, this algorithm can be used to deploy files very efficiently, without using multicast or other advanced tools which introduce a huge administration overhead.

Kastafior was never analyzed thoroughly, but it performs well in practice for the users of the Grid'5000 infrastructure.

IV. Observed Traps

In this section, we show some of the problems that an experimenter might run into when analyzing a distributed system. We encountered these issues while analyzing our own implementation of the Kastafior algorithm in the context of broadcasting files in a single Grid'5000 site. This implementation, called *chainsend*, has been written entirely in C.

The metric of interest in our study is the bandwidth per node, *i.e.* the average of all bandwidths. It is measured in every node as the amount of time to *receive* the complete file divided by the file size. This value is then averaged over all nodes to get a single value. This metric has been produced by measuring time in every node. In direct experimentation in Grid'5000 and in Distem we use the local clock of each node. In the case of Simgrid, we use the simulated time provided by the framework. We show this metric as a function of the number of nodes, to indicate the progression. The data points have also been interpolated using splines to ease visualization.

Our experiments leverage up to 100 nodes of the Nancy site of Grid'5000 (clusters *griffon* and *graphene*), up to 10 virtual nodes in Distem, and 92 nodes in Simgrid (*griffon* platform file). The file size used is 1 GiB.

A. Difficulties getting the platform right

The platform in which the experiment runs tells us about network size and characteristics, how nodes are connected to each other and all the information that is required to reproduce a similar setup. However, for non-trivial experiment sizes, it could become increasingly difficult to ensure that it represents exactly what the user wanted.

In the case of Simgrid, it is important to understand the platform syntax correctly. The description is given in the form of an XML file, but it is tedious to write explicitly. In order to simplify it, some syntax shortcuts have been put in place (e.g. the `cluster` tag) which alleviate platform writing significantly, but could introduce errors if the user does not understand what they and their attributes mean exactly.

Finally, as said before, another important issue is accuracy: it is possible that the description is correct in terms of what the user wanted to say, but wrong with respect to the reference platform. This problem is significant as it could induce false conclusions from the experiments. For example, in a bandwidth-limited experiment, if the platform description is wrong with respect to this metric, it is obviously not possible to reproduce a result in simulation even though the underlying model may be correct otherwise. Latency parameters also effect metrics in unexpected ways (e.g. delaying communication and thus reducing overall bandwidth usage) and it may not be easy to identify this problem in the platform description.

Distem also shows similar problems: it is not possible to map Simgrid platform files to Distem platform descriptions yet. This is a known issue, and even though it is able to load simple *v2* Simgrid platform files correctly, most advanced features are not working yet. Moreover, there is also the non-trivial problem of mapping a virtual platform on a set of physical nodes. For example, using up several physical nodes imposes limitations on inter-node bandwidth and communication time that have to be taken into account when designing experiments. Lastly, Distem has some limitations on what kind of routing it can emulate, thus rendering impossible, for instance, to experiment with redundantly connected nodes.

B. Missing hardware models in simulation

Both simulation and emulation abstract away some details which can be found in the real platform. These abstractions could lead to wrong or inaccurate results if they are not correctly accounted for. It is important to understand them to design good experiments.

Simgrid's MSG API, for instance, forces some restrictions on what kind of software can be accurately simulated. It should be noted that, as a result of this, applications written using this API cannot be implemented as if they were actual network applications.

Simgrid simulates only a network and a CPU, the latter only given that appropriate CPU parameters were input to the model. This implies that, for example, it doesn't simulate disk activity, therefore a disk-bound application would not produce the expected results in simulation.

In Figure 1, we show a comparison of the bandwidth per node in Grid'5000 by measuring the transfer time at two different points: as soon as the file has been transferred (before the `fsync` system call is issued), and after the data has been successfully written to disk (after fsync). As we can see, *chainsend* is a disk-bound application, and as such we cannot make a fair comparison against the Simgrid implementation unless we take the disk out of the picture. Even if we measure time after fsync, there is still disk activity going on that could affect the results. In order to mitigate this, all the following runs in Grid'5000 write the file to the special device `/dev/null`. This device discards data without writing anything to disk. We show the results of doing so in Figure 2. As we can see, the results are much more stable now. The reason behind this is that there is no interference of the I/O cache, as writes are properly discarded. This scenario is more realistic compared to what is actually simulated by Simgrid.

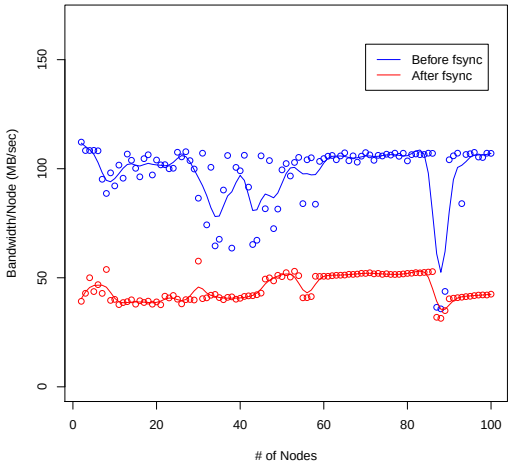


Figure 1. Bandwidth per node in Grid'5000 before fsync and after fsync.

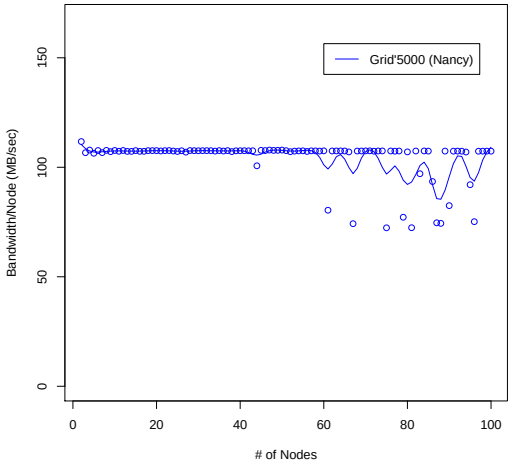


Figure 2. Bandwidth per node in Grid'5000 after discarding writes to disk.

C. Bad assumptions behind network models

Simgrid provides a network model for the simulation. This model makes some assumptions that have to be taken into account to make good use of the platform. They could be categorized as follows:

- Transport protocol: MSG assumes that all its communi-

cations are handled using TCP, ruling out every other transport protocol.

- Connection flow: the mailbox abstraction in MSG assumes that every task takes up its own connection, meaning that for each send, it simulates the time it takes to open a socket, do the three-way handshake, send data, receive its respective ACKs, and finally close it. This also implies that it can't simulate a continuous stream of data, unless it is sent as a single task.
- Connection flow arguments: there are two parameters in the default network model (LV08 [13]) that can be adjusted to change the behavior explained above, `bandwidth_factor` and `latency_factor`. The first one effects what percentage of the total bandwidth can actually be consumed by the connection, while the latter is a latency penalty factor, that effects how much time it takes to go from "slow start" to a "steady state". If this factor is closer to 1.0, the "slow start" effect is less noticeable. This factor could be used to simulate a stream of continuous data more accurately, but it is necessary to adjust it beforehand (i.e. it is not dynamically adjustable).

To highlight the effects of the connection flow parameters in Simgrid, we show bandwidth per node in Grid'5000 compared against Simgrid in Figure 3, using the default network model and connection flow arguments. As we can see, performance in Simgrid is roughly half of that in the real platform. This can be explained by what we have said before: in MSG, Simgrid simulates a complete connection for each send/receive. This means that for each file fragment being sent, there is a three-way handshake, data being sent on the network, and finally the connection is closed. At the same time, the congestion control algorithm in TCP takes place. This results in slow start taking place for every chunk of the file being sent. This is unrealistic compared to the real application, as the whole transfer happens during a single TCP connection.

If we change the default network model to CM02, which is a much simpler model that doesn't simulate accurately all the TCP congestion control algorithms, we obtain the results that are shown in Figure 4.

As we can see, the results for Simgrid are very similar to those of Grid'5000 now. This simpler model lets us fake the fact that Simgrid doesn't correctly account for single streams of data over multiple messages. In this case, it works as if there was no slow start at all, meaning that the stream is continuously in "steady state". This is very similar, although slightly overfitting, to the actual situation.

Moreover, there is an adjustable parameter, `SG_TCP_CTE_GAMMA`, which modifies how Simgrid takes into account the TCP congestion window when it updates the simulated time. This parameter can also be found in real TCP implementations (in Linux, it is possible to modify it on runtime by writing to the files `/proc/sys/net/ipv4/tcp_rmem` and `/proc/sys/net/ipv4/tcp_wmem`). These, among many other parameters, change behavior in actual TCP implementations and it is clear that not all of them behave

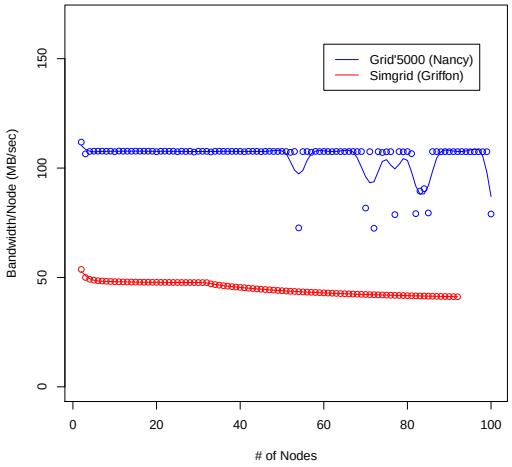


Figure 3. Bandwidth per node in Grid'5000 vs. Simgrid (default network model and arguments).

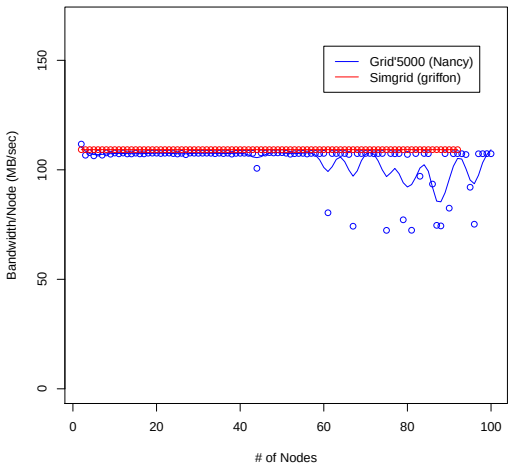


Figure 4. Bandwidth per node in Grid'5000 vs. Simgrid (CM02 network model).

the same way in practice. As a consequence of this, it is very difficult to decide on which implementation to follow as the "right one", as there are so many of them on the Internet.

D. Physical node mapping interference

Distem, on the other hand, has a different set of issues arising from its abstractions. In a Distem network, the network is presented as an overlay, the underlying network and the physical nodes are hidden from the application. This derives in several constraints that have to be carefully analyzed.

One of them is the node mapping: if the overlay is emulated completely on top of a single node, memory bandwidth and system software act as a bandwidth barrier which can't be overcome. If the mapping is 1 physical node $\leftrightarrow$ 1 virtual node, the barrier exists in the network equipment that connects the

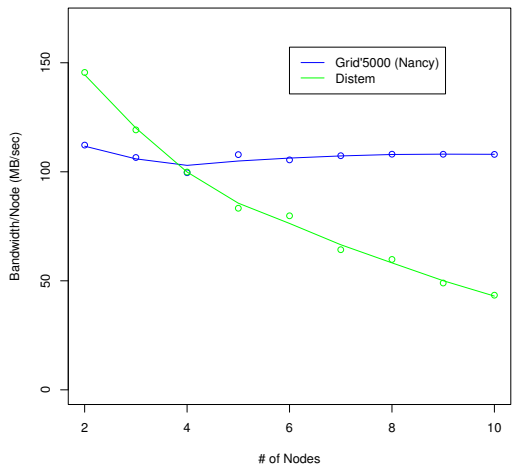


Figure 5. Bandwidth per node in Grid'5000 vs. Distem (ten virtual nodes in one physical node).

nodes to each other. A simple example in this case would be that it is not possible to connect virtual nodes on a Gigabit virtual network, if the underlying physical nodes are connected at 100 Mbps. In the same sense, it is also not possible to connect 100 virtual nodes on a single physical node at 1 Gbps, if memory bandwidth at that node is only 10 Gbps, and expect consistent results. In order to visualize this problem, we can see in Figure 5 the bandwidth per node going down as the number of virtual nodes increases, when the mapping is done over one single physical node. The expected bandwidth should be much higher, but it is limited by the way kernel buffers are managed. Test runs using improved *netns* and *veth* kernel subsystems yield a higher maximum bandwidth.

Finally, bandwidth is not the only limiting factor, also emulated latencies should be higher than those of the underlying network. As an example of this, it is not possible to emulate a link with a latency of 1 μs over one with a real latency of 1 ms. Also, if both latencies are the same order of magnitude, it is very likely that there is interference from the outside network.

V. Discussion

This section discusses the traps identified in the previous section and propose some recommendations and improvements both for the experimenters and for the tool designers.

A. Platform convergence

Our first conclusion on the platform issues is that the tools should enable users to converge to the platforms used by the different approaches. In our scenario, we have three sets of platforms: Simgrid platform files, Distem platform descriptions and real network testbeds. In order to compare results among all the tools, it is necessary that the platforms represent the same scenarios, otherwise the comparison would be unfair.

The tools could help the user achieve this by checking whether they match or not. For example, in the Emulab [4] testbed, which offers a subset of the features found in Distem as used on top of Grid'5000 (namely network topology emulation by means of slowing down fast links), there is a tool called *linktest* [14], that measures link latency and bandwidth after the platform has been instantiated, in order to identify differences with respect to the platform description. This tool is useful not only to corroborate that the experimental framework is able to reproduce the input scenario, but also to ensure that the user didn't make a mistake when they designed the platform model. For example, one such link measurement tool could be run in the real platform, in the platform as emulated by Distem, and finally a simulation of the same tool on top of Simgrid, and then all the results could be compared for statistically significant differences.

Similarly, being able to use the same platform descriptions in both Simgrid and Distem would be very useful to avoid error-prone work duplication. There is still the problem of handling virtual node mappings in Distem, which is a feature that doesn't make sense in the context of Simgrid, but being able to convert from Simgrid platforms to Distem, while keeping the ability to load Simgrid platforms would be a good start. As said before, there is already work in this direction, but it still needs some refinement.

B. Identify application traits

To get the most out of the tools, it is also necessary to understand what kind of application is being analyzed. Although it might seem like a chicken-and-egg problem to have to analyze a system in order to build a better analysis for it, the user has to understand very clearly what kind of application they are working on, as to properly identify traits that would not be conveyed by the experimental framework. This means, among other things, understanding what kind of traffic the application generates, what transport protocol is used in the real implementation, what is the application protocol like, what kind of network is targeted by the application, what kind of resources could act as bottlenecks.

Most of these questions require a good understanding of the application that is being analyzed and in some cases it is enough to analyze the source code to answer them. In case it is not that easy to infer, building experiments with the framework limitations in mind is a good start.

C. Converge experimental evaluation

Tools for experiment management are great improvements in terms of being able to automate experiment setup, execution and data gathering and analysis. We have used XP Flow to build all of our experiments. This tool provides an interface to work with Grid'5000 and handle all the steps of the experiment directly. The experiments themselves are structured using workflow logic, such as that used in business to describe processes.

Building experiments by hand has many problems: it is error-prone, it doesn't scale, it becomes increasingly difficult

to manage all the data as the experiment size and number grow. Working with several experimental tools expands this problem manyfold, as there is a new dimension that has to be taken into account to match similar experiments using different tools. One big problem in dealing with different tools is that they all have their own way to do things. This is worsened by the fact that there are also many tools for experiment automation. Using several tools might require writing boilerplate code and wrappers to match the way each tool works.

In order to converge to a single framework to build experiments, this code should be clearly modularized to abstract away the differences in a way that they can be written generically.

VI. Conclusions

We have evaluated three different platforms for distributed system analysis, one for each of the paradigms in experimental evaluation, concentrating on the methodological aspects. Our driving question has been the performance of chain propagation algorithms in a cluster setting. We have discussed traps that users of these platforms might run into and provided some insight on how to avoid them. In particular, we have identified problems in the accuracy of platform descriptions, missing hardware models, incorrect assumptions in network models provided by these frameworks and communication interference due to assumptions with regard to node mapping in overlays. By pointing out these problems, we have been able to create experiments to correctly assess the performance of our *chainsend* application in simulation. There is ongoing work to show the full picture, including also results in emulation. Our assessment in this case is that, due to the nature of this particular workload, emulation is not able to provide an accurate view while getting the full benefits of the platform.

Finally, an interesting topic to carry our work forward is platform validation. We plan to provide a platform validation tool similar to *linktest* for each paradigm, and also work towards the convergence of platform descriptions, in order to make it easier for the experimenter to use different tools.

It is our belief that each of the methodologies provide an unique point of view that has to be complemented in order to acquire a better understanding of the system. It is important to understand the tools and their limitations in order to use them appropriately. Our suggestions move towards making it more straightforward to manage experiments and compare results, trying to reduce repetitive, error-prone steps as much as possible.

Acknowledgments

M. Geier has a CONICET PhD fellowship. This work was supported by Inria.

References

[1] J. Gustedt, E. Jeannot, and M. Quinson, "Experimental Methodologies for Large-Scale Systems: a Survey," *Parallel Processing Letters*, vol. 19, no. 3, pp. 399–418, 2009. [Online]. Available: http://hal.inria.fr/inria-00364180

[2] O. Beaumont, L. Bobelin, H. Casanova, P.-N. Clauss, B. Donassolo, L. Eyraud-Dubois, S. Genaud, S. Hunold, A. Legrand, M. Quinson, C. Rosa, L. Schnorr, M. Stillwell, F. Suter, C. Thiery, P. Velho, J.-M. Vincent, and J. Won, Young, "Towards Scalable, Accurate, and Usable Simulations of Distributed Applications and Systems," INRIA, Rapport de recherche RR-7761, Oct. 2011. [Online]. Available: http://hal.inria.fr/inria-00631141

[3] K. Webb, M. Hibler, R. Ricci, A. Clements, and J. Lepreau, "Implementing the emulab-planetlab portal: Experience and lessons learned," in *In Workshop on Real, Large Distributed Systems (WORLDS)*, 2004.

[4] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," Boston, MA, Dec. 2002, pp. 255–270.

[5] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 3–12, Jul. 2003. [Online]. Available: http://doi.acm.org/10.1145/956993.956995

[6] R. N. Calheiros, M. A. Netto, C. A. De Rose, and R. Buyya, "Emusim: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications," *Software: Practice and Experience*, vol. 43, no. 5, pp. 595–612, 2013. [Online]. Available: http://dx.doi.org/10.1002/spe.2124

[7] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 255–270, Dec. 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844152

[8] N. Spring, L. Peterson, A. Bavier, and V. Pai, "Using planetlab for network research: myths, realities, and best practices," in *IN PROCEEDINGS OF THE SECOND USENIX WORKSHOP ON REAL, LARGE DISTRIBUTED SYSTEMS (WORLDS)*, 2006, pp. 17–24.

[9] H. Casanova, A. Legrand, and M. Quinson, "Simgrid: a generic framework for large-scale distributed experiments," in *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*, ser. UKSIM '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 126–131. [Online]. Available: http://dx.doi.org/10.1109/UKSIM.2008.28

[10] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard, "Grid'5000: A large scale and highly reconfigurable grid experimental testbed," in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, ser. GRID '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 99–106. [Online]. Available: http://dx.doi.org/10.1109/GRID.2005.1542730

[11] L. Sarzyniec, T. Buchert, E. Jeanvoine, and L. Nussbaum, "Design and Evaluation of a Virtual Experimental Environment for Distributed Systems," in *PDP2013 - 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, Belfast, Royaume-Uni, Aug. 2012, rR-8046 RR-8046. [Online]. Available: http://hal.inria.fr/hal-00724308

[12] E. Jeanvoine, L. Sarzyniec, and L. Nussbaum, "Kadeploy3: Efficient and Scalable Operating System Provisioning," *USENIX ;login:*, vol. 38, no. 1, pp. 38–44, Feb. 2013.

[13] P. Velho and A. Legrand, "Accuracy study and improvement of network simulation in the simgrid framework," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, ser. Simutools '09. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 13:1–13:10. [Online]. Available: http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5592

[14] D. Anderson, M. Hibler, L. Stoller, T. Stack, and J. Lepreau, "Automatic online validation of network configuration in the emulab network testbed," in *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, 2006, pp. 134–142.

Tropos For Embedded Real-time Control System Modeling and Simulation

Nesrine Darragi, Philippe Bon, Simon Collart-Dutilleul, El-Miloudi El-Koursi
Univ Lille Nord de France, IFSTTAR, ESTAS
20 RUE ELISEE RECLUS,
BP 70317, F-59666 VILLENEUVE D'ASCQ, FRANCE
Email: nesrine.darragi@ifsttar.fr, philippe.bon@ifsttar.fr, simon.collart-dutilleul@ifsttar.fr, el-miloudi.el-koursi@ifsttar.fr

***Abstract*—Simulation is the imitation of a system or a process in order to manage the complexity of simulated system or to optimize its performance. This paper presents a agent-based strategy of modeling and simulation. We introduce some modeling methodologies in order to determine the most adequate technique to deal with embedded control systems. We also introduce the Tropos and Agentology methodologies by describing used concepts and how they are integrated with the current stages of Tropos and Multi-agent System methodology. The above is illustrated using an embedded real-time control system as a case study.**

I. Introduction

There are many reasons to make use of formal methods in embedded real-time systems, particularly in railway signalling systems. These includes safety-criticality and complex real-time constraints. The EN50128 guidelines issued by the European Committee for Electrotechnical Standardization is a series of safety requirements for railway control. It contains recommendations based on the criticality, complexity and temporal behavior of the system. It does provide neither an exact procedure nor a unique methodology for the development of embedded critical systems.

Safety is the property which asserts that nothing bad happens in contrast to liveness which asserts that eventually a good thing happens. In our works we focus especially on safety properties and we try to prove that *error* or *fault* states are not reachable for every possible execution.

The aim of our models is to formalize and to anticipate these *dangerous* states which could be categorized as to deadlocked states or erroneous behaviour. It is a sort of future system validation. Starting with the analysis of Functional Requirement Specifications (FRS) which is a set of finite complex or simple sentences describing the behavior of the system, we could capture knowledge divided into two categories. The first one is the domain specific knowledge describing domain properties or assumptions. The second category is the commonsense knowledge which regroups the goals of stakeholders and the system actors.

The paper is organized into sections. After a short introduction, section 2 describes a state of the art of agent-based modeling methodologies. Section 3 is devoted to present the modeling and simulation strategy based on agentology methodology. The next section presents the case study and the application of our approachs.

II. State of the art

An agent-based modeling noted ABM [1] is an approach issued from the artificial intelligence which represents a separate technology. ABM is the opposite of the principle of discrete event or continuous simulation approaches which are destinated to simulate systems by simulating its tasks and *events* using state variables dependencies. Unlike them, ABM simulates systems by defining caracteristics and behaviors of system entities without specifying internal rules. The need of an agent-oriented methodologies is justified by the system complexity as well as the weakness of existing methodologies to support many emergent system characteristics such as dynamicity, cooperation aspects, distribution of software entities and temporal constraints.

Agent-oriented methodology is used to reduce the gap between existing software frameworks dedicated to the implementation of multi-agent systems (MAS) and, more precisely, agent-based simulation and methodologies and guidelines.

The current work is devoted to this approach. We are dealing with systems where we know the caracteristics of every elementary subsystem or component and we are interested specifically in the whole behavior of the system. Agents which are used to simulate these different components communicate and react with each other and with their environment in order to achieve their goals.

We established a comparison study between diverse MAS methodologies in order to choose the most appropriate one. The approach has to cover the development life-cycle of software from analysis to implementation. It is recommended that the chosen methodology heavily based on requirements engineering and influenced by the gaol approach which is our initial conceptual modeling and simulation prerequisite.

In general, modeling methodologies are influenced by approaches such as object oriented approach (Rational Unified Process for example), aspect-oriented approach or GORE [5] for *Goal Oriented Requirement Engineering* like *Keep All Objectives Satisfied* KAOS [10] or *Intentional STrategy Actor Relationships* (i*)[3].Table I shows how methodologies for MAS development are influenced by software development approaches and identified dependencies exist between these methodologies and adapted methods. The table is divided into three parts: MAS methodology, requirements engineering and object-oriented approach. For each methodology we indicate the approach or the method-driven MAS methodology in question.

TABLE I. INFLUENCE OF SOFTWARE DEVELOPMENT APPROACHES ON MULTI-AGENT SYSTEMS METHODOLOGIES

MAS Methodology	Requirements Engineering	Object Oriented
ADELFE [4]		Rational Unified Process
GAIA [1]		Agent-Oriented Methodology
INGENIAS [11]		Rational Unified Process
PASII [12]		Pattern for Agent
SODA [13]		AgentOriented Methodology
Tropos [2]	i* [3]	

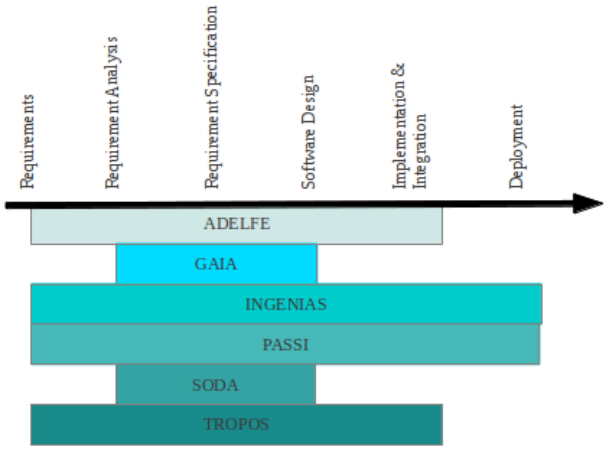


Fig. 1. Phases of the development process covered by different Multi-agent Systems development methodologies

In this work, we need to model and simulate MAS. Figure 1 shows different MAS development methodologies and software development processes they . For example Tropos, ADELFE, INGENIAS [11] or PASSI are used to analyse, to specify, to design and to implement MAS. Tropos methodology [2] responds to our needs in terms of development phases and approaches. It is an agent-oriented methodology for the analysis and design of software from early requirements to late requirements analysis right through to the building of a system model. It is established using goal-based concepts issued from i* and goal-oriented requirements language dedicated to non-functional requirements, Tropos uses other AUML [14] diagrams to deal with MAS aspects. The framework iSTAR or i* defines system agents as entities where relationships are based not only on shared data or informations but also on its common goals, beliefs or abilities.

Entities of Tropos include actors, goals (the strategic interests of agents), capabilities (the ability of an agent to decide the action to execute given its perceptions of its environment and external events), plans (a description of how an agent is able to reach a goal), dependencies (relationships between agents), beliefs (agent's knowledge related to its environment), resources, and finally contributions (a metric to evaluate the relationship between goals, plans or resources specifying if it is beneficial to achieve these goals or not).

Figure 2 shows the Tropos metamodel of the agent concept which inherits from the actor concept. An agent plays the roles and occupies a position. An actor is an agent or a role. An actor has goals and plans. It is dependent on other actors with which it may share resources and goals. The latter is

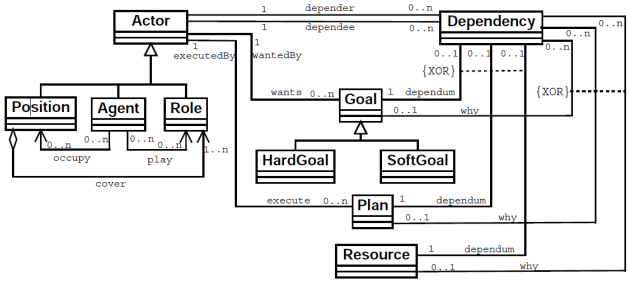


Fig. 2. Tropos metamodel of actor concept

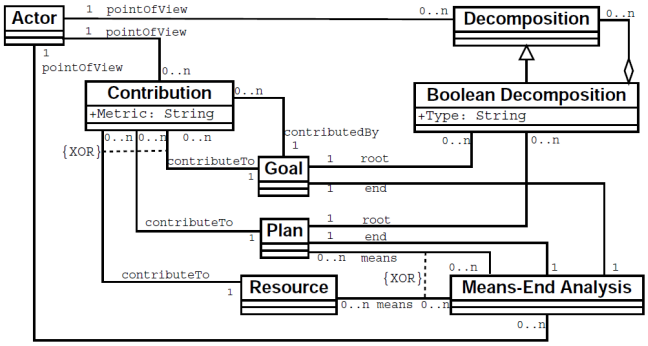


Fig. 3. Tropos metamodel of goal concept

the generalisation of soft and hard goals. The first category is intentions of non-functional requirements in general. The second one is the aim behind the functional requirements.

Figure 3 illustrates the Tropos metamodel of goal concept. An actor has many points of view of its contribution to achieve its goal. This latter may be decomposed by other refined sub-goals in another level of abstraction. A "And" or an "Or" relationships may regroup many sub-goals in order to fulfil the global or the root goal. It has a means or satisfaction.

The methodology of Tropos may be summerized in 5 phases :

- Early Requirement phase : Identification and modeling of stakeholders as social actors. Dependencies of these based on common goals, plans and shared resources are identified.
- Late Requirement phase : Identification of system actors and software agents and the dependencies between them.
- Architectural Design : Definition of subsystem architectures and their connections through data and control flows.
- Detailed Design : Specification of the characteristics and capabilities of each agent.
- Implementation : Implementation of the system detailed design.

III. MODELING AND SIMULATION STRATEGY

Inspired from the Watterfall model [8] but also by best practices contained in several other methodologies ,the agen-

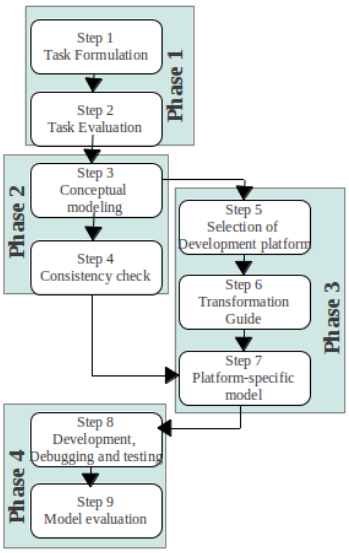


Fig. 4. Process Agentology

tology [6] is a methodology for agent-based modeling. The methodology is independant of any specific technology, programming language or excecution environments. The basic idea of this kind of simulation is Composed of 4 phases and 9 steps, as shown in figure 4. The first phase is the requirement definition wchich consists on formulation and evaluation of requirements or tasks. The second phase is the conceptual modeling. It is composed by two steps; modeling and consistency checking of models.The third phase id the platform-specific modeling which concerns the selection of a development platform in addition to definition of transformation guide and the last step is platform-specific modeling. Finally, simulation modeling which is the fourth pahse. It is dedicated to development, debugging, testing and model evalution. This methodology attempts to offer guidelines to assist the modeling of the system using of one of the most difficult approaches which is agent-paradigm. We will use the ontology as a platform-independent methodology to provide an agent-based model as a means to simulate this model in subsequent steps.

The goal of this work is the simulation of the behavior of subsystems and their interactions under hazardous conditions and to detect possible defects and software faults. We simulate the interaction of subsystems and components considered to be system actors within a real-time execution-like environment. Each one of our actors has specific characteristics such as capabilities, responsibilities, goals, plans and resources which could be shared with other actors. A real-time embedded system is characterized by being driven by real world events. This issue could be simulated by MAS.

First of all, we specify the study case in general and we provide functional requirement specifications (FRS) in the subsequent step. We are dealing with embedded real-time control systems which have some specificities which must be respected.

Conceptual modeling is ”the activity of formally describing some aspects of the physical and social world around us for the purposes of understanding and communication” [Wiki]. The aim of this phase of the simulation project is to transform requirements into system models that are platform-independent and which contain different diagrams of different views. The choice or these conceptual diagrams should be meticulous in order to depict various aspects of the system and cover different views.

First we propose to define an object model to the very high level of granularity which is the abstraction of the real system. In the second step, we create an agent diagram to determine system actors in the defined scope. We propose to be general in this phase and give a global model without detailed design. The next step focuses on the creation of a global model for each agent and the determination of dependencies between them. In the fourth step, a detailed agent model is provided.

IV. Case Study: ERTMS/ ETCS System

The FRS of European Rail traffic Management System/ European Train Control System (ERTMS/ ETCS) [9] is the chosen embedded control system to be studied. The aim of this project is to unify the transeuropean railway network with the same and unique train control system. ERTMS is decomposed by on-board equipment which is the embedded system and Trackside equipment which is the static system (Wayside). The architecture of the whole system is illustrated by figure 5.

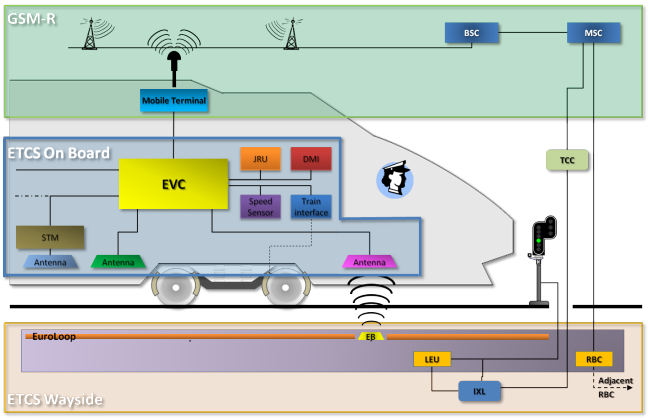


Fig. 5. ERTMS Architecture

We will focus on the Start of Mission (SoM) procedure since it is the first procedure to apply to start the train equipped by ERTMS. We note that in the embedded system is composed of ETCS on-board equipment beside other systems. The procedure ”Start of mission” is totally described in [9]. This procedure is used if the train is awake, if shunting movements are finished, if a mission is ended or if a slave machine becomes a leading engine. In fact, this procedure is used when the on-board system is in Stand-By mode. The procedure describes the status of system data required and details the table of requirements for the procedure.After the table, a complex flowchart describes the interactions between the different components of the system and gives a graphical vision of the procedure. We can notice that on-board system interacts with Radio network and RBC and the flowchart show how they interact between them.

A. System Evaluation for Simulation

The first step of the SoM simulation is to formulate the task which is provided by the subset of the procedure. We analyse the simulation description and we evaluate specifications in order to determine if the present case study is simulable or not by an agent-based model. The aim of this phase is to prove the efficiency of the making use of agent approach and its suitability and convenience in the study case. Therefore, we need to know some general specifications of agents such as making decision ability, the system dynamicity, granularity of the treated system, ect... In the case of SoM procedure , (i) there is one actor that makes decision decision- driver who can make many kinds of decisions. (ii) The system dynamicity is represented by faults avoidance in real-time execution. (iii) The system behavior is treated on a macro level which coincide with the principle of agent-based modeling, contrary to flowchart, activity diagrams or state transition diagrams which are typically viewed in macro-level. (iv) In micro level, each actor has its specificities and we note that there are different used agent architectures. (v) The environment of the MAS is characterised by the presence of spatial factors which may influence the simulation. All these previous metrics mean that an agent-based model is suitable for modeling an embedded real-time control system in general and our sturdy case in particular.

B. Conceptual Model

The concepts of hard and soft goals are used to model functional requirements and quality attributes respectively. Tropos, since it adopts i*, uses these concepts in addition to actors who could be an agent, role or position. Goals can be classified into types or categories. We talk about software and other behavioral goals.

A software goal announces an alternative that should be chosen by the system to reach another goal. A behavioral goal describes the expected behavior of a system. There are two subtypes of behavioral goals: ”Achieve/ Cease” and ”Maintain/ Avoid.” The first class includes goals describing a behavior that must be satisfied or not under conditions in a limited time in the future. The second class is the set of goals describing a behavior that should be satisfactory under conditions over time or behavior should never be present in the system under any condition respectively. We propose the following definition.

Definition 1. Lets S be a set of states, S_i an initial state, S_f a set of states in the future and t_0 is the current instant such that the following hold for $S_i \subseteq S$ and $S_f \subseteq S$:
$F(S_f) \rightarrow \exists t(t{>}t_0 \wedge P(S_f))$
$G(S_f) \rightarrow t(t{>}t_0 \rightarrow P(S_f))$
$Achieve[S_f]$: if $[S_i]$ then $F(S_f)$
$Cease[S_f]$: if $[S_i]$ then $\neg F(S_f)$
$Maintain[S_f]$: if $[S_i]$ then $G(S_f)$
$Avoid[S_f]$: if $[S_i]$ then $\neg G(S_f)$

C. Goal-based reflex agent Architecture

A goal-based agent is an agent having knowledge about its goals and what actions to choose to achieve it. A reflex agent is an agent who maintains an internal state and based in ”condition-action” rules it updates its state.

An hybrid architecture is a way to switch between the intuitive reaction and thought deliberation respecting the circumstances and environment events. Moreover, it is the latter that determines the expected behavior of the agent and these characteristics. An agent is considered as an autonomous entity or a decision-maker.

On an abstract view, an hybrid agent is composed of several layers each of which depends on a specific architecture of an agent. Each class of agents has its advantages and disadvantages. We can choose the most suitable functions in architectures using various agents although we face the problems of mutual interference.

The hybrid architectures are in hierarchical layers allowing the boundary delimitation between the different features of each level. The layers of the highest level are reserved for reflection, decision. This is the area of reasoning. The lowest layers are those devoted to communication and perception of the environment. Between these two levels, areas of interaction, knowledge or planning may exist and it depends mainly on the architecture.

Each hybrid agent must have at least two layers of different architectures. The general behavior of the agent will depend concepts of architecture but also the alignment of these architectures. The architecture of our agents as illustrated by figure 6 is hybrid and each actor of our system has the following properties:

- A set of states
- A set of goals
- A knowledge base (KB)
- Capabilities
- Actions

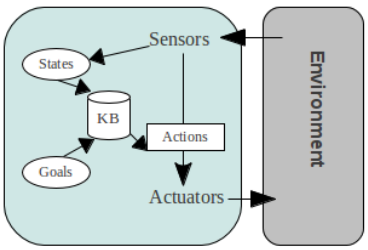


Fig. 6. General Goal-based Reflex Agent Architecture

V. Conclusion

In this work we present a modeling and simulation strategy for embedded control systems based on Tropos and Agentogy. Our approach differs from existing control system simulation approaches in the following way. Current approaches rely mostly on general-purpose simulation modeling paradigms based on continuous, discrete or hybrid simulation techniques. This trend is explained by the high degree of interaction between mechanical and electrical components in embedded control systems. Among others, our contribution is the use of agent-based simulations because these models are often appropriate and interesting to study the behavior of the entire

system where its characteristics are known. This paper is a first attempt to make ERTMS /ETCS modeling and simulation more specifically goals oriented. Future work is needed to validate these models in practice and to propose a detailed model and architectures for subsequent steps of simulations.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their helpful and constructive comments and suggestions on this research.

REFERENCES

[1] W. Michael and N. R. Jennings, *The Gaia Methodology for Agent-Oriented Analysis and Design*,In: Autonomous Agents and Multi-Agent System, 2000.

[2] J. Castro, M. Kolp and J. Mylopoulos *A requirement-driven development Methodology*,In: Lecture Notes in Computer Science, pages 108-123. Springer, 2001

[3] E. Yu *Towards Modeling and Resoning Support for Early-Phase Requirement Engineering*,In: Proceedings of the 3th IEEE International Symposium on Requirements Engineering, Washington, 1997.

[4] C. Bernon, M-P, Gleizes S. Peyruqueou and G. Picard *ADELFE: a Methodology for Adaptive Multi-Agent Systems Engineering*,Third International Workshop "Engineering Societies in the Agents World", Madrid, 2002.

[5] A.Van Lamsweerde, *Requirements Engineering, From System Goals to UML Models to Software Specifications*,Wiley, 2009.

[6] T. Salamon, *Design of Agent-based Models*, Academic series, 2011

[7] D. Bertolini, A. Perini, A. Susi and H. Mouratidis. *The Tropos visual modeling language. A MOF 1.4 compliant meta-model.* 2004

[8] W. Royce, *Managing the Development of Large Software Systems*, Proceedings of IEEE WESCON 26 (August): 19, 1970

[9] UNISIG, ERTMS Users Group , *Functional System Requirements Specification, FRS*, FRS V4.29, 1999

[10] A. Dardenne, A. V. Lamsweerde, S. Fickas *Goal-directed requirements acquisition*, In: Science Computer Program, vol. 20, page 3-50, April 1993

[11] J. Pavon, J. Gomez-sanz *Agent Oriented Software Engineering with INGENIAS*, In: Third International central and Eastern European Conference on Multi-Agent Systems (CEEMAS), Springer verlag, 2003

[12] P. Burrafato and M. Cossentino *Designing a Multi-Agent Solution for a Bookstore with the PASSI Methodology*, In: AOIS'02 at CAiSE'02, Toronto, May 2002

[13] A. Omicini, *SODA: Societies and infrastructures in the analysis and design of agent-based systems*, In: Agent Software Engineering, vol. 1975 of LNCS Springer, 2001

[14] A. Bauer, J.P. Muller, J. Odell, *Agent UML: A Formalism for Specifying*, In: Multiagent Software Systems Proceedings, ICSE 2000 Workshop on Agent-Oriented Software Engineering AOSE. Limerick, Springer Verlag, page 121-140, 2000