SystemC based Simulator for Virtual Prototyping of Large Scale Distributed Embedded Control Systems

Alberto Ferrari, Marco Carloni, Alessandro Mignogna
Francesco Menichelli
ALES, Rome, Italy
dang.nguyen@otis.com

Abstract—In this paper we present DESYRE, a SystemC-based virtual prototyping framework, that we have developed to build the simulator of a modern elevator system designed by Otis Elevator Company for large scale buildings. DESYRE aims at the simulation of industrial large-scale real-time distributed embedded systems and allows: the verification of the system functionality, taking into account the distribution over the network, the prediction of the performance, such as end-to-end latency, and the usage of the communication and computation resources for the entire range of scalability of the system. In the paper, we describe the framework details and its application to the construction of the virtual prototype of a scalable elevator system based on the CAN communication protocol. We show the tuning and validation of the simulated model against a test system composed of 24 physical nodes, linked to network and logic analyzers. We finally present results on the simulation of significant case studies (up to 20 floors and 8 cars, that corresponds to hundreds of interconnected nodes, having about 10 subcomponents each) to predict the scalability performances of the shared communication resources.

Keywords - Distributed Systems, SystemC, Platform Based Design, Real-Time Systems, Virtual Prototyping

I. INTRODUCTION

Large distributed embedded control systems are becoming common even in consolidated and mature applications such as elevator systems. Elevator systems in very large buildings can contain tens of elevator cars servicing more than one hundred floors and can consist of several thousand networked electronic control units (ECU), each containing microcontrollers, input/output devices, actuators, network interfaces, and communicating over several shared communication busses [1]. Elevator system architectures sometimes need to be updated in order to realize the benefits which can be achieved through the incorporation of new technologies. However, it is likely that scaling limitations exist in any system architecture, and as a result there is a risk associated with committing to any particular new architecture without fully understanding these boundaries. To avoid expensive and time consuming modifications, it is crucial that the elevator system designer is confident that the elevator system can achieve correct operation prior to installation. In this work we describe a set of virtual prototyping tools which provide that ability, and allow the system designer to quickly perform cost optimization through evaluation of architectural tradeoffs while ensuring that all requirements are met. In the scientific literature a large body of work exists which describes the benefits of systems development in a virtual environment [2][3][4][5]. Recently, numerous works describe simulation environments for embedded systems. In [6] a set of tools called Embedded System Environment (ESE) is presented, especially developed for multi-processor embedded systems, based on SystemC transaction level models. The importance of virtual platforms unifying functional and robustness analysis of embedded designs is shown in tools as the one described in [7], where a simulation environment for the LEON3, a 32bit SPARC CPU used by the European Space Agency is presented. The model is TLM based and unifies simulation functionalities with fault injection capabilities. Another important aspect in the simulation of embedded systems is the verification of real time constraints. In [8], for example, the issues regarding the combination of TLM hardware models and real-time software models are analyzed. Available virtual prototyping tools can be specialized toward specific applications [9] or generally suited to SoC designs [10][11]. Some of these are based on SystemC [12][13], because of its ability to integrate hardware and software together in a common language, making it a very attractive prototyping language to system designers. These tools traditionally focus on system-on-chip designs, and their primary target is not the simulation of large, distributed and interconnected embedded systems. On the other hand, while network simulators for large systems are widely available [14][15][16][17], they generally approach the modeling problem from a high level of abstraction [18], lacking the possibility to model delays due to limited software computation power, interrupt latencies, hardware resources sharing. In our case, virtual prototyping simplifies the effort required to build and test a wide range of potential system configurations under estimated worst case conditions very early in the design process, potentially saving the costs of construction and testing on a very large system, but requires a good level of accuracy at the node level, especially regarding software processing delays and hardware resource capabilities. To achieve these results we improve the DESYRE simulation framework to realize a virtual prototype of the system, which is able to simulate the network resources and the control/application functionalities, including timing information. The element of novelty of this work is the combination of its dimensions (can scale to the order of thousand of embedded nodes), the level of accuracy in the description of each node (simulation of the hardware
network components together with the protocol/application embedded software), the validation of the functional and timing simulation models against a physical test system.

The paper is organized as follows. In Section II we expose the details of the design problem, in Section III we describe the approach followed to simplify the modeling problem, based on the creation of a hierarchy of layers. In Section IV the DESYRE simulation framework, used to model all the components of the systems, is presented. Section V describes the back-annotation feature implemented in DESYRE for timing simulations. In Sections VI test cases, built both physically and virtually, are executed to validate the simulation model. Section VII reports the design space exploration analysis on system scaling. Finally Section VIII summarizes the obtained results.

II. PROBLEM FORMULATION

The elevator system in this paper we are referring to is highly scalable for being applied to a large variety of building configurations. To reduce cost and materials, dedicated point-to-point links are replaced with a shared communication bus. As a consequence, the system presents uncertainty in its communication performance due to scalability and adoption of shared resources.

A communication bus which is commonly used throughout the transportation industry including elevator systems is the Controller Area Network (CAN) bus [19]. This bus is used for inter-node communication on which only one node can transmit at any given time, causing variability in the end-to-end latency of messages. Much work is done focusing on the timing characterization of periodic CAN messages in [20], [21]. However, the Otis Elevator system which is being modeled contains a mixture of periodic and event triggered messages. The infrequent, soft-deadline nature of the latter makes possible to share network resources to an extent which is not previously realized. However, the event-based messaging scheme comes with the disadvantage of increased complexity and processing time with a potentially serious impact to the delivery of other time critical messages. In [24] mechanisms are proposed to mitigate this performance decrement but these are not achieved without additional costs in terms of processing and bandwidth. Due to the large range of conceivable configurations, the designer has the difficult job of ensuring that the same timing requirements are met whether there are ten or ten-thousand ECUs in the system. Traditionally, these systems must be constructed and tested before any performance metrics can be collected to determine whether or not the timing requirements are satisfied. Clearly, this can be very costly and time consuming due to the size and number of possible configurations.

III. DESIGN AND SIMULATION METHODOLOGY

We use a Platform-Based Design (PBD) [23] approach to develop a set of SystemC-based simulation models which can be composed into a virtual prototype of our complete elevator system. Each node of our elevator system model respects the architecture of a physical node and is composed of two primary layers, (a) the application behavior layer and (b) the network communication layer. These layers are present across all physical nodes in the system, and are illustrated in Figure 1. PBD adoption allows the verification of the functionality of the system independently of the network layer. This is achieved by the simulation of the application layer only with the assumption of an ideal network. The ideal network is then refined into the different layers and the physical topology without any needs to change the application interface. For each physical node the functionalities are partitioned between software and hardware. We chose to model the functionalities implemented by software layers (Application and Proprietary Network protocol) at behavioral level, while hardware functionalities (CAN Bus, CAN Controller and CAN Driver) are modeled accurately and include exact timing computation. Latencies due to software are introduced in the simulation recurring to annotation of computational delays obtained by measures from the test system described in Section V.

IV. DESYRE: A SIMULATION FRAMEWORK USING SYSTEMC

The models are developed in the DESYRE\footnote{DESYRE - Design Environment for distributed Real-time Embedded System (c) ALES S.r.l.} framework, a SystemC-based virtual prototyping environment supporting, among others, the PBD methodology and developed in several European projects [26], [27], and [28]. In this framework, the virtual prototype is composed of:

1) Functional components, written in SystemC, and/or automatically imported from other authoring tools, such as Simulink [24], and executing in zero time;

2) Architectural components refining at the transaction-level the communication and computation. They include a Real Time Operating System (RTOS) model and communication models (network protocols), exposing the effects of the resource sharing and constraints of the selected architectures;

3) Mapping components composing the previous two sets of models appropriately to create the virtual prototype of the system.

The models are organized in libraries belonging to a workspace, which defines the context where the simulation or the design space exploration, based on scenarios, take place.
The virtual prototype is specified as a hierarchical network with a set of parameterized configuration input files, compliant to the IP-XACT format [25]. These files define and instantiate the simulation components, representing both hardware and software elements, and connecting them together to specify the model of the entire system. To facilitate the generation of the system netlists for complex designs and for the design space exploration, DESYRE provides an exploration language (EL) to describe in a more concise form, as parameter sets and parameter relations, the different configurations to be simulated. The EL specification is used to automatically generate and parameterize the IP-XACT file sets, representing the selected scenarios. The designer has the freedom to run selectively the simulation of a scenario or of all scenarios as a batch exploration. A simulation is performed in three phases (see Figure 2.):

1) Netlist elaboration and component loading;
2) Simulation run;
3) Data post-processing and visualization.

The first two phases are repeated for each of the chosen scenarios. In the elaboration and loading phase, according to the designer’s netlists, the model builder of DESYRE dynamically creates in memory the system model by parsing the IP-XACT files and loading, through a component factory, the required SystemC models (compiled and present in the library). In the simulation run phase, the created system model is executed and the output traces are produced. In the data post-processing phase, traces are analyzed to aggregate and/or verify the performance and the functional data.

The dynamic technique for the model creation facilitates the design space exploration by 1) removing the need for the compilation of the different SystemC netlists (the flow is compiler free); 2) enabling the designer to quickly derive additional scenarios by parameterize the EL or IP-XACT files.

specification of system inputs. Working on the accuracy of the simulator, we enhance DESYRE by adding back-annotation properties to the models, with a granularity of function calls, giving to the user the feature of setting timing parameters. In order to provide representative timing information of the actual system, we observe some important performance metrics and use these to configure the simulation models. Each software component in our model represents embedded software running on a microcontroller which requires a non-zero amount of time to execute. This time delay is measured on a test system and is captured in the models by associating a fixed computational delay for the specified action. This process of back-annotation is repeated for each software component. After the simulation model is annotated with estimates for computational delay, we simulate the complete elevator system, examine, and validate system-wide metrics including signal latency.

A. Test System Configuration

A scalable test system architecture is shown in Figure 3. To validate the models, a test elevator system is constructed which consists of 24 physical nodes interconnected via 3 CAN-busses. Two nodes are able to communicate on multiple CAN-busses. An identical system is constructed in our virtual prototype which we validate against the behavior and performance of the test system. We instrument the test system with a data acquisition system (DAQ) consisting of CAN-bus analyzers, logic analyzers, signal generators and signal recorders in a synchronous fashion such that a complete picture of the state of the entire system can be reconstructed at any given instant from the collected measurements. The simulation models are easily capable of providing the complete system state at all times, and we compare these to our measurements. For the measurements the following setup are used:

- **CAN-bus Analyzer**, to record the time and contents of each message;
- **Logic/State Analyzer**, to profile the execution of code on the microcontroller;
- **Signal Generator and Signal Recorder**, to provide predefined input signals and track system outputs.

![Figure 2. DESYRE Simulation Framework](image)

During the development of this work, several optimizations to the framework are made to improve the level of automation in the process of modeling and executing the system.

V. BACK-ANNOTATION FEATURE FOR TIMING SIMULATIONS

The DESYRE framework described above is used to construct a virtual prototype of a complete elevator system according to a set of parameters including elevator system size and the

![Figure 3. Scalable Test System Architecture](image)
B. Back-Annotation

We configure the test system described above to automatically generate inputs to the system, and collect measurements of the computational delay for each of the functions which implement the behavior of the network stack. The input profile is selected such that it keeps the CAN-bus traffic low to reduce the number of CAN-interrupts and opportunities for bus contention, because both can have some impact on program flow and computational delay. After measurements are collected, the appropriate function delays are extracted and combined to produce the total software delay for each transaction of each layer of the network stack. With this data we characterize how much variability is present in each of these computational delays, and identify a single fixed value to apply to the components in the simulation model. This process is repeated for each transaction at each of the layers of the network stack. This is done so that an accurate profile of computational delays can be constructed throughout the entire network stack.

The data collected at the proprietary network layer is shown in Figure 4. The data indicate that the processing delay at this component level is not constant for every transaction. We find two major reasons for this variation in delays:

- **Task Preemption.** The processing of a message is sometimes preempted by an ISR or another high priority task. When this occurs, some additional delay is added to the measurement of the processing delay. The amount of additional delay can vary widely depending on the duration of the higher priority event. For the measurements shown in Figure 4, this is observed to occur in about 1 in 10 message transactions, and accounts for the slowest 10% of messages.

- **Message Size.** Not all messages are the same size, and as a result the processing delay for each message can vary. All messages of the same size are observed to have a nearly constant processing delay. When a cumulative distribution function (CDF) is produced of all individual message processing times, we observe steps in the curve at the points corresponding to the different size messages.

We select a single value to configure the simulation model for the proprietary network transmit and receive processing delays. While selecting a value we reject the portion of messages found to be affected by preemption because the simulation model includes this behavior, and this can be counted twice if it is included here. The value selected to configure the simulation model is the slowest of the non-preempted messages, providing an approximation of the worst case to be used as an abstraction of the processing delay of the proprietary network component during simulation. The processing delay through the CAN driver is split into two pieces which are not usually executed adjacent to each other in time: an ISR that services the hardware after each message transaction; and a conventional task which manages messages in a software queue. This non-adjacent task behavior is included in the simulation models with the implementation of a real time scheduler. Measurements are taken of both and are shown in Figure 5 and Figure 6. The same general features can be identified in the CAN Driver which is seen in the proprietary network in terms of the impact of message size and preemption on the total delay of a transaction with a couple of exceptions. The receive transaction has very little variability due to message size, and nearly all the variability comes from preemption. The transmit transaction is just the opposite in that it has very little preemption so nearly all the variability here comes from the message size. We repeat this analysis on the CAN Driver interrupts to identify the appropriate computational delays to be used during simulation. Just as with the CAN Driver receive transaction, the duration of the CAN Driver interrupt receive does not significantly depend on the size of the message. These interrupts are very rarely preempted, so the variations are entirely due to the message size or number of messages to process. The procedure for selecting the fixed values for the processing delay of the CAN driver component in the simulation model is also similar to that which is used for the proprietary network. The values selected are the largest of those that represent a single message transaction. This again provides an approximation of the worst case to be used as an abstraction of the processing delay of the CAN driver component during simulation.

---

2 We remind that back-annotation is required only for software latencies, while network latencies due to transmissions, contention, retransmission, etc. are accurately modeled in the simulator and do not require back annotation.
VI. SYSTEM VALIDATION

After the model is annotated with estimates for computational delay, we can proceed to simulate the complete elevator system to examine and validate system-wide performance metrics in order to assess the accuracy of our simulation model and provide context for any system-wide simulation results which are generated. In this section, we examine the performance of the test system and compare this to the performance seen in the simulated system. This comparison is performed on the same 24-node 3-bus test system we use for back-annotation of the model. We define input profiles for system inputs to provide identical inputs to both the test system and the virtual system. These input profiles are drastically different from the ones used during the measurements for time back-annotation. There are both event-based and periodic messages present in the elevator system. Some nodes generate an event-based message in response to a change in an input signal, and some nodes generate event messages in response to other event messages. Our pre-defined input profiles allow us to provide a variety of repeatable inputs to the system which cause the event-based messages to be generated according to the specified profile. The variety of input profiles allows us to evaluate performance under various loading conditions. For each input profile provided to the signal generators, we collect measurements from the CAN-bus analyzers, logic analyzers, and signal recorders to reconstruct message latency, bus utilization, and queue utilization. Message latency is ultimately dependent on several other system states, including task execution state, bus utilization and queue utilization. We examine the latency of round trip messages which are configured such that they have to traverse two CAN busses. Once received on the opposite end, these messages are processed by a task which then forms a new message as a response that must also traverse the same two busses and return to the node that generated the original message. For round-trip messages, the message latency is the total time from when the first message is transmitted until the last message is received. The latency of a particular message varies each time a message is transmitted due to many factors including network delay and processing delay, so it is useful to examine the statistical properties of a set of messages as opposed to any individual message. Ten measurements are collected with a given input profile, and significant variability seen between each set of measurements. This variability is caused by the variation in task activation times due to the asynchronous behavior of tasks on multiple ECUs in the system. In order to take this asynchronous task behavior into account in the simulation, a range of simulations is run which varies relevant task activation offsets to cover the range of possible operation in the test system.

<table>
<thead>
<tr>
<th>Method</th>
<th>Message Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation</td>
<td>Minimum (ms)</td>
</tr>
<tr>
<td>Measurement</td>
<td>40.6</td>
</tr>
<tr>
<td>Measurement</td>
<td>54.6</td>
</tr>
</tbody>
</table>

The simulation are found to cover a wider range than is observed during measurement, as reported in Table I. Therefore, the simulation model is capable of identifying more extreme message latency values than can easily be measured on a test system. The average is expected to be slightly higher in the simulation because the worst-case computational delay is back-annotated to the simulation model. This effect is then multiplied as each message encounters the worst case processing delay several times while in transit, which ultimately can be seen as a higher average latency. Each measurement and simulation run is combined into a CDF, and all are plotted together in Figure 7.

VII. DESIGN SPACE EXPLORATION

We improve the DESYRE virtual prototype to automatically explore potential elevator system configurations, and identify what, if any, are the limits of scalability, offering valuable information to the system designers. Simulations are performed on systems of various sizes and configurations, sometimes containing as many as 2400 nodes. In this section we describe an example in which we explore the scalability of the system along two dimensions, system size and input event frequency. We start with a small system configuration and gradually increase the size by adding additional elevator cars. We implement the process of automatically scaling the system, such as configuring, building, and running the system, with respect to some defined patterns. For each instantiation of a given size we examine the overall system behavior as we increase the average frequency of events across all system inputs. Events are applied to all system inputs independently, with timing determined by events determined by a uniform distribution. At some point during this set of tests, we expect to find that the system is overloaded, and no longer behaves according to specifications, indicating that a scaling limitation for the given system architecture is identified. As expected, this exploration identifies a scaling limit which is caused by the exhaustion of available communication resources on the CAN bus. The amount of information which must be carried over the network increases as the number of nodes on that network increases, and the CAN bus eventually becomes fully loaded. As the bus utilization reaches this limit, message queues begin to accumulate significant numbers of messages, and eventually run out of space. This represents a scaling limit due to the fact that messages are lost and correct system behavior cannot be maintained. The average event rate for the entire system and for each individual input is shown against system size in Figure 8. and Figure 9. In these plots the system behaves correctly.
everywhere below the indicated line with no events or messages lost. For example, Figure 8. shows that the system behaves correctly considering a 5-floor 4-car scenario if each node transmits less than about 33 events/sec; in case of a 20-floors 4-cars configuration, the maximum event rate per input drops to 4 events/sec. Figure 9. shows instead the maximum overall number of events transmitted on the system, obtained summing the events produced by each node. The plot has a maximum, because increasing the number of cars initially produces an increment in the number of nodes (transmitters) but also in bus resources (dedicated CAN lines). However, for systems with more than 4 cars, in these examples, the CAN lines shared between cars start saturating and become the bottleneck.

![Maximum Simultaneous Event Rate Per Input](image1)

**Figure 8.** Exploration of Event Frequency and System Size

![Maximum System-wide Event Rate](image2)

**Figure 9.** Exploration of Event Frequency and System Size

### VIII. Conclusion

We have developed a virtual prototype of a scalable elevator system along with the required analysis tools in DESYRE. The behavior and timings given by the simulations are validated against a test system to ensure that the models include sufficient fidelity and the selected levels of abstraction are appropriate. The simulation models are capable of discovering best and worst-case behaviors which are not easily produced in the test system. A design space exploration analysis has reported significant results regarding the correct behavior when system scales up. Ultimately, the DESYRE simulator helps system designers to perform automatic design space exploration by identifying the characteristics of scalability prior to implementation, and ensuring that the set of expected system configurations is compliant with timing requirements.

### References

1. OTIS Elevator Company, website http://www.otisworldwide.com
14. NS-2 Network Simulator, available online: http://www.isi.edu/nsnam/ns/
16. OPNET Network Simulator, available online: http://www.opnet.com