Simulation Tools for Dynamically Reconfigurable Automotive Embedded Systems - An Evaluation of TrueTime

Tahir Naseer Qureshi, DeJiu Chen, Magnus Persson, Martin Törngren
{tnqu, chen, magnper, martin} @md.kth.se
Department of Machine Design
Royal Institute of Technology – KTH, Sweden

Abstract

This paper investigates the requirements for simulation as a basis for architecture verification and validation in the development of self-configurable automotive embedded systems. One central characteristic of such systems is that the system configuration, in terms of functionalities, software and hardware components, as well as their properties and how they are related, varies dynamically according to environmental and internal conditions. The reasons for this could for example be due to the needs of optimized resource utilization, effective software maintenance, integration of external devices, and error handling. Both generic and specific requirements formulated for dynamic reconfiguration are presented. In this paper, we focus in particular on TrueTime, a Matlab/Simulink utility for simulation of real-time control systems. TrueTime is promising for supporting dynamically configurable systems but is found to lack memory and middleware (MW) abstractions, and does not fully support dynamic tasks. Since TrueTime provides abstractions similar to the programming of a real system, care has to be taken to ensure the separation between applications, middleware and the lower level platform. We finally discuss how the simulation environment can be linked to the architectural design and the run-time system.

1. Introduction

One important engineering activity in the development of automotive embedded systems is validation and verification (V&V), typically performed incrementally through model analysis and simulation, rapid-prototyping, software-in-the-loop simulation, and hardware-in-the-loop simulation. The aim is to provide early evaluation of solutions in respect to a given set of requirements and hence to enable efficient problem detection and resolution.

This work focuses on providing a practical basis for architecture V&V through simulation. The motivation and context is provided by research on a dynamically self-configurable middleware system for future automotive embedded systems. The middleware system is currently under development in the DySCAS (Dynamically Self-Configuring Automotive Systems) project, funded by European Commission and started in June 2006 [2]. The key system properties of concern include self-adaptability with performance and robustness guarantee [1, 2]. The system is self-adaptable in the sense that the system configuration, in terms of application and middleware functionalities, software and hardware configurations, varies dynamically according to current environmental and internal conditions. The adopted control schemes can be either Quality of Service (QoS) algorithms or policy-based rules. Such system properties, together with the involved architectural artefacts that are at different levels of abstraction and of heterogeneous types (e.g., different infrastructures and systems), pose high demands on the simulation environment.

The current state of the art practice is to embed several electronic control units (ECUs) into a vehicle. These ECUs performs specific functions such as engine control, vehicle suspension control etc. In case of failure of an ECU its functionality cannot be transferred to another one. Once the vehicle leaves the factory, upgrades and changes are very difficult. Moreover, a paradigm shift towards software and network centric ECUs is occurring. AUTOSAR (Automotive Open System Architecture) is the best example in this regards. In addition, consumers have very high expectations for multimedia systems such as navigation devices, laptops and PDAs etc. The DySCAS project targets a system that fulfils these expectations and making the automotive system dynamically reconfigurable and more flexible with focus on infotainment and telematics.

The future vehicular system takes the form of an embedded network system. This will comprise a wired core-network of processing nodes and sensor nodes spread throughout the vehicle. There will also be opportunities to incorporate mobile devices such as cell-phones, PDAs and other devices carried by the vehicle occupants. Such devices may have external connections to cellular networks and may also form ad-hoc networks between themselves. The automotive system will attempt to take advantage of the processing resources and connectivity of these devices when they are present.

Currently, a large number of tools from both industry and academia are available for modelling and simulation of embedded systems while also providing specialised V&V support for control and implementation design [3]. On the other hand, most of these tools have their emphasis on statically rather than dynamically
configured systems. Still, we considered TrueTime [4, 5] a promising basis based on a previously conducted survey on tools [3]. TrueTime is a Matlab/Simulink® based simulator originally developed for the control and implementation platform co-simulation of networked embedded systems. As a first step towards a complete simulation environment for the DySCAS middleware architecture, the support of TrueTime with respect to some dynamic configuration features in the context of DySCAS system and system development is evaluated.

The rest of the paper is organized as follows. First an introduction to the ongoing development of the DySCAS architecture is provided in section 2. In Section 3 we describe the context and overall requirements for modelling and simulation, as well as detailed scenarios that were simulated to evaluate the extent of usability of TrueTime. Along with this we identify low level tasks / services derived from the basic requirements. An introduction to TrueTime is presented in section 4 and the results of the evaluation are presented in section 5. Finally, we discuss future work in section 6 and sum up the report with the conclusions in section 7.

2. Workflow and architecture of DySCAS

The development of the DySCAS middleware system is based on user requirements and design constraints identified by automotive companies. At the top level, the user requirements are specified as four generic use cases (GUCs) listed below, which are further refined into more specific use cases (SUCs) for the specification of system requirements. For example, the balancing of ECU load is treated as a special use case in the category of GUC3.

1. **GUC1: New Device Attached to the Car** – automatic discovery and inclusion of external devices temporarily attached to the car information system. Such devices could for example be cell phones, laptops, and devices through wireless hotspot.
2. **GUC2: Integrating New Software Functionality** – automatic installation, removal, or repair of both application and system software through interfaces like in-vehicle information and diagnostics system, wireless hotspot.
3. **GUC3: Closed Reconfiguration** – reconfiguration of the system caused by an unexpected event, e.g. overload scenarios, component failure, etc.
4. **GUC4: Resource Optimization** – similar to GUC3 but triggered internally with the aim of optimizing the utilization of various hardware resources and devices.

The development of DySCAS middleware systems adopts an incremental process from conceptual design, to functional design, and to reference implementation, where each refinement level is characterized by an iterative loop of requirement specification, design and modelling, and V&V. The V&V activities, supported by analysis techniques and simulation tools, thus span a range of abstraction levels and allow on-track evolution of solutions against the requirements. For the architecture design, the V&V support of concern include the assessments of QoS and other control functionalities, behaviours and synchronisation of the middleware services, as well as the timeliness and performance of various middleware services as perceived by the end users.

2.1. Introduction to the DySCAS architecture

A typical configuration of DySCAS middleware (MW) system consists of the following major services, also illustrated in Figure 1:

- **Autonomic Configuration Management Service** – relating to the MW support for deducing system-wide dependencies of components and for planning changes.
- **Dependability & Quality Management Service** – relating to the MW support for online dependability control and QoS based optimization using information of current configuration and figures of resource utilization.
- **Autonomic Configuration Handler** – relating to the MW support for coordinating configuration operations.
- **Resource Deployment and Component Management Service** – relating to the MW support for resource and execution control as well as software loading on a networked system platform.
- **Repository Service** – relating to the MW support for storing, maintaining, and retrieving files, configuration rules, component images, and for logging runtime information for diagnostics purposes.

![Figure 1: An overview of the DySCAS architecture being developed in the DySCAS project [3].](image-url)
Network) and LIN (Local Interconnect Network), and other legacy software platforms.

The architecture development is supported through UML modelling, capturing the system requirements, structures, behaviours, and providing support for traceability and documentation of design solutions. The behaviour description is based on a combination of state-machine and activity diagrams.

As parallel tasks, possible system designs and behaviour specifications are also being evaluated both by simulation and by proof-of-concept implementations. The main focus of the simulation work so far has been to identify a simulation strategy and tool environment. Also, limitations of the simulation environment, and how to cope with them, have been investigated. This work and its results and challenges are the main focus of this paper.

3. Simulation context and requirements

As introduced in the previous section, the work in the DySCAS project includes three types of environments which have several possible common connections, illustrated in Figure 2.

![Figure 2: Desirable connections between the different development environments of DySCAS: 1 - Code generation/refinement, 2 - Software- and Hardware-in-the-loop (SIL and HIL) simulation, 3 - Feedback to design work, 4 Implementation evaluation, 5 - Use of the same (or similar) configuration tool and strategies.](image)

The architecture design and behaviour definitions are captured by UML models\(^2\). The simulation is carried out in an environment consisting of MatLab/Simulink/TrueTime, and the reference development platform consists of real hardware and development tools like compilers and linkers.

The overall requirements on the simulation environment are as follows (divided into mandatory and optional):

- Simulation of structural changes, i.e. adding/removing nodes and tasks (mandatory).
- Simulation of logical behaviour, e.g. modes and policies part of the DySCAS system can be simulated (mandatory).
- Performance simulation in terms of timing behaviour and processing/communication resource utilization (mandatory).
- Simulation of power consumption and memory usage (optional).
- Models can be defined at different abstraction levels for evaluation purposes (mandatory).
- The structure of the simulated DySCAS MW shall be as similar as possible to the proposed one, i.e. similar to a real implementation. (mandatory)
- The simulated MW shall be clearly separated from the application and from the OS and high level communication and hardware. (mandatory)
- It is highly desirable that the system configuration is carried out in the same way for the simulation and the reference implementation. (optional)
- It shall be possible to run (software in loop) SIL, where the MW API is the interface to the external code. (optional)
- Information flows shall be easy to follow for debugging and visualization of results. (mandatory)
- The correct logic sequence of events within the MW shall be prioritized over real-time issues. (mandatory)

3.1. Detailed requirements

Simulation and modelling can be done at different abstraction levels, ranging from low-level like hardware emulation to higher-level behavioural simulations. Depending on what aspect of the simulated system that is most important to evaluate, the most suitable level of abstraction will be different, [7].

Given the overall requirements, we can identify the following detailed capabilities (services/mechanisms) that a simulation environment needs to provide.

- **Key abstractions:** It must be possible to simulate operating system kernels with at least fundamental functionalities such as scheduling and admission control. Moreover, the basic behaviour of digital hardware devices such as laptops, PDAs etc. and networks should be supported. Support for both dynamic and static middleware services such as client-server interaction, virtual functional bus, configurations, device drivers and interfaces is also required. Memory, power and other properties such as temperature which can affect the efficiency and speed of hardware may be provided as an option. All these abstractions should be viewable, configurable and modifiable at different levels.

- **Sensing:** Every dynamic system whether physical, virtual, digital or analog require some kind of sensing,
therefore, for the case of dynamically reconfigurable systems, sensing of parameters such as CPU utilization, end-to-end utilization, execution and communication times is also required for both simulation and implementation. Sensing of memory and power utilization can be provided as an optional component.

- **Fine grained actuation:** Similar to sensing, different actuation mechanisms such as execution of different task for different modes, changing of network parameters like transmit power, decision between polling mechanism or other kind of network schedules, are also required for simulation.

- **Structural reconfiguration:** Possibility to add, detach hardware, create/kill tasks, download new software, update or downgrade an existing software, runtime code generation and software migration is also a requirement for simulation environment.

- **Fault simulation:** Different kind of faults can occur in a system, such as hardware failure, network congestion, faulty software upgrade etc. For a simulation environment targeting dependable systems, such kinds of faults are also required to be simulated.

- **Design-flow and reuse:** Different tools for different purposes are used in the design; therefore, it must be possible to use models from different tools in cooperation with each other, for example inclusions of models built with UML tools in Simulink. Ideally, the simulation environment can share software components and configuration tool and concepts with the other environments as outlined in Figure 2.

### 3.2. Scenarios for tool evaluation

The simulation tool for DySCAS is required to concentrate on concepts to aid the architecture development and investigate ideas early in the project. After a review of both the generic and specific use cases along with the requirements described in section 3.1 for the DySCAS project, the following simulation scenarios were identified to be simulated for analysis of the most primitive requirements.

i. Two networks, a wireless and one CAN network (controller area network).

ii. A master node (ECU 1) connected to the CAN network.

iii. A common node / gateway (ECU 2) between the two networks.

iv. An ECU (ECU 4) monitoring basic activities on the network which takes over as master node in case the original fails.

v. A PDA which can be connected and disconnected to the wireless network.

vi. An ECU (ECU 3) connected to the CAN network assigned to perform specific tasks like cruise control and also playing streaming audio on the car speakers when a PDA is connected to the vehicle.

vii. Migration of software from one node to another node. The two nodes can be on different networks.

viii. All nodes attachable and detachable from the network.

ix. Quality of service and admission control mechanism at each node as well as on the network level.

All these scenarios were supposed to validate the possibility to use TrueTime to simulate a runtime reconfigurable system, similar to DySCAS. Based on these scenarios, the following activities were defined for the investigation of TrueTime.

i. Master sending heart beats on the network and keeping record of all nodes present.

ii. An audio player on the PDA.

iii. Common node receiving and forwarding messages to PDA and master node.

iv. A resource monitor on each ECU, including PDA for monitoring CPU utilization, memory consumption, execution times, load imbalance, voltage and frequency etc.

v. Adding, removing and migration of software from an ECU.

vi. Negotiation, contracting and authentication services on respective ECUs.

### 4. Investigation of TrueTime

TrueTime [4, 5] is a Matlab/Simulink® based simulator for real-time control systems. It targets embedded systems software, real time operating systems, distributed control and networks. It is also possible to monitor performance through simulation. A brief description of TrueTime is as follows.

TrueTime provides abstractions for a basic RTOS (Real time operating system) with messaging services, and simulation of networks and network interfaces. A very basic interface for simulation of controlled power consumption is also available. Concepts more abstract than that, like higher level communication protocols, middleware, automatic power management, etc,
are out of the scope of TrueTime and need to be explicitly modelled by the modeller or added to the toolbox library.

In TrueTime, kernel, network (wired), wireless network and battery Simulink blocks are introduced, the interfaces of which are shown in Figure 4. The kernel blocks are event-driven and execute code that models, e.g., I/O tasks, control algorithms, and network interfaces. The scheduling policy of the individual kernel blocks is arbitrary and decided by the user. For wired and wireless, networks messages are sent and received according to the chosen network model. The battery block is used to simulate battery powered devices.

Figure 4: The TrueTime block library.

A key characteristic of TrueTime is that the system model and programming is very close in nature to the programming of a real system. APIs are provided for system initialization, RTOS calls and network communication. Applications are written using Matlab M-files. TrueTime allows the execution time of tasks and the transmission times of messages to be modelled as constant, random, or data-dependent. Furthermore, TrueTime allows simulation of context switching and task synchronization using events or monitors. This means that TrueTime will be based on the Simulink simulation engine, but modifies the default simulation behaviour by imposing time durations of actions and by supporting pre-emption.

In addition to the block library outlined in Figure 4, TrueTime provides a collection of C++ functions with corresponding MATLAB MEX-interfaces. Some functions are used to configure the simulation by creating tasks, interrupt handlers, monitors, timers, etc. The remaining functions are real-time primitives that are called from the task code during execution. These include functions for AD-DA conversion, changing task attributes, entering and leaving monitors, sending and receiving network messages, and more. TrueTime is configured in a C++ or MATLAB m-file, called an initialization script.

5. Evaluation Results

The result of the simulation for the scenarios described in section 3.2 will now be summarized. Listing 1 displays the sequence of operations when a PDA is connected to the system. The listing is extracted from the MatLab command window and the monitors of the main and redundant controllers shown in Figure 6 and 7. The "high level" in Figures 6 and 7 represent that the corresponding ECU is connected to the network and vice versa. The Simulink model is also shown in Figure 5.

Listing 1: Sequence of operations after the connection of PDA

- ECU2 detected a new device and sending this info to the controller
- The controller has received the signal from ECU2 about the detection of a new device
- Controller asking for the Authentication code
- ECU2 received the command to ask for authentication code from PDA
- Forwarding the requirement of Authentication to PDA
- Authentication code requirement received from ECU2 forwarded by the Controller
- Sending authentication code, for our case it's DySCAS
- ECU2 has received the authentication code from the PDA
- Processing Requests
- ECU2 forwarding the authentication code to the controller
- Controller received the authentication code
- Controller sending the ok signal
- ECU2 received the clearance signal from controller
- ECU2 forwarding the clearance signal to PDA
- PDA sending request to send data to ECU3
- ECU2 received the request from PDA to send data to ECU3
- PDA connected
- Forwarding this request to the controller
- Controller received the request for sending data to ECU3
- Controller sending the acceptance for sending data
- ECU2 received the acceptance for sending data to ECU3 by PDA
- Forwarding this acceptance to PDA
- Request to send data accepted to ECU3 accepted, therefore stopping playback at PDA
- PDA Sending audio data to ECU3 via ECU2
- ECU2 received the data for audio
- ECU2 forwarding the audio data to ECU3
- ECU3 received the audio data
- Now playing the audio from ECU3
5.1 TrueTime Capabilities

From the modelling and simulation results it can be stated that TrueTime can handle a variety of dynamic mechanisms such as

i. **Adding and removing hardware:** This feature can be used for invoking task on different network nodes in case of failure of a node as well as addition and removal of external devices such as PDA, laptop etc.

ii. **Transfer of inline code from one network node to another:** The Matlab command “Inline” is useful for simulating downloading and migration of software, however, it is not possible to simulate, installation and running a completely new software / task during runtime, as TrueTime requires all tasks to be defined before simulation.

iii. **Software reconfiguration:** Software reconfiguration can be considered as complement of what is described above. The feature of stopping and starting of a task can be used to simulate different failure modes and software crash.

iv. **Sensing:** TrueTime supports runtime changing of certain attributes of a task running on an ECU such as priority, relative and absolute deadlines, worst case execution time (WCET) and period. Along with this feature, the state of each task can be viewed by the monitor provided with each ECU. All these features can probably be used as sensors for end to end utilization, end to end delays through time stamping configuration.

v. **Actuation:** Using the fixed priority (FP) scheduling out of three other scheduling types (i.e. earliest deadline first, rate monotonic and deadline monotonic scheduling) and the feature of changing priority online it may be possible to incorporate higher or special scheduling policies. For CAN, network priorities can be changed online.

vi. **Fault Simulation:** Hardware failures can be simulated with TrueTime. It is also possible to simulate faults using the external interrupts, which can be used as proof-of-concept for fault tolerant system.
vii. **Voltage Scaling:** The feature of voltage scaling in TrueTime kernel can be used for simulating efficient use of power, but to a limited extent.

viii. **Open Source Code:** The feature that distinguishes TrueTime from other tools is the open source code. Therefore, this feature can be exploited to test and evaluate the feasibility of new architectures such as DySCAS. This includes the methods to overcome limitations described in the following subsection.

### 5.2 Limitations

Despite the above mentioned qualities TrueTime has some further limitations as follows.

i. **Memory abstraction:** TrueTime does not provide any explicit memory abstraction. This means that it isn’t possible to use TrueTime to simulate memory usage, overflow allocation etc. without extending TrueTime.

ii. **Middleware abstraction:** There is no abstraction for middleware services of any kind such as relationship and constraint management, software repository described in section 2.

iii. **Power consumption:** The DySCAS use case of graceful degradation requires an efficient power management system for the devices in use which is so far not available in TrueTime except for the voltage scaling feature.

iv. **Shut down of running tasks:** It is not possible to completely shut down /kill a task running on TrueTime kernel at run time.

v. **Prior knowledge of execution time:** The TrueTime requires prior knowledge of execution times for each task running on an ECU.

### 6. Future work

In order to formulate a complete set of requirements as well as development of a suitable tool, flexible enough to handle most of the anticipated features for dynamic configuration of future systems, the following work will be carried out in the near future.

1. Extending TrueTime for incorporating simulation of major architectural parts of DySCAS middleware which includes development of software repository, QoS control as well as resource and execution service.
2. We will investigate the possibilities to incorporate basic functions of AUTOSAR such as basic software and communication services.
3. Investigation for modification of TrueTime kernel source code to add features such as memory management, user defined functions such as scheduling and admission control mechanisms. An example of such a function is scheduling using control systems theory.
4. Possibly extending the existing TrueTime kernel blocks for additional standard sensing and actuation parameters and monitors like CPU, memory and power utilization.
5. Investigating the incorporation of UML generated software components into the Simulator.
7. Evaluation for the case of changing network conditions such as communication overflows, congestion etc.
8. Addition for support for Hardware in loop (HIL) and task for execution time analysis.
9. Inspired from the results from complementing research work [8], support for dynamic configuration management using commercial tools such as PDM will also be evaluated. This work will also include the development of suitable meta-model.
10. Apart from this, the support for the Bluetooth communication standard would also be desirable.

The final objective of the work presented in this paper is to develop a simulation environment capable of generic simulations of dynamic networked and embedded computer systems, specifically automotive ones. The anticipated interfaces of the simulation environment are illustrated in Figure 8. The simulated environment can conceptually be seen as having two parts – one which simulates a generic distributed system, and one which simulates the actual middleware and the software applications. Interfaces between them are used for setup and monitoring of the simulated network. In the ideal case, the software used for the simulation should be identical with the one used in the actual implementation (relating to the requirements in Section 3).

To support this kind of separation, the fact that TrueTime models are so close to the implementation have to be considered. One solution would be to have the configuration tool to generate the mapping of applications to the software/hardware platform. This corresponds to the viewpoint taken in AUTOSAR.

Another interesting issue is concerned with the connection between the design and run-time configuration. We anticipate the simulation environment with a configuration tool will be useful in investigating alternative solutions.

![Figure 8: Conceptual view of future simulation environment including it’s interfaces.](image-url)
7. Conclusions

In this paper we have discussed requirements for simulation of dynamically configurable systems. We have also defined simulation scenarios and evaluated TrueTime in a case study. Several limitations were discovered, where most of them are due to the fact that TrueTime was designed for simulation of distributed control systems with static computing environment. TrueTime shares these limitations with many other simulation tools. Most of these limitations can however be dealt with.

Further work will deal with the extensions of TrueTime in order to support simulation of dynamic mechanisms such as sensing, actuation, faults, middleware services network reconfiguration, dynamic loading of software etc. Moreover, models from different tools such as UML diagrams, operating system APIs etc. also need to be incorporated in a single software environment for better results and efficiency.

References


