KTH Industrial Engineering
and Management

Experiences in Simulating a Dynamically Self-Configuring Middleware: A Case Study of DySCAS

TAHIR NASEER QURESHI, DEJIU CHEN, MARTIN TÖRNGREN,
LEI FENG AND MAGNUS PERSSON

Stockholm 2009

Mechatronics Lab
Department of Machine Design
School of Industrial Technology and Management
Kungliga Tekniska Högskolan- The Royal Institute of Technology

TRITA-MMK 2009:04, ISSN 1400-1179, ISRN/KTH/MMK/R-09/04-SE

Abstract

The increased usage of electronic components, wired and wireless networks, software and advanced telematics systems in modern automotive systems has raised the overall complexity in terms of both the system functionalities and their development and maintenance. One way to handle a few of the complexities is the use of middlewares and introduction of self-management mechanisms. This report presents the experiences and efforts for verification and proof of concepts by simulations of a dynamically self-configuring middleware for automotive systems. A brief overview of the architecture and control flow of the simulated middleware, requirements for a self-managing systems and their simulation platform are also presented.

The presented simulations comprise mainly of three scenarios which cover a few of the major functionalities of the middleware under consideration. An evaluation of SimEvents^{TM} and discussion about the possible extensions and challenges are also presented.

Keywords:Dynamic Reconfiguration, Middleware, Modeling, Simulation, Embedded Systems, Resource Optimization, Software Download, Component Model, Model Transformation, Unified Modeling Language (UML), Context Aware, Hybrid System, Discrete Events, Software Architecture, DySCAS, Real-time systems.

Contents

1 Introduction

For the past two decades the amount of electronics and software used in automobiles has increased significantly. According to a study [24] the innovations from many fields such as electronics and materials engineering are continuously being integrated in the automotive industry. It is also expected that electronics and software systems will replace many mechanical systems and thus will become the key technology in the development of automobiles. In the same study it is also forecasted that the electronics and software will share around 35% of the total vehicle manufacturing cost. A survey carried out by IBM [38] identified eight significant aspects of vehicle development by 2020. Out of these eight aspects, software and electronics are expected to have the highest levels of innovation. This will contribute towards *intelligent, greener, economical and customizable* vehicles providing information, entertainment, safety and convenience to the consumers.

From a simple steering lock to an advanced anti-spin system, a modern vehicle has a large number of functionalities. This is achieved by employing a large of number of electronic control units (ECUs). Depending on the criticality, these ECUs are connected to each other through one or several networks to communicate with each other. Examples of the networks used in modern automobiles include CAN (Controller Area Network) [20] and the MOST (Media Oriented Systems Transport) [13] standard. More and more functionalities are being introduced by exploiting this interconnection of the ECUs and the flexibility in using software. This is leading to the increasing complexity in development. Due to this reason a large amount of time is spent on the verification and validation (V&V) before a product enters the market.

Features such as flexibility, scalability, quality, reliability, management of the increasing complexity, support for commercial off-the-shelf (COTS) software and hardware, cost and resource optimization are now considered essential for automotive systems. Moreover, there is also a need for self-management and post-deployment configurations such as software updates. This is due to the increasing maintenance costs and usage of mobile devices, and the shorter life time of vehicles internal components such as navigation devices as compared to the overall system. Through the self-management and post-deployment reconfigurations features such as transfer of functionality within a network in case of failure and integration of modern devices like navigation system with latest map and traffic conditions can be achieved.

The development and hence the time spent on testing can be reduced to some extent by verification and validation (V&V) at early development stages. The methods for V&V include but not limited to simulations and mathematical evaluation such as formal

verifications. Depending on the type of system, the chosen level of abstraction and tool, simulation can be the least time consuming and effective method for V&V. However, the credibility of a simulation is dependent on the choice of simulation tools, specifications of simulation models and the granularity. Therefore, it is a standard practice to perform hardware in loop (HIL) simulations, software in loop (SIL) simulations or even a small scale implementation in addition to stand alone computer simulations before the start of the actual production of a system.

Some of the above mentioned issues and requirements such as flexibility and scalability to a great extent are addressed by the AUTOSAR (AUTomotive Open System ARchitecture) [4] effort. The outcome of the AUTOSAR effort is a component based software architecture framework for managing the software and hardware complexities, scalability of software components, support for the use of commercial off-the-shelf components etc. However, more work is required to introduce run-time / post-deployment reconfiguration and upgradation mechanisms for self-management, fault-tolerance, resource management and robustness.

The DySCAS (Dynamically Self-Configuring Automotive Systems) project [15] is one of the efforts towards context awareness and self-configuration in telematics domain in automotive systems. This report presents a part of the simulation work carried out to support the development of a framework for a dynamically self-configuring middleware for automotive systems within the DySCAS project. These simulations were carried out by using the results from the efforts [36] for a simulation platform for DySCAS middleware. This simulation platform is based on SimEvents™ [27] which is an extension of Simulink® for discrete event simulations. The three scenarios covered in the simulations are two of the four DySCAS generic use cases and system startup process. One of the simulated scenario uses the algorithm [16] developed in the DySCAS project to demonstrate the resource management capability of a DySCAS system. A comparison of two simulations tools i.e. TrueTime [34] and SimEvents™ is also presented in the context of DySCAS simulations in addition to the requirements for self-managing systems and their simulation platforms.

The report is organized as follows. The motivation and background of the presented work along with its aim and scope is presented in this section. The next section gives an overview of the DySCAS architecture which is followed by a discussion about the requirements of self-managing systems and the simulation platform and the generic challenges in section 3. The requirements derived specifically for simulations of a DySCAS system are also discussed in section 3. This is followed by a discussion on related work. The SimEvents™ and the DySCAS library are introduced in section 5. This is followed by

the description of the simulation scenarios and results in section 6. The evaluation of SimEvents^{TM} based on the simulation experience is presented in section 7. In section 8 the conclusions and the possibilities for future research are discussed.

1.1 The DySCAS project

DySCAS (Dynamically Self-Configuring Automotive Systems) [15] is a European Commission funded project targeting self-management and context awareness in automotive systems. It considers the needs for variability handling and advancement in *product life-cycle management, resource optimization* and *supervisory control for runtime system verification, validation and error-handling* [11]. The major outcome of the project is the specifications of a framework for dynamically self-configuring middleware capable of integrating new devices and software, legacy solutions and complementing existing standards such as AUTOSAR [4]. Guidelines for tools, algorithms and implementation are also provided by the DySCAS project.

One of the most interesting features of DySCAS is the multidisciplinary integration of technologies such as control systems [39], autonomic computing [23, 25], middleware [5] and policy-based computing [2]. The DySCAS project also provides a ground for research on different mechanisms for resource management, quality of service, autonomic configuration etc.

The application scenarios considered in the DySCAS project are exemplified by the following four generic use cases [9].

1. ***GUC1: A new device attached to the vehicle*** related to the discovery and incorporation of new devices in the vehicle's communication range.

2. ***GUC2: Integrating new software functionality*** related to maintaining the software functionalities of both the operating system and application software. This also includes the addition of completely new software which is not known at the design time.

3. ***GUC3: Closed reconfiguration*** related to the provision of the support for advanced error handling and fault tolerance. For example, shut down of non-safety critical services in case of power shortage, relocation of software in case of failure of hardware, etc.

4. ***GUC4: Resource optimization*** which is similar to GUC3 but more related to optimization of system resources by balancing of workloads on different ECUs,

selection of different scheduling and quality of service techniques for guaranteed quality of service and reliability.

The above use cases are further refined into specific use cases. For example load balancing is considered as a specific use case of GUC2. Functional and non-functional requirements [9] of the DySCAS middleware are also derived from these use-cases.

The verification and validation work carried out in DySCAS includes safety analysis [17], formal verification [12], prototype implementations [10] and simulations [17, 12, 2].

1.2 Aim, approach and scope

The objectives of the work presented in this report are as follows:

- Derive requirements for a simulation platform for dynamically self-configuring systems and identify the challenges in developing and working with such a platform.
- Evaluate SimEvents™ for its capability to simulate DySCAS type systems.
- Verify the correctness of the DySCAS simulation library [36].
- Evaluate the structure and logical behavior of the DySCAS architecture.
- Demonstrate a few of the DySCAS scenarios by simulations through implementation and test cases in a simulated environment.
- Compare SimEvents™ with TrueTime based on the experiences gained from the simulations.

The following approach is followed to achieve the objectives. Based on the DySCAS scenarios and requirements [9] and the state of the art survey [8], the general requirements and challenges for self-managing systems and their simulation platforms are identified. This is followed by the derivation of the specific requirements for the DySCAS verification and validation. The logics and the behavior of the DySCAS architecture [11], SimEvents™ and the mapping scheme [36] from UML™ to SimEvents™ are verified by simulating three different DySCAS scenarios. Finally the TrueTime toolbox and SimEvents™ are compared based on the experiences gained from this work and an earlier evaluation [37].

The following delimitations were applied to set the scope of the work. With respect to DySCAS this work is limited to the core service components and the overall system startup. Fine grained details in the DySCAS specifications [11] are considered out

of scope of this work. One of the major delimitations of this work is that it is not directly supposed to provide any sophisticated algorithms for configuration, quality of service (QoS) or load balancing etc. Instead, the algorithm simplified from another DySCAS work [16] is simulated for proving the concepts and the possibility for using more sophisticated algorithms. Furthermore, out of the four DySCAS use cases only two use cases i.e. GUC1 and GUC4 are simulated.

2 The DySCAS architecture

This section presents a conceptual overview of the DySCAS architecture. For detailed specifications the readers are referred to [11].

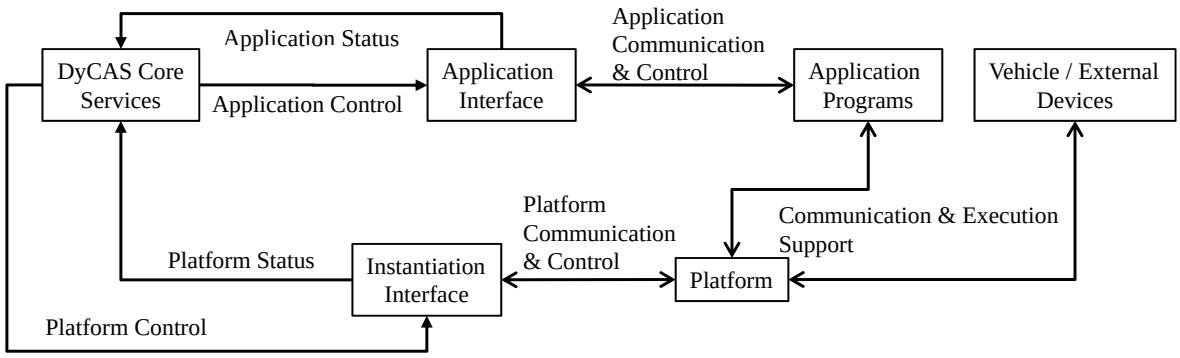

Figure 1: A conceptual view of the DySCAS architecture.

As shown in figure 1 DySCAS follows a well-defined strategy in terms of data and control flow. The platform communicates with the application programs, the external devices as well as the middleware services for controlling the execution/triggering of the tasks. Microprocessor, operating systems, communication networks such as CAN [20] are few examples of the elements of a platform. The *core services* are the major elements of a DySCAS system. The *application* and *instantiation* interfaces provide the means for the core services to communicate with the application programs and the platform. The following discussion is mainly focused on the core services.

Except for one, every core service is further divided into local and global service and is derived from the same basic component model as shown in figure 2. The signals, data and the input and output ports of the basic component model are classified into the following five types:

- *Service requests* related to the operations / decisions performed by a specific component.

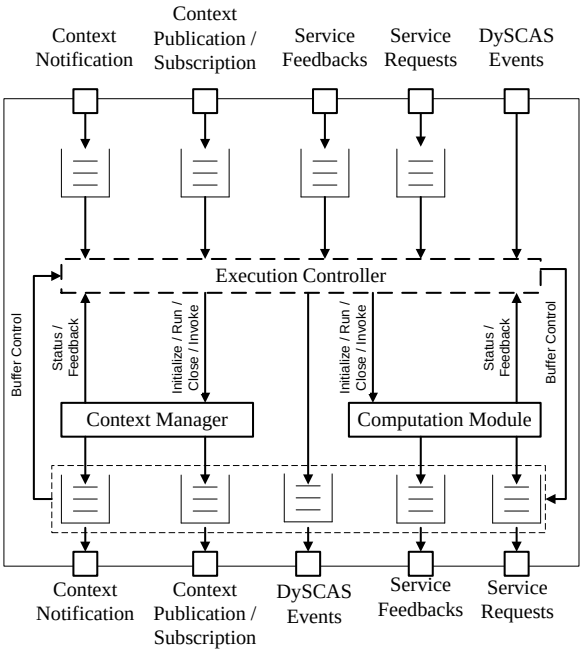

Figure 2: The basic component model for the DySCAS core services

- *Service feedbacks* related to the results of the service requested.
- *Context information publication/subscription* for measured or derived context information.
- *Context information notifications* for change of context information.
- *DySCAS events* such as error signals, signals related to component initialization and requests for change of mode.

Each component model has the following three different types of internal modules:

- Context management module responsible to receive, derive and disseminate context information.
- Computational module for performing computations and making decisions based on the information available from the context management module. The decision functions can be both static such as fixed algorithms or code and dynamic such as policy based mechanisms [2].
- Execution controller which controls the behavior of the overall component model. The behavior includes change of the operation modes based on a middleware event, reading and writing on the input and output queues, invoking the internal computational modules based on the type of received signal. Configuration, run, error,

wait and shutdown are the five modes supported by the execution controller. A simplified representation of the execution controller is shown in figure 3[1].

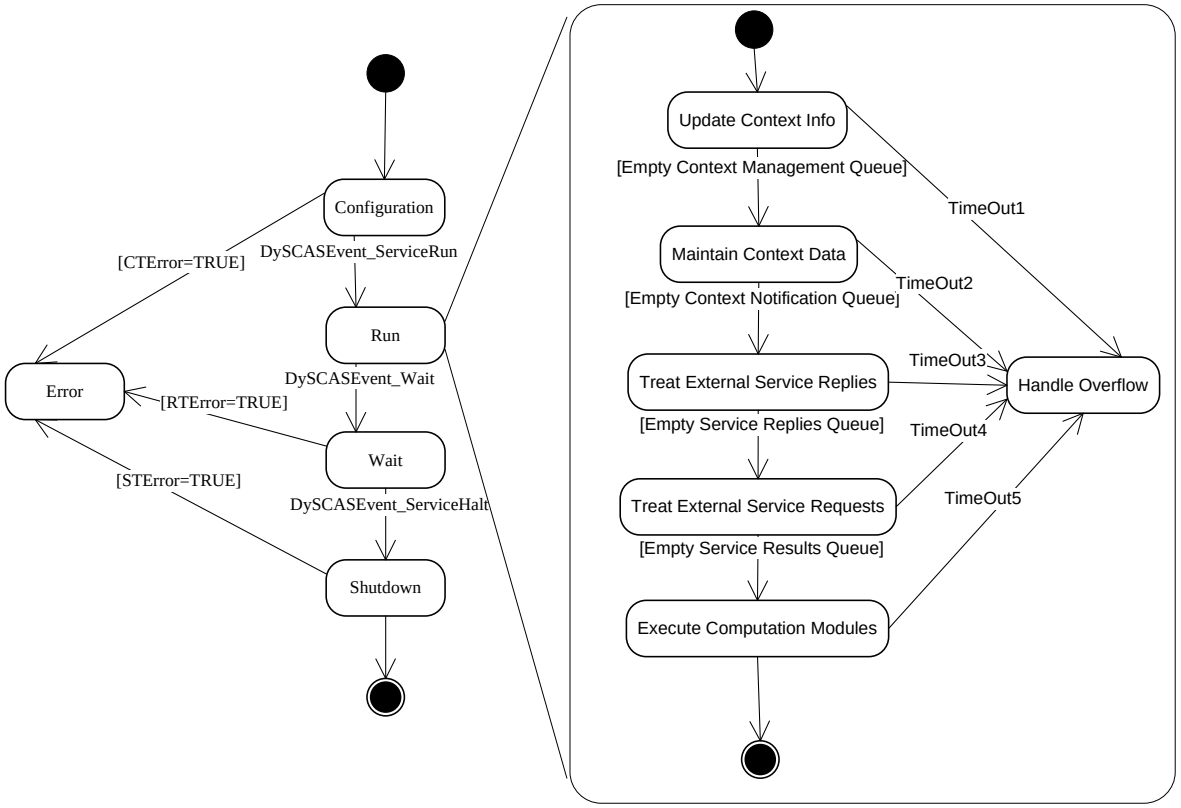

Figure 3: Execution controller for the DySCAS component model.

The core services and their functionalities are as follows:

- *Resource Deployment Management Service* (RDMS) supports resource and execution control as well as software loading on a networked system platform. Most of the lower level operations are carried out by this service. Apart from the context monitor, *external device control* and *application and resource control* are the two types of internal computational modules.

- *Dependability & Quality Management Service* (DQMS) is responsible for on-line dependability control and QoS based optimizations. *Quality control* is the module for performing computations within a DQMS.

- *Autonomic Configuration Management Service* (ACMS) supports the deduction of dependencies between different components for overall system. It also works as a

[1]The simplification also includes exclusion of some of the state transitions

planner for re-configurations. The computational modules related to ACMS are the *task scheduler* and *configuration resolver*

- *Autonomic Configuration Handler* (ACH) is the coordinator for the configuration operations scheduled by the ACMS.
- *Repository Service* supports storage, maintenance, and retrieval of files, configuration rules, component images, and logging of runtime information.
- *Software load management service* related to software loading and execution.

3 Requirements and challenges

One of the knowledge areas discussed in SWEBOK [1] is the software requirement area concerned with analysis, specification and validation of software requirements. This area is considered very vital for software development and also for systems engineering in general. For this reason it is important to understand the requirements for a system as a whole and derive specific requirements for the required tools. The generic requirements for self-managing systems like DySCAS, simulation platform and associated challenges are discussed in the following subsections.

3.1 Requirements for a self-managing system

Self-management in systems [29] implies the following characteristics:

- ***Self-configuration***: This includes the ability to derive knowledge about the internal and external states such as memory capacity and number of external devices connected to the system and automated configuration of components and system.
- ***Self-optimization***: The ability to monitor system resources, fine tuning of parameters and continually seeking opportunities for improving performance and efficiency of the system.
- ***Self-healing***: Automated detection, diagnosis and handling of hardware and software faults.
- ***Self-protection***: Ability to detect, identify and protect against malicious attacks and maintenance of overall security and integrity of the system.

Figure 4 illustrates a self-managing system from a control systems perspective. A self-managing system is able to keep track of external environment such as vehicle speed in case of an automotive middleware, the status of the platform on which it resides such as network utilization, its internal configuration like the number of available services and quality of service levels of the application programs. All these entities including the human machine interface can be referred to as the *plant*.

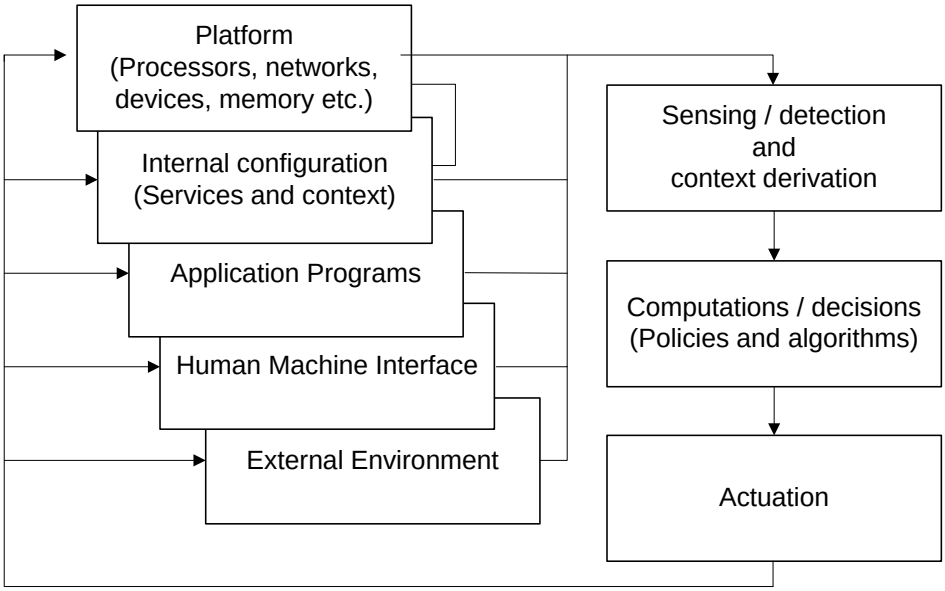

Figure 4: Control model for a self-managing system

The measured / sensed information is further processed for derivation of a refined context such as overall system quality of service. Based on the context information different computations are performed by using dedicated policies and algorithms to achieve efficiency e.g. fault handling and optimization. The outcome can be decisions and schemes for re-configuration resulting in actuation signals for different sequence of operations required for re-configuration. Change of quality of service levels for the applications, change in vehicle speed, a message at the navigation screen are a few examples of the actuation signals.

Based on the above discussion following requirements can be formulated for a self-managing system:

- ***Monitoring of external and internal context*** such as vehicle location in case of automotive systems, attached /detached devices, system resources which includes but not limited to processor utilization, network bandwidth and system states including information such as software versions, error codes and policies.

- ***Identification of unexpected events*** which are not part of normal operation, malfunctions of components, resource imbalance, new software and devices, detachment of devices.
- ***Verification and decision:*** Dynamic reconfiguration in a system implies some kind of intelligence and planning capabilities. Usage of different kind of policies, dynamic and feedback based algorithms enable the system to optimize and evaluate different configurations. Verification of a new software for security, dynamic and runtime decisions for different devices are a few examples of the requirements on verification and decision.
- ***Execution:*** A self-configuring system is required to execute many activities some of which are as follows:
  - Setting up of communication and data routing between the system and a remote device which may include a server at the vehicle manufacturer or workshop for updating and downloading new softwares.
  - Controlling the power, memory and processor utilization and other resources.
  - Download, installation, rollback, execution and migration of software between different nodes of a distributed system.
  - Error handling such as configuration rollback.

All of the above activities are required to be performed in a timely manner which can vary from one system to another. Moreover, these activities should not cause any kind of disturbances or distraction for the system users. These systems should also be robust towards disturbances, errors and failures.

3.2 Requirements for simulation systems

Based on the properties of a self-managing system, following requirements can be derived for tools for simulating such systems.

- **Component models:** A simulation platform should be able to support
  - Structural properties of a software component including its interfaces, signals and the data types.
  - Behavioral properties such as change of modes and states, activities like start-up mechanisms, interactions between different components in a system and sequence of operations for various scenarios.

- **External environment:** It should be possible to simulate external environment with which the system under consideration interacts. For an automotive middleware a sensor measuring the vehicle speed can be considered as a part of the external environment.
- **Core dynamics:** The dynamic behavior is one of the key features of a self-managing system. Therefore, it should be supported in the simulation platforms.
- **Key abstractions:** Following abstractions should be supported
  - Operating systems required to run different application programs.
  - Networks which are commonly used in distributed systems.
  - Self-configuration mechanisms and algorithms including policy based configurations if applicable for the system under consideration.
  - Message and signal handling mechanisms.
- **Sensing and actuation:** It should also be possible to simulate different sensing and actuation mechanisms. Measurement of processor utilization and changing the quality of service levels of applications running on an ECU are a few examples of sensing and actuation.
- **Fault generation and testing:** The simulation environment should support fault generation/injection mechanisms to test the fault handling capability of the system.
- **Dynamic loading and linking:** For systems which require on-line update and replacement of software, the simulation environment should support dynamic linking and loading mechanisms to simulate upgrades and updates of software components.
- **Real-time properties:** A self-managing system is required to perform its activities within a certain time limit and with some constraints such as length of queues for messages. Therefore, a simulation platform should enable its user to test and analyze different real-time characteristics of the system.
- **Levels of abstractions:** It should be possible to simulate a system at different levels of abstraction. Furthermore, it should also be possible to choose between different levels for different parts of a system under consideration. For example, simulation of logical behavior of a networked system may not require detailed behavior of the network.

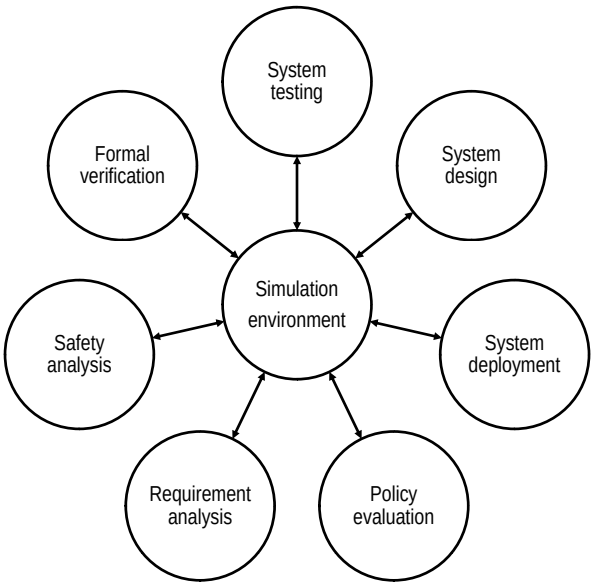

Figure 5: Ideal connection between simulation environment and other tools

- **Design flow and reuse:** It should be possible to use models from different tools in cooperation with each other, for example inclusion of models built with UML™ in Simulink®. Ideally, the simulation environment should be well connected to other tools for formal verification, safety analysis, analysis of policies (if policy based mechanisms [2] is used) etc. This requirement is illustrated in figure 5.

- **Comprehensibility and visualization:** The simulation environment should be user friendly and comprehensible for better understanding of the system. Moreover, it should also be possible to visualize different aspects such as sequences of activities and changes in system parameters.

3.3 General challenges

Complex embedded system and the development of their simulation tools put forth great challenges for the developers. This is also applicable for DySCAS type systems. A few of the challenges are as follows:

- ***Levels of abstraction and complexity:*** In order to reduce the analysis complexity, simulations are performed at different levels of abstraction. Depending on the purpose, some parts of the system need to be simulated at a very low level of abstraction while a higher level may be sufficient for other parts. Too abstract models can yield invalid results. On the other hand fine-grained simulation can take longer time for developing models and for analysis making it difficult to

achieve answers in a reasonable time along with adding the difficulty in traceability and understandability. The complexity in developing a simulation platform increases with the increased levels of abstraction. For example, it is more complex to simulate a complete operating system as compared to its basic mechanisms such as scheduling. The complexity can increase considerably if different abstraction levels need to be simulated at the same time. Therefore, the decision to choose levels of abstraction and details to be simulated for different parts is also one of the major challenges.

- ***View integration and model transformation:*** Because of multitude aspects of modeling DySCAS type architecture, several views have to be considered. Examples of different views include safety, requirements and deployment. For efficient development, these views need to be integrated. In addition, the development process is iterative and changes in the design are frequent especially in the early stages of development. This gives rise to the need for transformations of models between different domains and tools. For example a system model designed using UML™ [42] may need to transformed into a model suitable for some specific simulation environment.

- ***Choice and integration of tools:*** There are many commercial and non-commercial tools available. Some tools are open source while others are closed source. Each tool covers only a fraction of the tool requirements. This makes it challenging to choose between different tools and sometimes gives rise to the need for integrating various tools. For example, a CAD tool can be used with a simulation tool such as Simulink® for evaluation of mechanical systems.

3.4 Specific requirements for the DySCAS simulations

The major purpose of the simulations presented in this report was the proof of concepts of the DySCAS architecture [11]. The requirements for simulation systems derived in [37] are further refined for the objectives described in section 1.2 as follows:

- It should be possible to simulate different use cases described in section 1.1.

- The simulations should cover the following aspects of the DySCAS core service components:

  - Structure including ports, signals and internal modules.
  - Input and output queues and message handling.

  - Context derivation and dissemination.
  - The execution controller[2] responsible to control the working of internal modules.
  - Decision functions.

- Startup behavior of one implementation of a DySCAS system.
- It should be possible to visualize different activities which are difficult to demonstrate with actual implementations.
- Policy-based mechanisms [2] (optional).
- Basic operating system characteristics such as scheduling and pre-emption of tasks.
- Sensing of the context information and actuation to trigger re-configuration processes.
- It should be possible to evaluate timing aspects the dependencies between different services and tasks.

4 Related work

In addition to the presented work, the DySCAS middleware and its architecture have been simulated in various ways for different purposes. A few dynamic reconfiguration and quality of service mechanisms for the DySCAS middleware are presented in [16, 18]. This work includes simulations using TrueTime and demonstration of quality of service control as well as resource optimization and migration of software between different nodes in a network. A load balancing scheme and simulations for a dynamically configurable middleware is presented in [28].

In addition to above a lot of work has been done in terms of simulations of different middlewares especially the middlewares developed for wireless networks. One such effort is the WISDOM (Wsn mIddleware Service moDules simulatiOn platforM) [31] framework written in the Java language for simulation and verification for different middleware protocols in wireless sensor networks. [43] presented a simulation of the Agilla middleware [19] using TOSSIM (TinyOS mote simulator). UCS (Ultra CORBA Simulator) [41] is one of the simulators for simulating CORBA(Common Object Requesting Broker Architecture) [14] middleware. The provided support in UCS includes simulation for both

[2]Discussed in section 2

client and server sides, naming service, GIOP (General InterORB Protocol) and various other functionalities. The former two methods i.e. the client-server and naming/trading mechanisms are also used in the DySCAS middleware.

A DySCAS system can be considered as a hybrid system due to its discrete event nature and its interaction with the environment which can be both continuous time and discrete time. In [6] a framework for constructing simulation models of hybrid manufacturing systems is presented. In this framework the UML™ class diagrams drawn for a specific system are implemented using Matlab® functions and Stateflow® for simulating the state machines. This makes it very close to the presented work where the simulation library used in this work to a great extent corresponds to UML™ activity and state-machine diagrams. A survey on languages and tools for hybrid systems is presented in [35]. This survey was mainly carried out on tools for both simulation and formal verification. A comparative study pointing towards the need for tool integration is also presented in this survey along with the emphasis on the need for standard interchange format for filling the current gap between different tools.

One of the objectives of the presented work is the verification of the DySCAS architecture. This also implies some real-time properties such as timings. A lot of ideas and inspirations can be obtained from the already existing work. A few tools have resulted from different efforts. MAST (Modeling and Analysis Suite for Real-Time Applications) [22, 32] provide a set of tools for schedulability analysis, calculation of blocking and slack times and optimized priority assignment techniques for real-time systems. One of the interesting feature of MAST is that its model can be used in a UML™ environment for designing a real-time application. ARTISST (A Real-Time System Simulation Tool) is another tool for analyzing timing characteristics of computing systems. It is capable of simulating different RTOS schedulers and complex task execution patterns with user defined details. Torsche (Time Optimization of Resources, SCHEduling) [40] is a scheduling toolbox developed for Matlab®. Currently it supports scheduling on single and distributed processors, cyclic and real-time scheduling.

5 SimEvents™ and the DySCAS library

SimEvents™ is a toolbox for Simulink® for simulating discrete event systems. *Entities* and *events* are the two basic concept used in SimEvents™. While the former refers to the *discrete items of interest* such as packets and frames in a communication network, the later refers to a discrete incident such as change of state or occurrence of other events like a function call [26]. The following blocks are included in the library of version 2.4

of SimEvents^{TM}.

- Generators for generating entities, (function-calls) events and (numeric) signals.
- Attributes for managing data attached to an entity. A new reference signal for a controller is one of the examples of attributes.
- Queues for storing entities. The current version supports FIFO, LIFO and priority queues.
- Servers for arbitrating entities in a path based on a specified service time. The execution time of a task on a specific processor scheduled by an operating system is one of the example of the usage of servers in SimEvents^{TM} [3].
- Routing for modeling entity path. It includes input and output switches, path combiner and replicate blocks.
- Gates for regulating admission of entities. This is particularly useful for simulating scenarios such as blocking of queues in a server.
- Entity management for combining and splitting entities.
- Signal management for manipulating signals which currently includes specifying initial values and latching of signals.
- Timing for assigning time related properties to an entity such as time-out.
- Event translation for generating function calls based on arrival of events or change of a signal.
- SimEvents sinks providing support for plotting different data values.
- Ports and subsystems for discrete event simulations.

A review of SimEvents^{TM} is presented in [21]. In this review SimEvents^{TM} was evaluated for environment, entity, channel, queue, server, logic control, abstraction and user control aspects. Some of the evaluated features include customization, replication, priorities, testing and feedback. Apart from few, most of the evaluated aspects exist in SimEvents^{TM} either directly or indirectly. The review found SimEvents^{TM} a promising simulation tool for future given that alternate mechanisms can be implemented for mechanisms which are not directly provided. A few applications have already been implemented using SimEvents^{TM}. These applications include simulation of queues for

manufacturing systems [33], anti-lock brake system (ABS) with CAN (Controller Area Network) [7] and modeling of a nuclear facility [30]. SimEvents^{TM} has also been used to simulate real-time systems including the Ethernet network, operating system and memory management [3].

Due to the above mentioned support for discrete event systems, continuous and discrete time systems and state machines by Matlab® / Simulink® / SimEvents^{TM} / Stateflow®, a library of the DySCAS core service components was developed by transforming the DySCAS models specified in UML^{TM} to SimEvents^{TM} [36]. Figure 6 shows a simplified version of the transformed model of the *Local dependability and quality management service*.

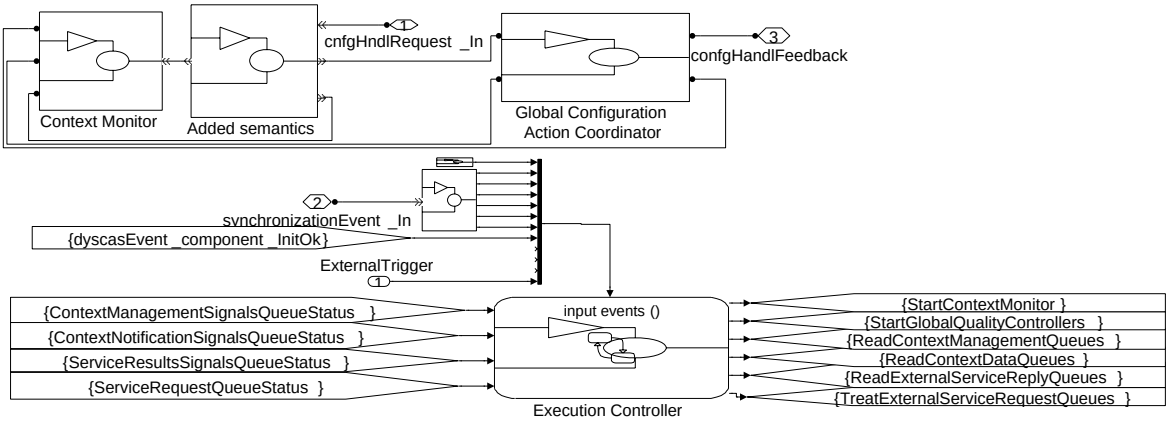

Figure 6: A simplified representation of a core service component in SimEvents^{TM}

Each core service in the library conforms to the DySCAS specifications [11] and comprises of an execution controller implemented in Stateflow®, input queues and their handling mechanism. Function calls are used as the input events to the execution controller. The *External Trigger* input represent the trigger from operating system. This implies that the triggering of each component has to be provided by user of the library. Other inputs include the status of four different kinds of queues for context management, context notifications, service requests and feedback signals. The read and write operations of the input queues are controlled by the execution controller. These signals as well as the signals required for initializing the internal modules are also implemented as function calls. The Stateflow® model of the execution controller is shown in figure 7.

By default the queue length for each input port is 5 which can be changed by the user. Furthermore, the internal functionalities of the computational and context management modules is also required to be provided by the user. An example implementation using

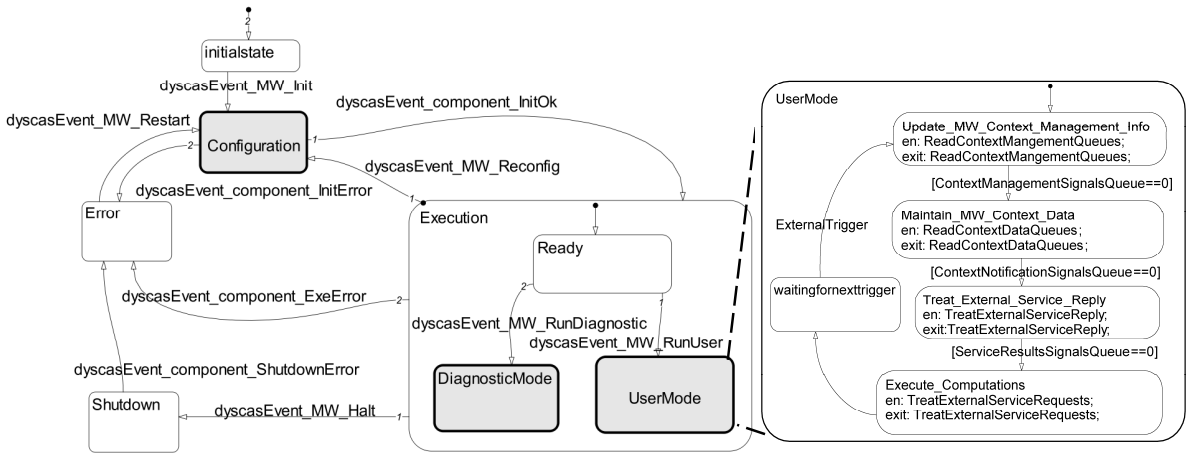

Figure 7: Stateflow® implementation of the execution controller

entities for the interface data and signals is provided in [36] which can be used as a reference for developing simulation models.

6 Simulation case studies

The following scenarios are considered for simulations:

- Attachment of a new device attached to the vehicle.
- System startup.
- Application quality of service (QoS) control.

The interaction between DySCAS core services and the platform were illustrated in figure 1. For the simulations, the instantiation interface comprises of *Master Configuration Management service*(MCMS), *Slave Configuration Management Service*(SCMS), *local node handler* and *External Device Handler*. These three components are specified in [11] and are required for system startup and monitoring of resources from the platform. The core services involved are the *Local Resource Deployment Management Service*(LRDMS) and the Local Dependability and Quality Management Service(LDQMS). The functionalities and composition of these components will be further clarified in the following sections.

6.1 New device attached to the vehicle

This scenario is one of the DySCAS specific use cases. The major activities in this use case include detection of a new device, derivation and dissemination of context infor-

mation, monitoring of available resources, authentication of new devices and planning and execution of the actual device connection. Figure 8[3] shows a schematic view of the service interactions.

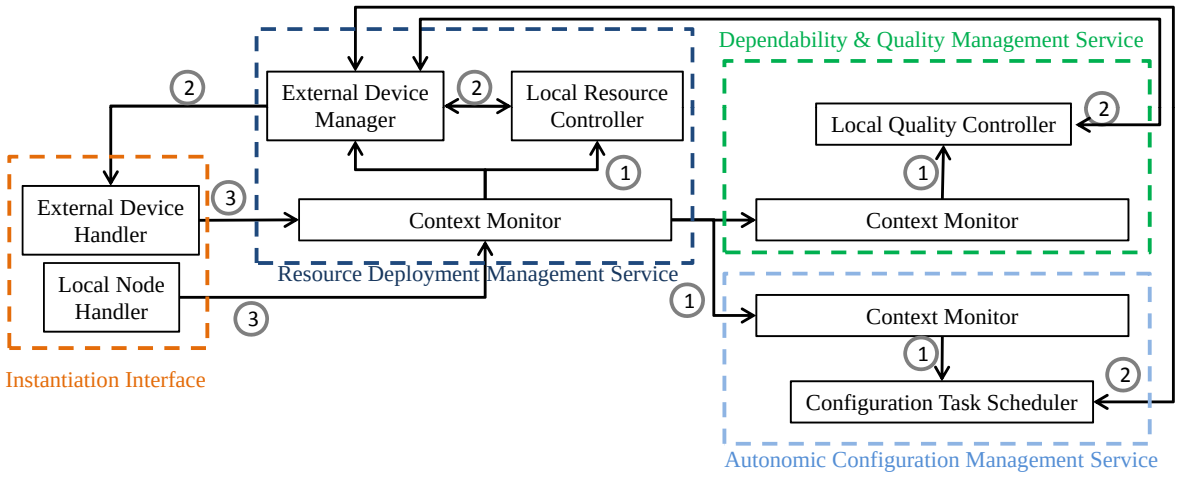

Figure 8: Service interactions for the attachment of a new device: (1) Context dissemination, (2) Service requests/feedbacks, and (3) Platform / Device status

A successful detection sequence is listing in listing 1.

Listing 1: Detection of a new device

```
1 Check for available resources.(Local Resource Controller)
2 Check if the device is authentic.(Local Quality Controller)
3 Connect the new device.(External device handler)
```

Listing 1: Detection of a new device

6.1.1 Context management and dissemination

The local node handler periodically sends the status of the platform including memory, processor utilization and the total number of available devices to the context monitor located in the resource deployment management service. This context is evaluated and disseminated to the context monitor (serving as proxies for the main context monitor) of the dependability and quality management service.

6.1.2 Service requests and feedbacks

In case of a change of a context, a service acts by making some own decisions and / or sending requests to other services. On detection of a new device the LDQMS checks for

[3]Note: The numbers in the figure represent the type of signals instead of any sequence

the available resources and sends a request to LDQMS for checking the authenticity of the new device. Based on a positive feedback from LDQMS and availability of enough resources, the LDQMS sends a request to the external device handler to connect the new device.

6.1.3 Platform / Device status

The number of connected devices and information about the services available through these devices are forwarded by the instantiation layer to the LRDM.

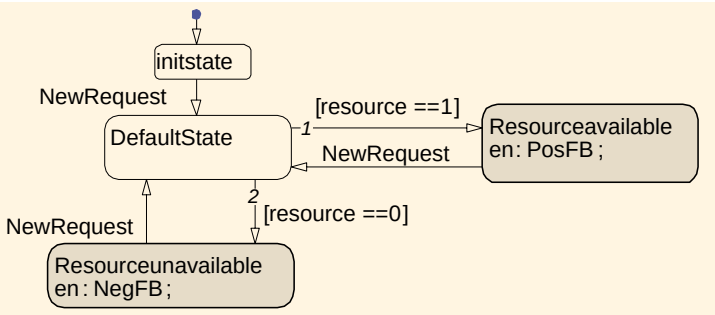

Figure 9: Decision function illustration

If any of the operations fails i.e. authentication and availability of enough resources to connect the device, the device is considered as rejected. For this scenario, the decision functions are implemented as Stateflow® charts where the required context value is either one or zero. This is illustrated in figure 9 for the case of LRDMS. For the simulations the context information is manually generated and altered at run time by using Matlab® commands. The variable *resource* corresponds to the combined availability of resources such as memory and processor. The same applies for device authentication decision function in LDQMS.

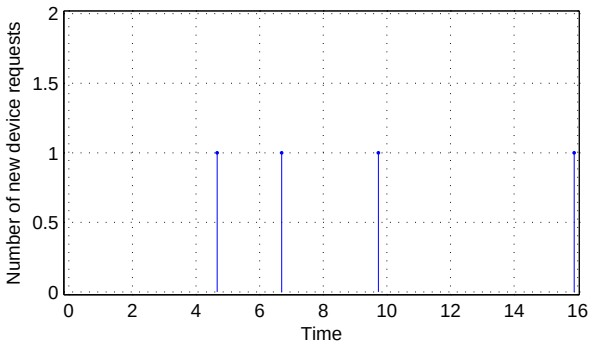

Figure 10: Requests for new devices

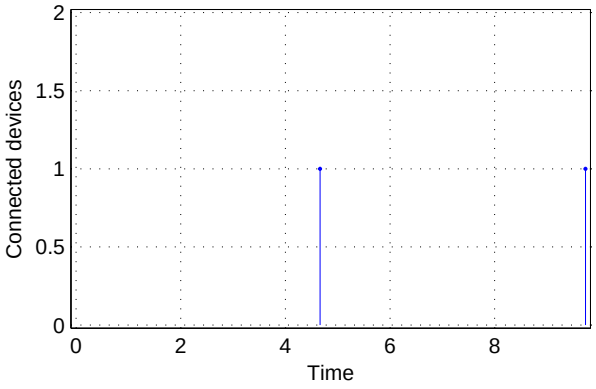

Figure 11: Connected new devices

Figures 10 and 11 show graphical outputs from one run of simulations where four devices are detected at different time values. Only the second and fourth devices are attached. The first rejection is due to the lack of resources and the second one due to the authentication failure.

6.2 System deployment and startup

The DySCAS startup activity is specified by a state-machine shown in figure 12.

For the simulations, a networked system with three nodes was assumed, where two of the nodes are only used for generating the signals required for startup.

6.2.1 Master/Slave configuration management service

System initialization is carried out by the *Master Configuration Management Service* (MCMS). The input signals for MCMS are signals from the *Network Communication Management Service* (NCMS) and *Slave Configuration Management Service* (SCMS) of each node. The output signals are commands for each SCMS for initializing and running the services of their respective node. The Stateflow® model shown in figure Annex.1 illustrates the startup activities in MCMS.

The SCMS of each node is also implemented by a Stateflow® block. The inputs to SCMS are the initialization status (success/failure) of the services along with the commands from MCMS. The outputs from SCMS are the commands to the services on the associated node for initialization and triggering of running state (User mode in figure 7. Feedback signal to the MCMS regarding the status (Failed/Ready) of the initialization activity is also one of the outputs of SCMS. On receiving an initialization command from the MCMS, SCMS sends initialization commands (implemented as function calls) to each

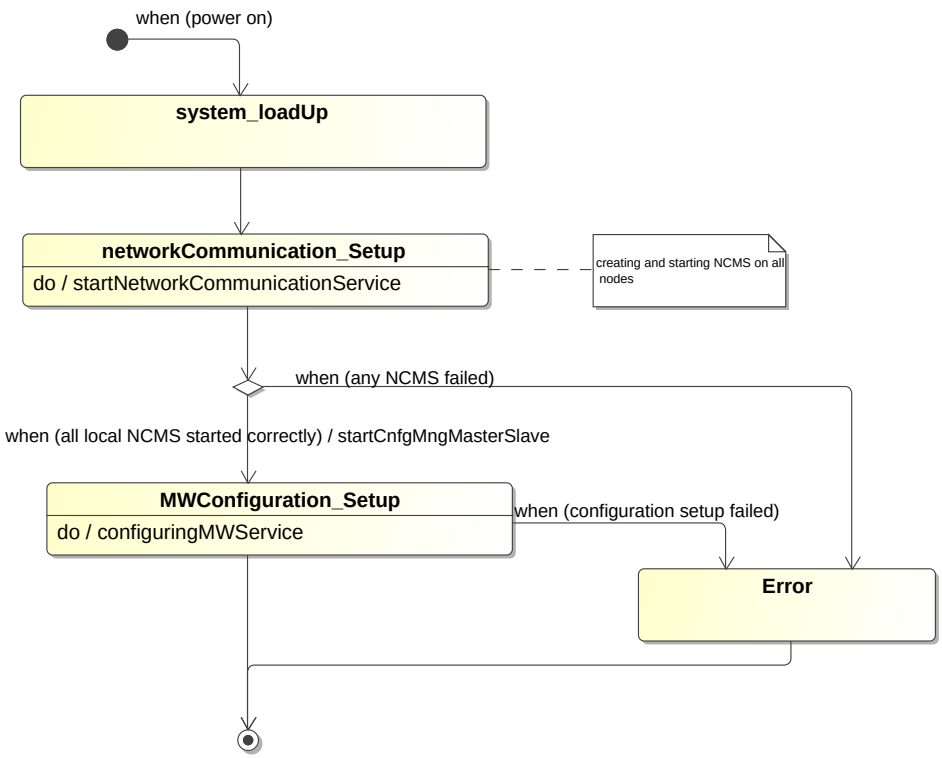

Figure 12: DySCAS system startup in UML^{TM}[11]

service in a specified sequence. If any of the services fails to initialize, a node failure signal is send to the MCMS.

6.2.2 Network communication management service

The purpose of the network communication management service is to coordinate the communication between both local and external services. As this part of the simulations is focused only on the startup behavior, the startup of network services is simulated by generating an event based on a step signal where the step time can be varied by the user.

6.2.3 Graphical user interface

One of the objectives of the simulation was easier understandability of the DySCAS system. Therefore, a GUI is also developed for this purpose as shown in figure 13.

By pressing the buttons one by one, from top to bottom in sequence, the user can understand the steps involved in the startup behavior.

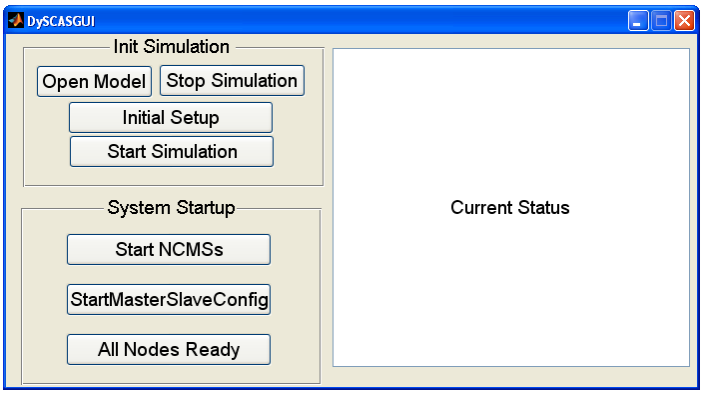

Figure 13: DySCAS Graphical User Interface

6.3 Application quality of service control

This simulation evaluates and demonstrates the capability of a DySCAS system for optimizing system resources with respect to changing system conditions. Figure Annex.2 shows the top level view of the simulations. The application software programs are modeled as periodic entity generators with varying execution time. The execution time is selected based on the input QoS level such as 'QoSLevel_task1' for the first application program. A very simple execution platform consisting of a 'queue' and a 'single server' (a SimEvents^{TM} block) is used to model the processor and the operating system. The time required to run a specific task i.e. an application or a middleware service is based on the assigned execution time (SimEvents^{TM} 'Set attribute' block is used for assigning the execution times). The 'Application' and the 'Instantiation' interfaces are used to forward the values of the required quality of service levels to application programs and sending the processor utilization to the middleware services respectively. The interactions between the middleware services is shown in figure 14.

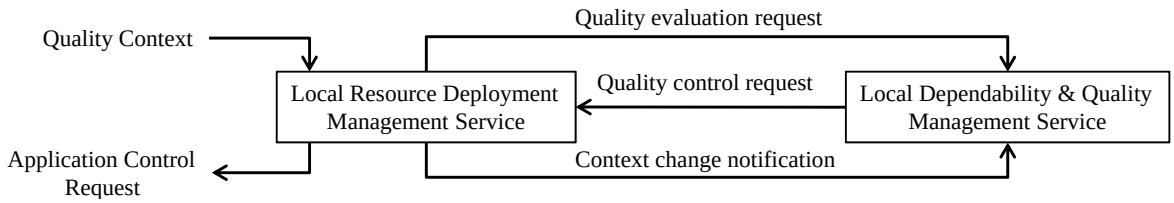

Figure 14: Middleware service interactions for QoS control

Any change in context is notified to the 'Local Dependability and Quality Management Service'(LDQMS) by the 'Local Resource Deployment Management Service'(LRDMS). The LRDMS requests LDQMS for quality evaluation if the overall QoS level (processor utilization for the presented simulation) of the system changes. Based on the evalua-

tion results the LDQMS requests the LRDMS for changes (application qulaity of service for the presented simulations). The LRDMS forwards the request to the application interface.

For the simulations four application tasks are simulated. Each application task is specified with a predefined set of benefit and significance levels for each QoS level. On receiving the new QoS level request from LRDMS, the applications change their QoS level i.e. execution time and period.

Two different algorithms were tested for QoS control. The algorithms and their results are as follows:

6.3.1 Algorithm I

This is a very simple algorithm implemented as an embedded Matlab® function and described by following equation.

$$Commanded\_QoS = \begin{cases} 1 & \text{if actual QoS level} = 3 \\ 2 & \text{if actual QoS level} = 1 \\ 3 & \text{if actual QoS level} = 2 \end{cases}$$

The three actual QoS levels are defined as

$$Actual\_QoS = \begin{cases} 1 & \text{if cpu utilization ¡ } 0.6 \\ 2 & \text{if } 0.6 \text{ ¡ cpu utilization ¡ } 0.8 \\ 3 & \text{if } 0.8 \text{ ¡ cpu utilization ¡ } 1.0 \end{cases}$$

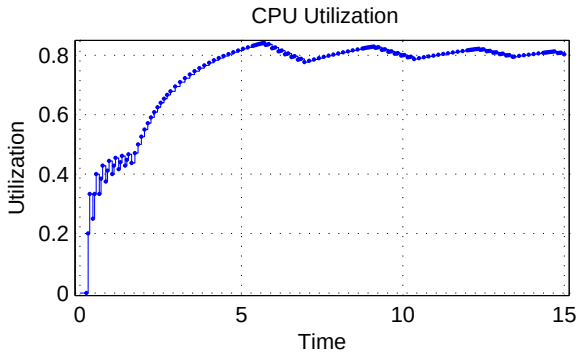

Figure 15: CPU utilization

Figures 15 and 16 show the cpu utilization and the commanded QoS. The initial delay in QoS commands is the time taken by the startup. Moreover, it is important to note that the algorithm does not take account of the benefit and significance levels.

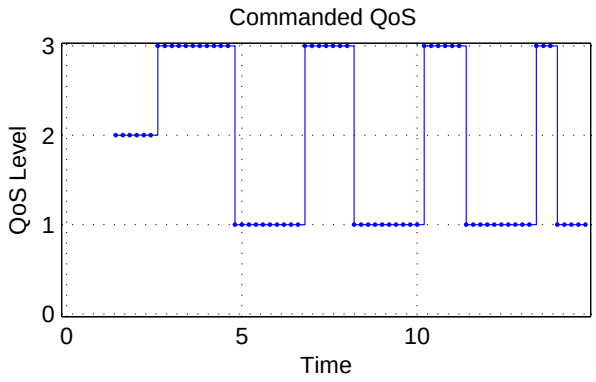

Figure 16: QoS commands

6.3.2 Algorithm II

The interaction between the services is the same as described in figure 14. The difference lies in the fact that instead of sending quality evaluation request, the LRDMS request LDQMS to increase or decrease QoS levels of applications. This algorithm is a part of reconfiguration scheme proposed in [16]. The LDQMS selects one application task based on the implemented algorithms and sends a request back to LRDMS with application id and the required QoS level. The algorithms for increase and decrease of application QoS are presented in the annexure of this report. For detailed description of algorithm the readers are referred to [16].

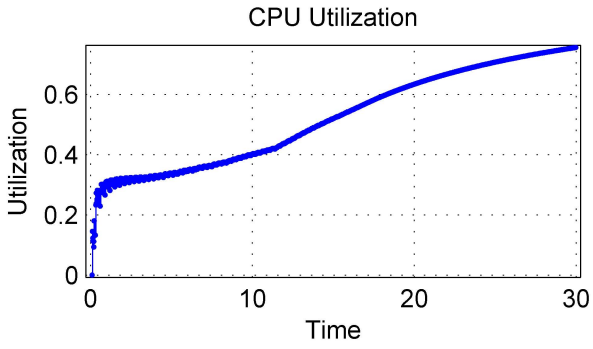

Figure 17: Overall CPU utilization

Figures 17 and 18 show the outputs from one run of simulations where the objective was to keep the highest possible QoS level for each application.

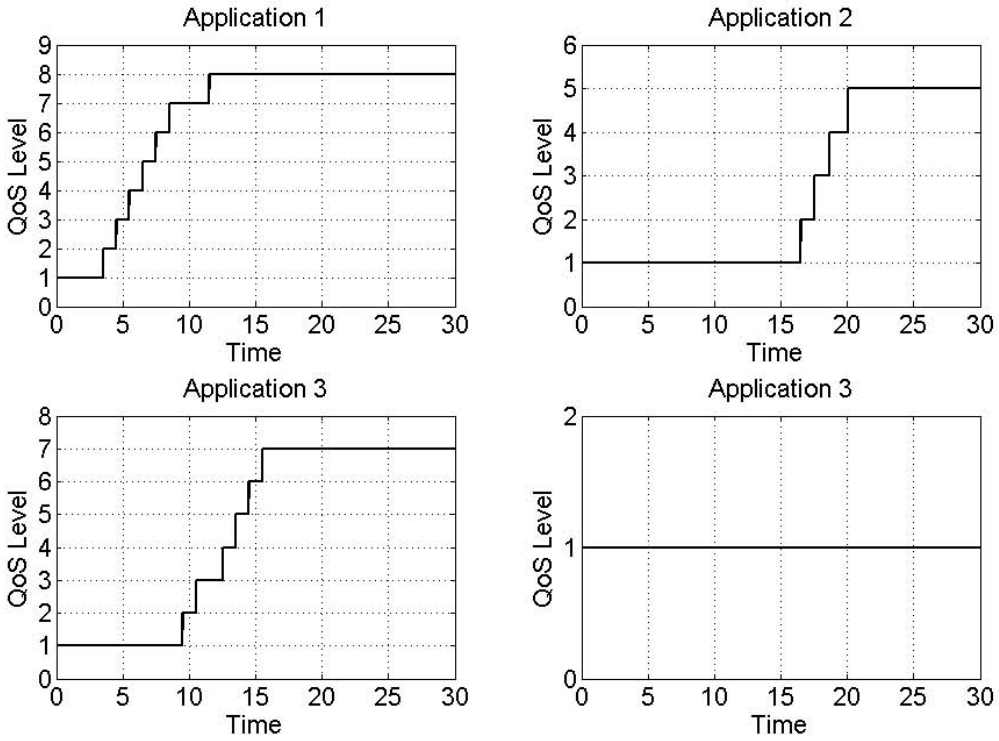

Figure 18: Applications QoS levels

7 SimEvents™ evaluation summary

A general evaluation of SimEvents™ for discrete event simulations is presented in [21]. Furthermore, [12] presents the evaluation of SimEvents™ with respect to DySCAS requirements [9]. This section summarizes the evaluation for simulating a self-managing system in general using SimEvents™. The evaluation is based on the requirements mentioned in section 3.

7.0.3 Component model

With combined usage of Simulink® and SimEvents™ it is possible to simulate a component model. The possibilities include the usage of:

- Simulink® subsystem and a combination of Simulink® (input and output ports) and SimEvents™ (conn port) for structural properties such as ports and interfaces.
- SimEvents™ entity with attributes for signal and its parameters for an interface with multiple signals.

- Stateflow® and SimEvents™ for change of modes and states. This is illustrated in figure 7.

7.0.4 External environment

The SimEvents™ toolbox alone is not capable of simulating the environment external to the system. This is due to the fact that SimEvents™ uses discrete-event simulation (DES) model of computation. However, similar to the component model case, a combined usage of Matlab®/Simulink® and SimEvents™ can fulfill the purpose to a great extent. This may also include HIL (Hardware in loop) and SIL (Software in loop) simulations.

7.0.5 Key abstractions

It is possible to simulate operating system mechanisms such as scheduling and preemption, networks, message and queue handling such as blocking and non-blocking behavior.

The capability of using algorithms is provided by Simulink®. However, dynamic update of algorithm i.e. change of algorithm is not possible. This limitation can be handled to some extent by a pre-defined code for selecting different functions at runtime based on changeable variable.

7.0.6 Dynamic linking and loading

Matlab® only provides a limited support for dynamic loading. This is to our experience not possible with Simulink® and hence with SimEvents™.

7.0.7 Code generation

Code generation is an important aspect for efficient embedded system development. Unlike other toolboxes, code generation is not supported by SimEvents™.

7.0.8 Real-time properties

SimEvents™ supports the simulation of real-time properties in terms of timging. This includes scheduling and canceling timeouts, starting and reading timer for different SimEvents™ entities. It is also possible to use different types of queues together with different types of gates which is useful for simulating constraints such as queue length limitation.

7.0.9 Comprehensibility, visualization and levels of abstractions

It is possible to build a system in hierarchical form by using subsystems. This makes it easier to comprehend a simulated model as well as simulating different levels of abstractions. The graphical representation of state-flow and other scopes such as 'attribute scope', 'signal scope' and 'entity counter' give a good support for the visualization of signals and attributes.

7.0.10 Complexity

For very high levels of abstractions it is easier to manage the complexity. However, with increasing complexity and levels of abstraction, the management of the simulated models becomes difficult. A possibility to handle this short coming is to use configurable masked subsystems for a one large component of a system like a DySCAS core service.

8 Discussion and future work

In this report we have derived requirements for a simulation system for a self-managing system. Simulations of a DySCAS system are also presented. In our previous work [37] we evaluated TrueTime for simulating a DySCAS system. As compared to SimEvents^{TM}, TrueTime is more closer to real implementations due to its support for operating system kernel, networks, mailboxes for communication between different tasks, physical inputs and outputs. Despite this fact, it is difficult to simulate a middleware using TrueTime. Furthermore, additional code has to be written in order to overcome the shortcomings of TrueTime which includes measurement of resources like processor and memory utilization. It is also difficult to visualize and comprehend different activities implemented using TrueTime. Thus it can be concluded that both SimEvents^{TM} and TrueTime have their own advantages and short comings. In TrueTime the complexity increases with the increase in number of tasks and code whereas in SimEvents^{TM} the complexity increases due to the usage of many blocks from its library to implement a single function.

The future work can be divided into three categories:

8.0.11 Improvement in DySCAS system simulations

So far only a small part of the DySCAS specifications has been covered. The extension possibilities include:

- Refinements in the DySCAS core service library by making it easily configurable.

The idea is to have a masked subsystem where different parameters such as queue lengths and timings can be changed by one single dialog box.

- Addition of operating system and network for more fine grained simulations.
- Replacement of the dummy applications with some real life examples such as video streaming.

8.0.12 Tool integration and support

It will be interesting to investigate

- Support for code generation from a SimEvents^{TM} model.
- Integration of TrueTime and SimEvents^{TM}/Stateflow® to complement each other. This may include control of system modes using Stateflow® and actual execution in TrueTime.
- Dynamic loading and linking mechanism to replace the current replications of tasks for simulating transfer and execution of application programs from one node to another in a network.

8.0.13 Algorithms, policies and system modeling

DySCAS type systems require a lot of algorithms and policies for self-management. Due to many closed loops in the system a new trend is to control the computing systems using control systems theory [39]. This includes but not limited to load balancing, quality of service and admission control. One possibility is to develop mathematical models (often required for control systems) for DySCAS systems, new algorithms and investigate their performance with the existing mechanisms. Last but not least is the possibility to work towards the development methodology for efficient policies.

References

[1] Alain Abran, James W. Moore, Pierre Bourque, Robert Dupuis, and Leonard L. Tripp. *Guide to the Software Engineering Body of Knowledge (SWEBOK).* ISO Technical Report ISO/IEC TR 19759, http://www.swebok.org/. 2004.

[2] Richard Anthony and Cecilia Ekelin. Policy-Driven Self-Management for an Automotive Middleware. In *Proceedings of the 1st International Workshop on Policy-Based Autonomic Computing (PBAC 2007)*, Jacksonville, Florida, USA, June 11-15 2007.

[3] Anuja Apte. *Modeling System Architecture and Resource Constraints Using Discrete-Event Simulation*. Matlab Digest, http://www.mathworks.com/, March 2008.

[4] AUTOSAR website. http://www.autosar.org/.

[5] David E. Bakken. *Encyclopedia of Distributed Computing*, chapter Middleware. Kluwer Academic Press, 2001.

[6] Osvaldo Barbarisi and Carmen Del Vecchio. UML Simulation Model for Hybrid Manufacturing Systems. In *13th IEEE Mediterranean Conference on Control and Automation*, pages 358–363, June 2005.

[7] Michael I. Clune, Pieter J. Mosterman, and Christos G. Cassandras. *Discrete Event and Hybrid System Simulation with SimEvents*. Fascicle of Management and Technological Engineering, Volume VII (XVII), 2008.

[8] DySCAS Consortium. *Existing Technologies*. Deliverable 1.1A, DySCAS-Dynamically Self Configuring Automotive Systems, IST project no. FP6-IST-2006-034904. http://www.dyscas.org/doc/DySCAS_D1.1A.pdf, July 2007.

[9] DySCAS Consortium. *Scenario and System Requirements*. Deliverable 1.2, DySCAS-Dynamically Self Configuring Automotive Systems, IST project no. FP6-IST-2006-034904. http://www.dyscas.org/doc/DySCAS_D1.2.pdf, May 2007.

[10] DySCAS Consortium. *Demonstrator Application and Specification*. Deliverable 3.3, DySCAS-Dynamically Self Configuring Automotive Systems, IST project no. FP6-IST-2006-034904. http://www.dyscas.org/doc/DySCAS_D3.3.pdf, February 2009.

[11] DySCAS Consortium. *DySCAS System Specifications*. Deliverable 2.3, DySCAS-Dynamically Self Configuring Automotive Systems, IST project no. FP6-IST-2006-034904. http://www.dyscas.org/doc/DySCAS_D2.3_CoverPage.pdf, February 2009.

[12] DySCAS Consortium. *Evaluation Report*. Deliverable 4.3, DySCAS-Dynamically Self Configuring Automotive Systems, IST project no. FP6-IST-2006-034904. http://www.dyscas.org/doc/DySCAS_D4.3.pdf, February 2009.

[13] MOST Cooperation. *MOST Specification Rev 3.0.* www.mostcooperation.com, 2008.

[14] CORBA website. http://www.corba.org/.

[15] DySCAS website. http://www.dyscas.org/.

[16] Lei Feng, DeJiu Chen, Magnus Persson, Tahir Naseer Qureshi, and Martin Törngren. Dynamic Configuration and Quality of Service in Autonomic Embedded Systems: Theory and Pratice of DySCAS Project. Technical Report TRITA-MMK 2008:12, ISSN 1400-1179, ISRN/KTH/MMK/R-08/12-SE, Mechatronics Lab, Department of Machine Design, KTH, Stockholm, Sweden, 2008.

[17] Lei Feng, DeJiu Chen, and Martin Törngren. Safety Analysis of Dynamically Self-Configuring Automotive Systems. Technical Report TRITA-MMK 2008:13, ISSN 1400-1179, ISRN/KTH/MMK/R-08/13-SE, Mechatronics Lab, Department of Machine Design, KTH, Stockholm, Sweden, 2008.

[18] Lei Feng, DeJiu Chen, and Martin Törngren. Self Configuration of Dependent Tasks for Dynamically Reconfigurable Automotive Embedded Systems. In *47th IEEE Conference on Decision and Control, Cancun, Mexico*, December 2008.

[19] Chien-Lieng Fok, Gruia-Catalin Roman, and Chenyang Lu. Agilla: A Mobile Agent Middleware for Sensor Networks. Technical Report WUCSE200616, Washington University in St. Louis, 2006.

[20] Robert Bosch GmbH. *CAN Specification, Version 2.0.* 1991.

[21] Michael A. Gray. Discrete Event Simulation: A Review of SimEvents. In *Computing in Science and Engineering, Volume 9 Issue 6*, 2007.

[22] Michael González Harbour, José Javier Gutiérrez, José Carlos Palencia, and José María Drake. MAST: Modeling and analysis suite for real time applications. In *Proceedings of the 13th Euromicro Conference on Real-Time Systems, Page(s):125 - 134*, June 2001.

[23] Paul Horn. Autonomic Computing: IBM 's Perspective on The State of The Information Technology. In *AGENDA'01, Scottsdale, AR*, 2001.

[24] HypoVereinsbank and Mercer Management Consulting. *Automobile Technology 2010: Technological Changes to The Automobile and Their Consequences For Manufacturers, Component Suppliers and Equipment Manufacturers.* 2001.

[25] IBM. An Architectural Blueprint For Autonomic Computing. June 2005.

[26] The Mathworks Inc. *SimEvents^{TM}2, Getting Started Guide*. The MathWorks, Inc., 2008.

[27] The Mathworks Inc. *SimEvents^{TM}2, User's Guide*. The MathWorks, Inc., 2008.

[28] Isabell Jahnich, Ina Podolski, and Achim Rettberg. Towards a Middleware Approach for a Self-configurable Automotive Embedded System. In *SEUS '08: Proceedings of the 6th IFIP WG 10.2 international workshop on Software Technologies for Embedded and Ubiquitous Systems*, pages 55–65. Springer-Verlag, 2008.

[29] Jeffrey O. Kephart and David M. Chess. *The Vision of Autonomic Computing*, volume 36, pages 41–50. IEEE Computer Society Press, Los Alamitos, CA, USA, January 2003.

[30] Hyo Jik Lee, Sung Hyun Kim, Hee Sung Park, and Byung Suk Park. Discrete event system simulation approach for a nuclear facility operational analysis. In *proceedings of Innovative Production Machines and Systems, IPRPMS 08*, 2008.

[31] Hock Beng Lim, Bang Wang, Cheng Fu, Phull Arpan, and Di Ma. *WISDOM: Simulation Framework for Middleware Services in Wireless Sensor Networks*. In 5th Consumer Communications and Networking Conference,CCNC, 2008.

[32] Modeling and Analysis Suite for Real-Time Applications (MAST) website. http://mast.unican.es/#intro.

[33] Marius Nica, Lucian, Macedon Ganea, and Gheorghe Donca. *Simulation of Queues In Manufacturing Systems*. Fascicle of Management and Technological Engineering, Volume VII (XVII), 2008.

[34] Martin Ohlin, Dan Henriksson, and Anton Cervin. TrueTime 1.5 - Reference Manual. Department of Automatic Control, Lund University, Sweden, http:// www.control.lth.se/truetime, January 2008.

[35] Carloni Luca P., Passerone Roberto, Pinto Alessandro, and Sangiovanni-Vincentelli Alberto L. *Languages and tools for hybrid systems design*. Foundations and Trends in Electronic Design Automation, 2006.

[36] Tahir Naseer Qureshi, DeJiu Chen, Lei Feng, Magnus Persson, and Martin Törngren. On mapping UML models to Simulink/SimEvents: A Case Study of Dynamically Self-Configuring Middleware. Technical Report TRITA-MMK 2009:05,

ISSN 1400-1179, ISRN/KTH/MMK/R-09/05-SE, Mechatronics Lab, Department of Machine Design, KTH, Stockholm, Sweden, 2009.

[37] Tahir Naseer Qureshi, DeJiu Chen, Magnus Persson, and Martin Törngren. *Simulation Tools for Dynamically Reconfigurable Automotive Embedded Systems - An Evaluation of TrueTime.* at Real-Time in Sweden (RTiS'07), Västerås, Sweden, August 21-22, 2007.

[38] Sanjay Rishi, Benjamin Stanley, and Kalman Gyimesi. *Automotive 2020: Clarity Beyond The Chaos.* IBM Institute for Business Value, 2008.

[39] Karl-Erik. Årzén, Anton Cervin, Tarek Abdelzaher, Håkan Hjalmarsson, and Anders Robertsson. *Roadmap on Control of Real-Time Computing System.* EU/IST FP6 ARTIST2 NoE, Control for Embedded Systems Cluster, http://www.artist-embedded.org/artist/IMG/pdf/18b_Control_Roadmap.pdf, 2005.

[40] P. Šůcha, M. Kutil, M. Sojka, and Z. Hanzálek. TORSCHE Scheduling Toolbox for Matlab. In *IEEE Computer Aided Control Systems Design Symposium (CACSD'06)*, pages 1181–1186, Munich, Germany, October 2006.

[41] Regular Triangle Team. UCS User Guide V1.2.0. Technical report, 2008.

[42] UML website. http://www.uml.org/.

[43] Süleyman Özarslan and Y. Murat Erten. Simulation of Agilla Middleware on TOSSIM. In *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 1–6, ICST, Brussels, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-20-2.

Annexure A: Algorithms for Quality of Service

Input: Task set T, a list of QoS levels of all tasks Q
Output: $success, Q$

```
1  n := |T|;
2  ratMax := 0, taskId := 0;
3  for i=1 to n do
4  |  if Q[i] = T[i].q or Q[i] = 0 then continue;
5  |  Δres := T[i].cpu(Q[i] + 1) − T[i].cpu(Q[i]);
6  |  if Δres ≠ 0 then
7  |  |  Δbft := T[i].sig × (T[i].bft(Q[i] + 1) − T[i].bft(Q[i]);
8  |  |  ratio := Δbft/Δres;
9  |  else
10 |  |  ratio := ∞;
11 |  end
12 |  if ratio > ratMax then
13 |  |  taskId := i;
14 |  |  ratMax := ratio;
15 |  end
16 end
17 if taskId > 0 then
18 |  Q[taskId] := Q[taskId] + 1;
19 |  success := 1;
20 else
21 |  success := 0;
22 end
23 return success, Q;
```

Algorithm 1: Increase the QoS Level of One Active Task

Annexure B: Figures

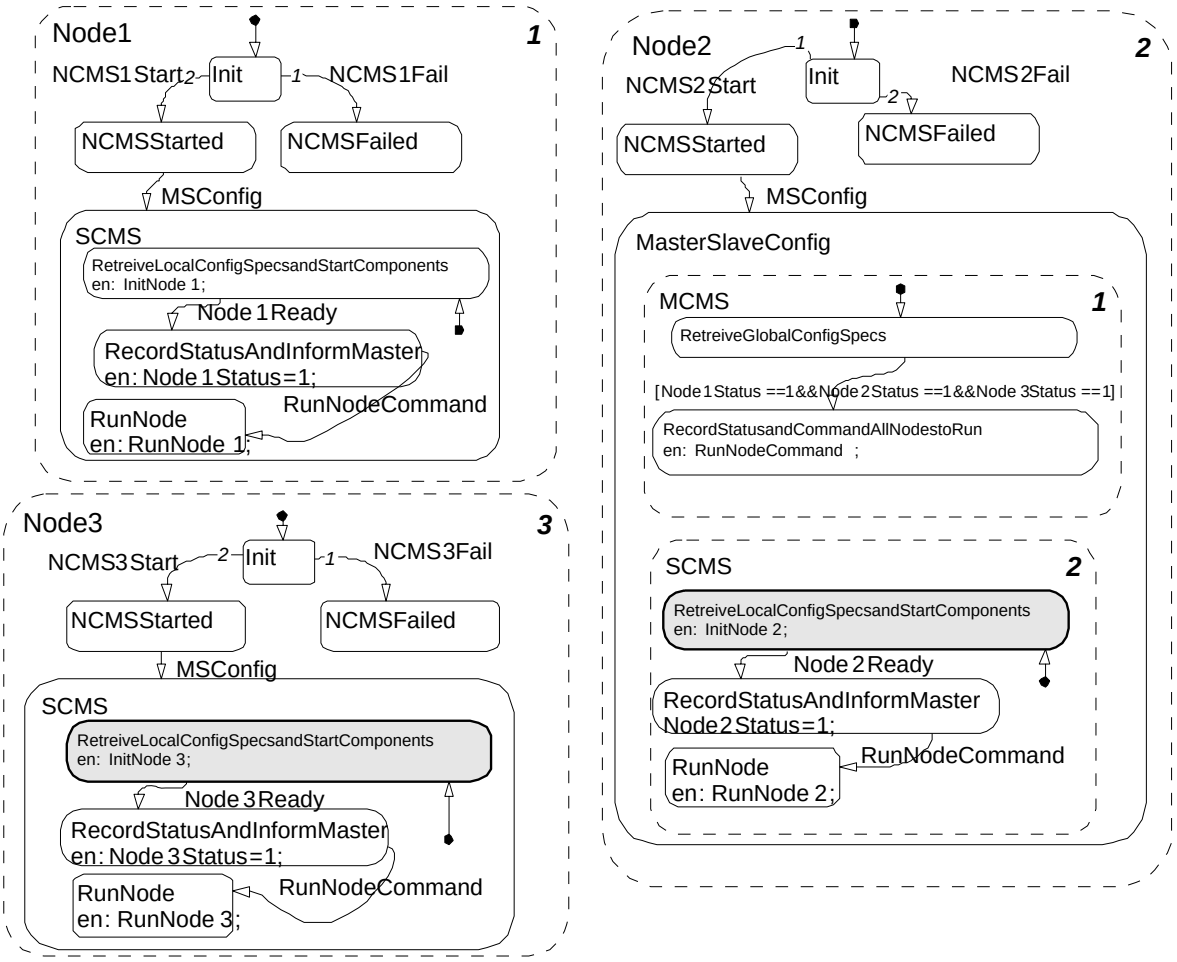

Figure Annex.1: System startup in Stateflow®

Input: Task set T, a list of QoS levels of all tasks Q, resource type $tp \in \{cpu, bw, mem\}$
Output: $success$, Q

```
1  n := |T|;
2  ratMin := ∞, reduce := 0, taskId := 0;
3  for i=1 to n do
4      if Q[i] = 1 then continue;
5      for j=1 to Q[i] − 1 do
6          switch tp do
7              case cpu: Δres := T[i].cpu(Q[i]) − T[i].cpu(Q[i] − j);
8              case bw: Δres := T[i].bw(Q[i]) − T[i].bw(Q[i] − j);
9              case mem: Δres := T[i].mem(Q[i]) − T[i].mem(Q[i] − j);
10         end
11         if Δres ≠ 0 then break;
12     end
13     if Δres ≠ 0 then
14         Δbft := T[i].sig × (T[i].bft(Q[i]) − T[i].bft(Q[i] − j);
15         ratio := Δbft/Δres;
16     else
17         ratio := ∞;
18     end
19     if ratio < ratMin then
20         taskId := i;
21         ratMin := ratio;
22         reduce := j;
23     end
24 end
25 if taskId > 0 then
26     Q[taskId] := Q[taskId] − reduce;
27     success := 1;
28 else
29     success := 0;
30 end
31 return success, Q;
```

Algorithm 2: Decrease the QoS Level of One Task

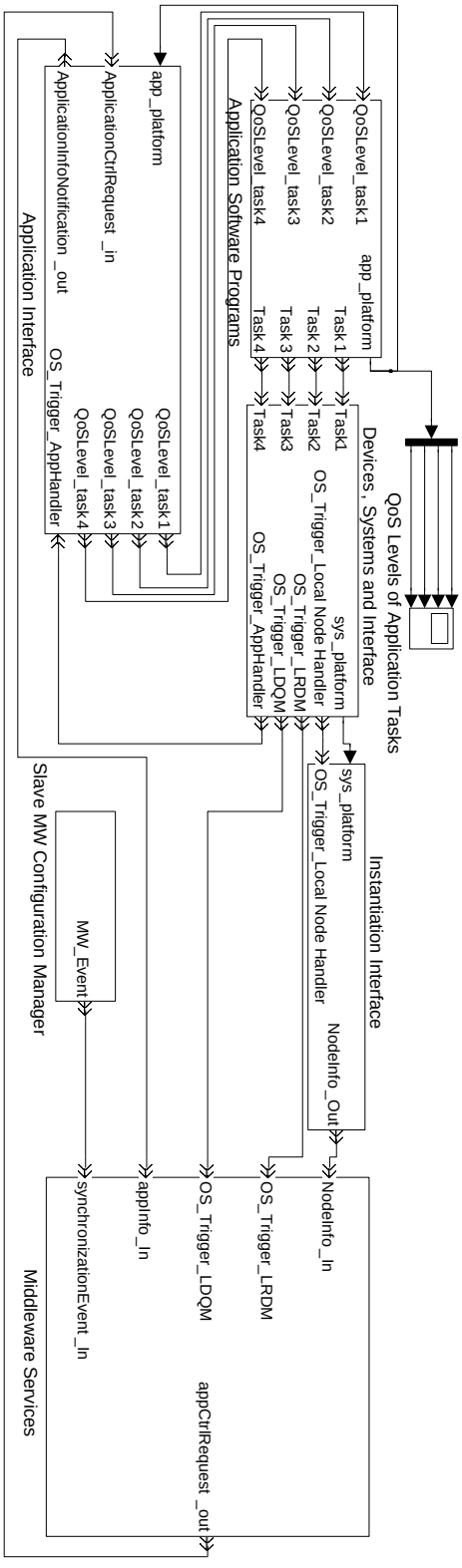

Figure Annex.2: Simulation setup for QoS Control