MODEL-BASED SYSTEM DESIGN OF TIME-TRIGGERED ARCHITECTURES – AVIONICS CASE STUDY

Hauke Fuhrmann   Reinhard von Hanxleden, Christian-Albrechts-Universität zu Kiel, Kiel, Germany
Jörn Rennhack   Jens Koch, Airbus Deutschland GmbH, Hamburg, Germany

Abstract

With the growing complexity of distributed systems, the inter-communication within and between subsystems is increasing. The time-triggered communication approach in the time-triggered architecture (TTA) offers deterministic, fault-tolerant communication services with additional features that enable the developers to better manage complexity and to find design flaws earlier in the development process. However, to fully utilize the advantages of TTA, we advocate the use of a model-based design process. In this paper, we report on experiences with such a model-based design of an avionics system, realized as a TTA using commercial modeling tools and extensions that support the Time Triggered Protocol (TTP). A goal of the Dependable Embedded Components and Systems (DECOS) project, an Integrated Project within the European Union Framework Programme 6, is to explore the integrated distributed time-triggered architecture paradigm. A high-lift flap system serves to validate and demonstrate this paradigm in the aerospace application domain. This paper gives an overview of the application and the system architecture, and describes the model-based development process. However, we also still see room for improvement, in particular regarding the integration and presentation of information for the designer.

Introduction

The aerospace industry has to cope with highly safety-critical computer systems and therefore has to deal with multiple topics:

Embedded systems are rapidly growing in complexity: With the decrease of hardware costs more multi-purpose processors are employed and customized by software implementation for specific applications. Additionally the increase of hardware performance leads to a trend of integrating multiple functions into single electronic control units (ECU) for cost reductions. This approach imposes high demands on software middleware that keeps application programming manageable and results in safe systems. *Real-time* systems have special demands on the response time behavior. To control a dynamic physical system requires reactions at special points in time. Operating systems and application programs need to consider timing behavior in order to achieve determinism in the temporal domain. *Distributed* systems are employed to cope with redundancy and wiring complexity. Shared communication bus systems replace point to point connections. For different applications and safety requirements, different methods for shared media access control have been employed. Sophisticated communication protocols enable us to develop fault-tolerant, deterministic communication networks, but require a decent know-how and effort for setup.

Model-based system design is proposed for development, to cope with the growth of complexity in all of these areas. Graphical representations augment or replace textual implementations for illustrating and documenting parts of a system. The content of high abstraction models can be used by appropriate tool-chains to further process the data and relieve the developer of work in lower abstraction levels. For example models can describe configuration of systems (*e.g.*, describing distribution of the system onto multiple ECUs and the communication behavior between them). Post-processing tool-chains use the model information to automatically configure the operating systems and communication controllers (*e.g.*, building communication schedules in time-triggered communication). Other model types describe the functional behavior of a system. Model suites such as *Matlab Simulink* [1] or *SCADE* of *Esterel Technologies* [2] allow to specify data flow of the application in a way well suited for closed control loop design. The *Statechart* [3] formalism and dialects like *Stateflow* [4] and the synchronous *SyncCharts* [5] (with minor modifications known as

Safe State Machines (*SSMs*) [6]) are used to model control flow of reactive applications. Usually models inherit such precise semantics that they can be executed. On the one hand simulation is used to validate system behavior in a very early stage of the development process, especially prior to physical integration, and on the other hand generators can synthesize code for the target platform in order to directly derive the target implementation from models.

This paper is divided mainly into two sections, of which the first gives an overview about the DECOS project in general and the second describes our contribution, the model-based system design of an aerospace demonstrator within the DECOS project.

Time-Triggered Architecture

For receiving fault-tolerance in highly safety-critical distributed systems much effort must be spent (*e.g.* in system design and error containment) if event-triggered communication buses such as the *Controller Area Network* (*CAN*) [7] are used. These buses are mostly developed for (soft) real-time systems with flexible requirements concerning timing. In contrast, the time-triggered communication approach in the *time-triggered architecture* (*TTA*) [8] offers deterministic, fault-tolerant communication services with additional features that enable the developers to better manage complexity and to find design flaws earlier in the development process.

As TTA is a general notion, there are different implementations available. The *Time-Triggered Protocol* (*TTP*) [9] is a communication protocol for the TTA and specifies its communication subsystem. Other approaches are, for example *FlexRay* [10] and *TTCAN* [11].

The TTP describes the communication scheme of the system. The set of nodes participating in communication via TTP is called the *TTP cluster*. The access to the bus follows the *time division multiple access* (*TDMA*) scheme as it is depicted in Figure 1. The global communication schedule has to be known by every communication controller in the cluster. In a TTA, not only the communication scheme is time-triggered, but the task execution at the individual nodes as well, in order to synchronize task execution with message transfer.

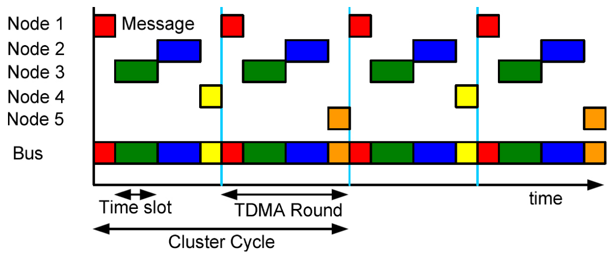


Figure 1 The TDMA scheme of a TTP network

The Dependable Embedded Components and Systems Project

The DECOS project (*Dependable Embedded Components and Systems*) [12] should deploy a process of developing embedded systems that are built on COTS (commercial-of-the-shelf) hardware and software components and nevertheless enable us to develop safe systems.

The key to the problem is to develop fundamental and enabling technologies which are independent of domain and technology in order to facilitate the paradigm shift from *federated* to *integrated* design of dependable real-time embedded systems. This shall lead to reduced development, validation and maintenance costs in both the software and the hardware domain.

The major objective is to investigate the compositional system framework and to develop a set of generic hardware and software components. As DECOS is platform independent, they are usable on various platforms that support the defined core services *deterministic and timely message transport, fault-tolerant clock synchronization, strong fault isolation* and *consistent diagnosis of failing nodes*.

The TTA was chosen because it offers *strong composability, effective fault propagation barriers, fault-tolerance* by active replication, *strong diagnosability* and *formal analysis* of critical architecture functions. The particular choice within the aerospace subproject fell to TTP as it consequently follows the paradigms and safety issues of the TTA and is technically mature.

The DECOS project works at two sides: The technical implementation and the model-based development process.

Technical Implementation

Technical issues are solved in order to enable the shift from federated to integrated systems. The goal is to eradicate the *one-function-one-ECU* paradigm. Instead integrate multiple functions onto single ECUs and multiple independent communication channels onto single physical communication busses. Safety requirements make this goal a challenging one, because natural fault-containment-regions given by the bounds of hardware in federated systems need to be replaced by appropriate software and hardware mechanisms within the integrated ECU. These mechanisms are designed that safety-critical applications can run next to non-safety-critical applications at the same ECU without affecting each other.

The main idea is to build upon the core services of a time-triggered architecture. Therefore DECOS focuses on developing a consistent framework of middleware on top of any appropriate TTA in order to be technology independent and to increase reusability of applications. This is depicted in Figure 2. The technical TTA implementation (as long as it offers the required core services) can be replaced, while keeping all programming interfaces and the tool-environment for the developer.

The basic entity that enables the concurrent execution of different criticality applications is the *Encapsulated Execution Environment* (*EEE*). It is developed in an operating system and offers separation of applications in both temporal and spatial domain into so-called partitions: The utilization of hardware memory protection guarantees the interaction of different partitions only at the explicit interfaces. The employment of a time-triggered operating system with task schedules fixed at design time lead to deterministic temporal behavior at run-time.

Other topics tackled at the DECOS project lead to a consistent full-featured architecture. For performance reasons, a fault-tolerance layer is implemented in hardware: Redundant message transfer via physically separated communication links is initiated and re-merged in hardware controllers in order to relieve the application processor of this periodical task. Additionally virtual communication links are established on top of physical time-triggered communication busses. The whole communication network gets mapped into logical communication links between an arbitrary set of *distributed application subsystem* (*DAS*) parts. The physical network gets abstracted to achieve transparent distribution: Each logical communication link can comprise any connections between physical ECUs, spanning even across separated physical networks, that get connected via special gateways. This stays transparent for the application developer. Only at hardware integration the actual distribution onto physical hardware and their communication links needs to be specified. Logical communication links can even be of event-triggered nature by sending event-triggered messages wrapped in physical time-triggered communication. This especially helps to integrate legacy software that formerly used for example CAN communication into this architecture. Additionally the architecture is augmented by a diagnosis subsystem that continuously checks the states of messages to provide diagnosis information to the application.

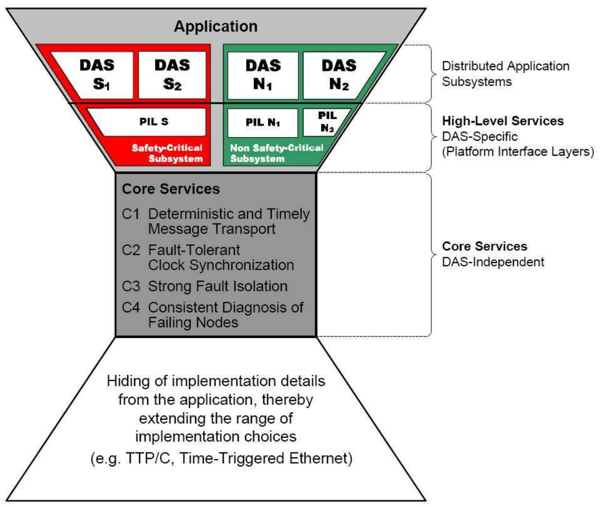

Application
DAS S1
DAS S2
DAS N1
DAS N2
PIL S
PIL N1
PIL N2
Safety-Critical Subsystem
Non Safety-Critical Subsystem
Distributed Application Subsystems
High-Level Services
DAS-Specific
(Platform Interface Layers)
Core Services
C1 Deterministic and Timely Message Transport
C2 Fault-Tolerant Clock Synchronization
C3 Strong Fault Isolation
C4 Consistent Diagnosis of Failing Nodes
Core Services
DAS-Independent
Hiding of implementation details from the application, thereby extending the range of implementation choices
(e.g. TTP/C, Time-Triggered Ethernet)

Figure 2 DECOS service hierarchy

Development Process

The technical issues outlined above are developed to show the feasibility of such an architecture. A concurrent task in the DECOS project is to build up and increase usability of this approach. Major drivers are cost reductions in all domains. Depending on the level of criticality, certification according to DO-178B [13] increases the cost of developing software by 300–500% [14]. Modular certification [15] is a certification strategy that promises a massive reduction in certification cost through modularization and reuse of certification arguments. The DECOS architecture offers modular certification by separating the certification of architectural services from applications and by supporting independent safety arguments for different DASs. The clear interface between DECOS architecture and application software allows to prevalidate the DECOS architecture items and the safe separation of different DASs allows to independently generate certification arguments for each DAS.

In order to enable the employment of the DECOS architecture in industrial applications, a development process is required that can handle the complexity of the architecture. Therefore within the DECOS project a complete toolchain is developed to lead to a seamless process where as much as possible is automated and the tools are integrated. A Key paradigm is the model-based system design in different steps as outlined in Figure 3.

Development is separated into two "swimlanes": SW development and HW-SW integration.

HW-SW Integration

After defining the system requirements, a Platform-Independent Model (PIM) has to be generated. This UML diagram following the PIM metamodel (defined in the DECOS project as a stereotyped UML class diagram) defines application specific but platform independent configuration of the system.

This includes separation of the systems to *jobs*, system resources like sensors and actuators, communication between jobs, assertions to message variables (*e.g.* restrictions or acceptance criteria) for diagnosis of the final system and performance (*e.g. message and task periods*) and dependability (*e.g. redundancy degrees*, *safety integrity levels*) issues.

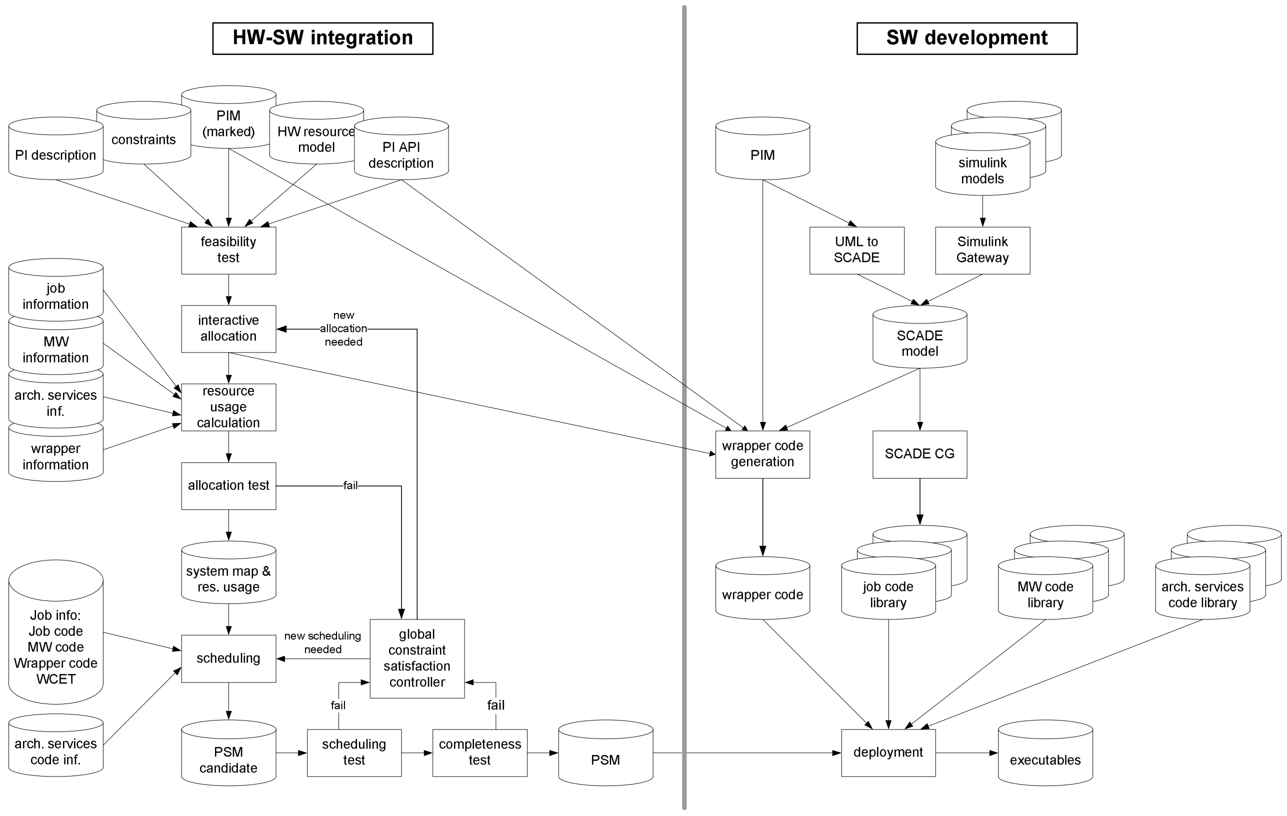


Figure 3 Steps in the DECOS development process

This model is platform independent, because it comprises no hardware specific information. Separation to different jobs is done completely transparently and only reflects the application requirements concerning functional distinction but does not necessarily indicate the physical distribution of the system. Therefore the used hardware is provided by a separate HW resource model and the developer has to follow an interactive process. The process combines these models and defines the system hardware and distribution, where one step is the mapping of jobs to physical ECUs. The toolchain comprises all hardware and software configuration tools, including schedulers for both the global communication schedules and the local ECU task schedules. All information gets merged in the final *Platform Specific Model* (*PSM*), which stores the information for deployment. This approach of separating application specific and hardware specific information allows defining the application prior to committing to specific hardware. Hence for the same application, hardware could be replaced later on. These changes do not affect the PIM, which increases reusability potentials.

Software Development

The other "swimlane" tackles functional SW development. UML State Machines lack formal semantics and therefore inherit many unclarities that deem them questionable for modeling of safety critical systems [16]. Therefore common practice to model functional behavior is to use special modeling suites dedicated for modeling (data flow and control flow (Statecharts) style), simulation and code generation. The DECOS project employs the *SCADE* tool suite from *Esterel Technologies*. It offers data flow modeling for discrete time systems and control flow modeling with its *Safe State Machines*. Semantics are inherited by the underlying synchronous Languages Lustre [17] and Esterel [18]. The formal semantics enable formal analysis on the one hand, *e.g.* formal verification with model-checking [19], and on the other hand such clear code generators, that they were qualified to aerospace and automotive requirements (*e.g.* DO-178B).

Next to SCADE the multi purpose tool suite Matlab/Simulink/Stateflow is established in industrial functional modeling. A main difference is that it supports also continuous time modelling that is necessary to correctly reflect and simulate many physical phenomena. Hence Simulink can be employed to model a non-linear mechanical environment. Simulink is widely used in industry and therefore many prototypes get modeled in Simulink first to get quick results. Therefore a Simulink import utility has been developed further within DECOS. This Simulink-SCADE Gateway transforms discrete time models from Simulink to SCADE and this way Simulink models can be utilized in the DECOS development process.

The Aerospace Subproject

The DECOS project is partitioned in different subprojects, of which one part develops the architecture, hardware, tools and processes and the other part builds industrial demonstrators. For demonstration of the research and development results of DECOS, several test benches get implemented employing the new technology. The authors contribute to the aerospace demonstrator subproject and therefore build up a test-rig that validates the achieved results of DECOS and demonstrates their practicability in a highly safety-critical aerospace application environment. For actual tool-development and improvement (also outlined in this paper) we are cooperating with and advising the other subprojects.

The demonstrator implements an electronically synchronized *high-lift flap system*. This system is motivated by the current state-of-the-art for such systems at Airbus [20].

As depicted in Figure 4, a flap system consists of two flap panels at each wing. The flap system can increase the concavity of the wing if activated and therefore increase the high-lift temporarily. This is used at low speed for landing and take off purposes only. Hence a flap system is safety-critical. If the system fails during the flight, it will not be available for landing, which is obviously a serious problem. If the left and right flap panel are not perfectly synchronized, the ascending force on the two wings differ. This compromises the controllability of the plane and might ultimately lead to a crash.

To avoid the asynchronous state of the flap panels, they need to be synchronized. In the state-of-the-art flap system this is done mechanically: A tight mechanical shaft physically connects the left wing flap panel with the right wing flap panel. This shaft lies across the whole aircraft fuselage. A central power control unit actuates the shaft with the help of two electrical motors. If one motor fails, the other side will move the whole shaft with half the speed by a speed summing differential. This way both sides are always perfectly synchronized unless a shaft breaks or blocks, whereupon the shaft brakes freeze the system. However, this scenario is highly unlikely. The drawback of this solution is obvious: The shaft across the fuselage is very inflexible and the development of the tip-to-tip shaft transmission is very laborious and includes the internal construction of the fuselage into the development of the flap system, which makes the system neither modular nor reusable.

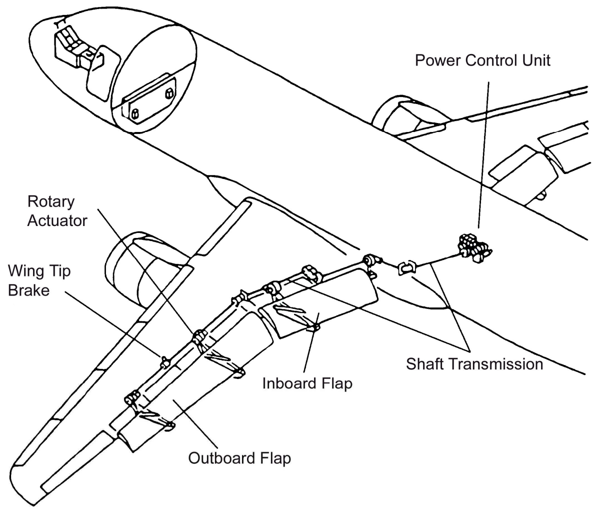


Figure 4 State-of-the-art flap system [20]

An electronically synchronized flap system could give more flexibility by introducing modularity and therefore increase both maintainability and availability, reduce installation costs and enhance agility by using flaps for emergency maneuvers.

The demonstrator is a test platform for the aerospace domain to realize the implementation, test and integration of tools, methods and components (hardware and software) being developed in the DECOS project and to validate the achieved results and demonstrate their practicability in this highly safety-critical application environment.

The Test Bench

The demonstrator uses two of the flap panels: One gets implemented by a physical test rig of a flap panel, which exists at the Technical University Hamburg-Harburg (TUHH). The other panel is implemented as a real-time simulation modeled in Matlab/Simulink and executed on computers with the same communication interfaces than the physical panel.

Figure 5 shows the architecture of the demonstrator. The physical test rig comprises all relevant elements of the flap panel. The left and right side of each flap panel are still connected via a mechanical shaft, while the shaft across the fuselage gets removed. Each shaft side is powered by a powerful electronic motor. The motor is controlled via the *motor control electronics* (*MCE*) whereas *actuator control electronics* (*ACE*) control the synchronization process and form an outer control loop. *Position pick-off units* (*PPU*) measure the current angle of the flap shaft and a hydraulic *cross shaft brake* (*CSB*) is able to fix the shaft in case of faults. The ACEs and PPUs communicate via a time-triggered bus in order to exchange information for the synchronization control loop. The ACEs control the CSB and only if both ACEs on each side indicate correct functionality, the CSB is released.

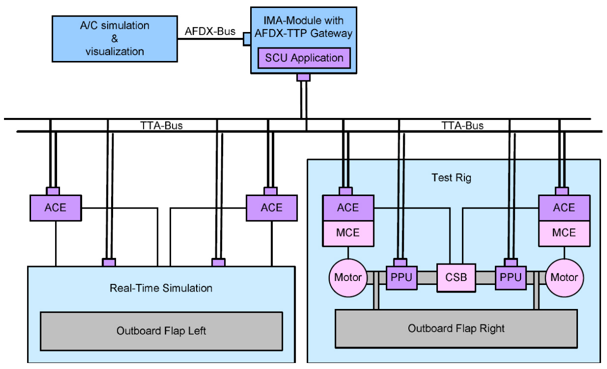


Figure 5 The architecture of the high-lift flap system demonstrator

A central *system control unit* (*SCU*) monitors the behavior of the system and commands the desired flap angles. The SCU is implemented as an application on an IMA (integrated modular avionics) module which has the interface to the TTA bus on the one hand and the interface to the event triggered communication bus *AFDX* (Avionics Full Duplex Switched Ethernet) [21] on the other hand. Through the AFDX channel other aircraft systems are able to communicate with the SCU application and could even communicate directly with the TTA nodes by the AFDX-TTP-gateway functionality. In our demonstrator, a rudimentary aircraft simulation running on a standard PC with AFDX interface communicates with the SCU.

Model-Based System Design

The aerospace flap system demonstrator gets developed using the DECOS development process outlined above. This employs model-based system design. The so-called Virtual Test-Bench is an abstract simulation model representing the physical aerospace test-bench. It permits to simulate the overall test-bench behavior, integrate and test the functional components (HW/SW) of all project partners in this model and enables to validate the design at a high level of abstraction. The architecture design tools and validation methods developed in DECOS are applied wherever applicable. With aid of the simulation model we performed a concept validation of the physical test-bench before developing the individual components. Finally, the physical test-bench will be validated and tested against the virtual simulation model.

Within the DECOS process both Matlab/Simulink and SCADE are employed, because they are used for different purposes: the first for the design of continuous models, the latter for the design of discrete models for qualified code generation and formal analysis. Hence both tools were used at our Virtual Test-Bench, too. In order to get a seamless integrated development process, SCADE offers methods for interaction with Simulink. These features were employed for our Virtual Test-Bench and the resulting process will be explained in the following, illustrated by Figure 6.

The final goal was the *overall simulation model*, a functional model that comprises mainly three elements: (1) The application controller model that is the actual system-under-development (SUD) and that will be used for code generation for the HW-SW-integration of the physical components. For validation of the controller concepts, simulation of the model is required prior to testing of the generated code at physical hardware. (2) For realistic and significant simulation results, the application controller part needs to be fed with inputs corresponding to reactions of the mechanical environment of the system. Therefore an extra model was developed by TUHH in a former project that represents the whole mechanical subsystem of the flap panel and considers all relevant dynamic effects: main natural frequencies, damping, friction and backlash of the mechanics. This model is integrated in an mechanical environment model, which is interfaced with the controller model and hence consumes the controller commands and produces feedback of the sensors. (3) To control the simulation itself either interactively or in a batch mode, a custom graphical user interface was required. This is the third part of the overall simulation model and connected to both, the controller model and the environment model. The three model parts including the development process will be explained in more detail in the following sections.

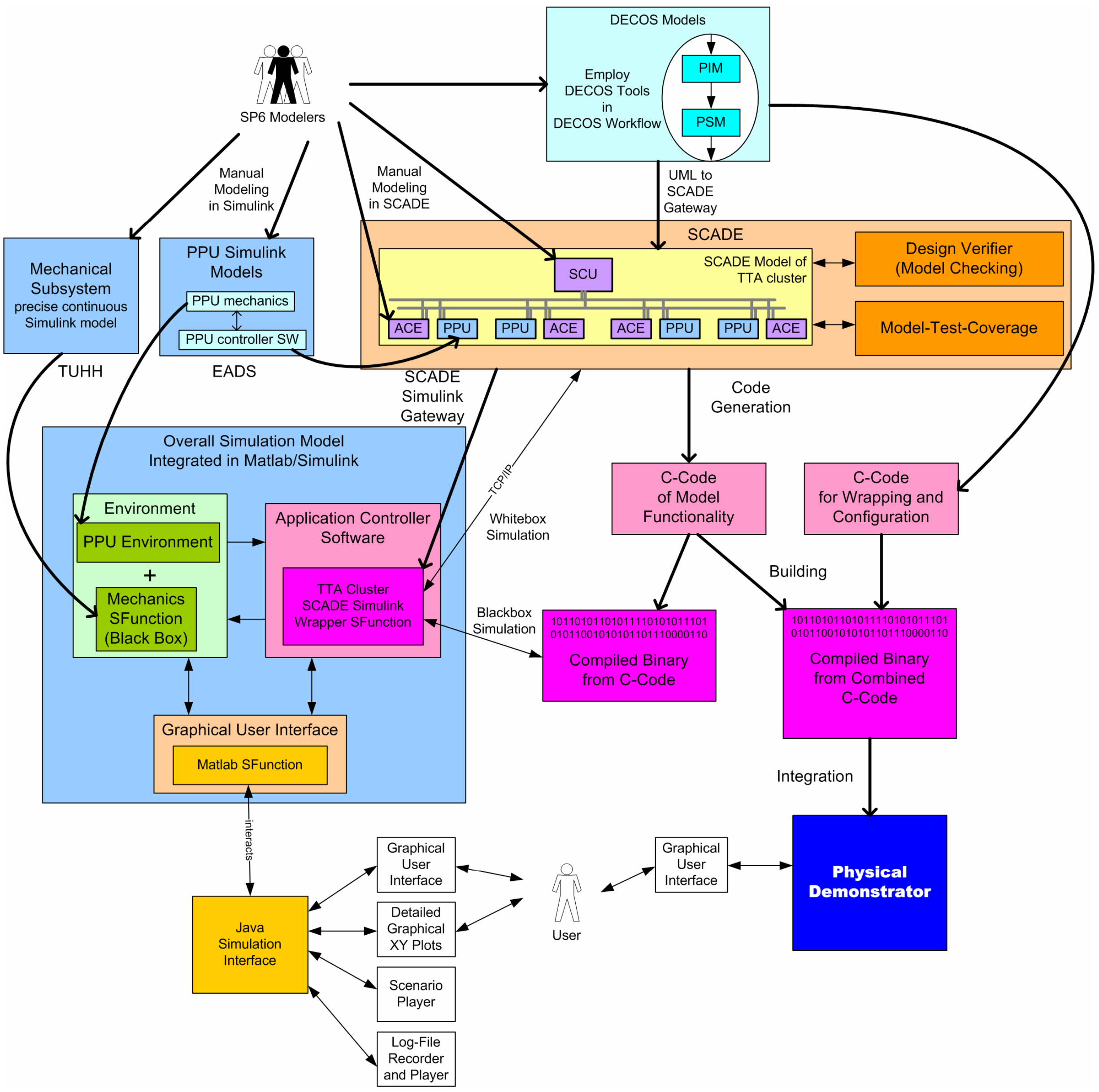
Environment Model

The environment model is of continuous time character, because it represents a non-linear dynamical physical system. The overall simulation model integrates especially this continuous time model and therefore needs to be able to execute continuous time simulations. Hence the overall simulation model itself is a Simulink Model.

As the electronic synchronization is a new feature of the flap system, the model of the mechanical subsystem needed to be adapted to this new approach. Especially the smart sensors, the PPUs, were created in Simulink (including a standalone mechanics and controller part).

Application Controller Software

The model for the application controller part was required to be implemented in SCADE, due to the possible qualified code generation and formal analysis. Modeling started in following the DECOS process outlined in Figure 3 and the corresponding section above. Hence we built a Platform Independent Model (PIM) for the flap system that specifies the distribution and communication scheme of the system. The functionality was separated into distinct jobs and interfaces, ports and messages were created with all required attributes to define the communication behavior and access to hardware resources like sensors and actuators.



DECOS Models
Employ DECOS Tools in DECOS Workflow
PIM
PSM
SP6 Modelers
Manual Modeling in Simulink
Manual Modeling in SCADE
UML to SCADE Gateway
Mechanical Subsystem
precise continuous Simulink model
PPU Simulink Models
PPU mechanics
PPU controller SW
TUHH
EADS
SCADE
SCU
SCADE Model of TTA cluster
ACE
PPU
PPU
ACE
ACE
PPU
PPU
ACE
Design Verifier (Model Checking)
Model-Test-Coverage
SCADE Simulink Gateway
Code Generation
Overall Simulation Model Integrated in Matlab/Simulink
Environment
PPU Environment
+
Mechanics SFunction (Black Box)
Application Controller Software
TTA Cluster SCADE Simulink Wrapper SFunction
TCP/IP
Whitebox Simulation
C-Code of Model Functionality
C-Code for Wrapping and Configuration
Building
Blackbox Simulation
Compiled Binary from C-Code
Compiled Binary from Combined C-Code
Graphical User Interface
Matlab SFunction
Integration
interacts
Java Simulation Interface
Graphical User Interface
Detailed Graphical XY Plots
Scenario Player
Log-File Recorder and Player
User
Graphical User Interface
Physical Demonstrator

Figure 6 Overview of the Virtual Test-Bench with the SCADE and Simulink interaction

Although the complexity of the application seems to be not very high (9 ECUs participate in the TTA, with only three different types), the PIM consists of many hundreds of objects and more than 2000 relations between them. This was hardly manageable within a multi-purpose UML editor and therefore a domain specific PIM editor was created within DECOS to facilitate this task.

A UML-SCADE gateway can be employed to automatically create a SCADE skeleton model from the UML system description. Jobs in the PIM become empty SCADE nodes, which need to be filled with the functional model content in SCADE itself. The SCADE model shall be employed in the simulation and therefore the jobs needs to be connected. The UML-gateway automatically creates a SCADE base model that integrates all job nodes and interconnects them corresponding to the communication scheme of the PIM. Access to system resources (sensors/actuators) in the PIM become inputs and outputs of this SCADE base model and form the interface to the environment during simulation and the interface to hardware device drivers at code generation.

The jobs of the system control unit (SCU) and the actuator control electronics (ACE) were modeled directly in SCADE. The SCADE language notation was used for describing data flow, especially for modeling the control loops of ACE and PPU. Safe State Machines were mainly employed in the SCU to model the complex monitoring and fault reaction strategies of the system. Nevertheless it is not always an obvious decision between data flow and control flow characteristics of a system part. Especially the monitoring subsystem shows properties of both worlds and therefore it was sometimes difficult to decide which part to model in the SCADE language notation and which part in a Safe State Machine. The upcoming fusion of data flow and control flow models [22] in future SCADE releases is therefore much appreciated.

Only the position pick-off units (PPU) are created in Simulink. A Simulink-SCADE gateway is able to semi-automatically transform the Simulink model into a SCADE model. Although such gateway was already available, it gets augmented and adopted within the DECOS project to fit to the needs of this process. Especially the import of multiple Simulink models into one SCADE model is a new feature and requires automatic namespace creation to prevent label overlapping from different models.

The final integrated SCADE model comprises all functional behavior of the controller ECUs and represents the distribution of the system. In the final version it might even represent the time-triggered communication by connecting the different jobs within SCADE not only with direct links but with delay and job-trigger operators (so-called condact). That yields to a model representing the communication exactly as specified in the configuration (PIM) and the schedules (PSM). Therefore formal analysis could be employed to proof system properties even across communication borders and even for the response time domain.

To the SCADE model formal verification methods can be applied to increase the confidence in the system implementation. The SCADE suite supports both, model-checking and *model-test coverage* (*MTC*) [23].

Graphical User Interface

In order to fully customize a graphical user interface (GUI) of the overall simulation model, we implemented it in the Java programming language. In Simulink so-called SFunctions are used to implement custom functions. Those SFunctions can be implemented in C, C++ and the Matlab programming language. The Matlab language itself allows to call methods in Java objects. This feature is used here to generate an interface between Simulink and the custom Java application: A Simulink wrapper SFunction stores all input values from environment and controller, calls the Java application and passes the values each major simulation step and vice versa. The application itself consists of four parts and because the GUI is only one part of it, it is called the Java Simulation Interface (JSI). The graphical user interface can be used to interactively control the simulation by a user. For example pilot commands or different airspeeds can be applied. The environment model can simulate many different faults of the mechanical subsystem, such as motor disconnection, powered runaway, shaft blockade or brake freeze. These faults can be manually injected by the user via the GUI. Graphical plots display the history of relevant system variables for analysis of simulation runs. A scenario player can load predefined flight scenarios and execute the simulation independently without user interaction. Finally log files of simulation values can be recorded and replayed in order to display a simulation run without performing the computation prone simulation itself again.

Code Generation

For deployment, C code of multiple different sources needs to be integrated as shown in Figure 3. This comprises the configuration of the distributed time-triggered architecture in the PSM, functional code from the SCADE models, wrapper code integrating this functional code and code from libraries for middleware (*e.g.* redundancy management) and architectural services coming from the operating system. In Figure 6 this is summarized to functional code directly generated from the SCADE models and code for configuration and wrapping generated from libraries and the HW-SW-integration toolchain of DECOS. These files become combined and compiled into the individual binaries for the distributed ECUs.

Simulation

As mentioned above, the different model parts finally are integrated in the overall simulation model in Simulink. In order to execute the whole simulation in Simulink, one need some possibility to execute a simulation of the application controller model that is modeled in SCADE. Fortunately such feature is provided directly by the SCADE suite. Analogical to the GUI SFunction, SCADE creates an SFunction, as well, that is embedded into the overall simulation model. This SFunction is implemented in C/C++ and therefore able to call other C/C++ procedures or library functions. Two different kinds of simulation modes are available: (1) *Whitebox* simulation uses TCP/IP to connect to the SCADE simulation tool itself and let Simulink and SCADE exchange data on-the-fly during the simulation run. While the Simulink model part is executed in Simulink, the SCADE part is automatically opened in SCADE and simulated using the SCADE simulation engine. At each simulation step, Simulink calculates its block's outputs, passes them to SCADE, which in turn consumes these as inputs, calculates its own outputs and passes them back to Simulink for the next step. This way, the SCADE simulation viewer can be used to display all details of the SCADE model during simulation for debugging. Internal variables of the data flow part can be viewed as well as the internal states of the statecharts. (2) *Blackbox* simulation can be used to do simulation without SCADE and to speed up simulation time. Therefore prior to simulation the SCADE code generators are employed to generate the functional C code of the model. This code automatically gets compiled into a binary C-library, which represents the functional behavior of the SCADE model. During simulation, the SFunction within Simulink interacts only with this binary C-library: At every simulation step, the Simulink outputs are passed to the library, a step function is called that executes one step of the internally represented SCADE model and the outputs are returned to Simulink. This is much faster than the whitebox simulation and the simulation can be done without concurrent execution of SCADE.

Development Results

The high-lift flap system of the aerospace demonstrator of DECOS was successfully modeled using the process outlined above. We were using both SCADE and Simulink, the first for formal analysis and code generation and the second for simulations with a complex continuous time environment model. This enabled us to validate the concepts for the physical high-lift flap system on a very high level of abstraction and an early stage of the development process, especially prior to physical testing of physical hardware.

Combined with the UML based system configuration modeling and the comprehensive toolchain of the DECOS project, the development becomes an integrated and nearly seamless process. This approach allows the developer to stay on a very abstract level of the system, because many protocol and hardware specific configurations are done automatically by the sophisticated tool chains. Unfortunately the experience shows that this approach, while it eases the development as such, it also reduces insights into the time-triggered behavior of the system. Although static schedules predefine the points in time when messages are sent and tasks are triggered, this introduces a fixed delay in the flow of information from one task over TTP messages to another task.

An example is given in Figure 7: The upper diagram shows the flow of information from an event recognized by some sensor until the reaction of the system by an actuator. Here the information needs to be processed by three distributed tasks, t1, t2 and t3, which communicate by sending the messages m1 and m2. The lower diagram shows the end-to-end latency from event to reaction as to be the sum of fixed delays and actual processing times in this flow of information. Only one set of task schedules and communication bus schedule is displayed, but obviously the end-to-end latency highly depends on the details of these schedules and does not necessarily relate to the message period of the tasks. Commercial toolchains like from TTTech [24] split up responsibility of the different schedules: The global communication schedule is done by the system integrator and the local task scheduled for the hosts are done by the subsystem suppliers in order to be as much independent of the integrator and other subsystem suppliers as possible. This approach complicates the situation for response time analysis, because the end-to-end latency is a product of all schedules together and therefore it can neither be analyzed by one of the partners alone nor a schedule-set with a certain latency be found. The only solution is to restrict the task-schedulers to very small allowed task intervals.

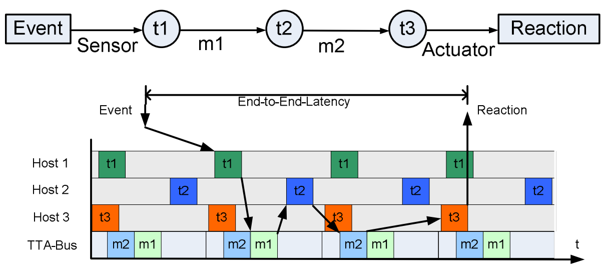


Figure 7 End-to-end-latency in the time-triggered architecture

To improve this situation, we would appreciate a solution that allows to review how information beginning at some event in the real world flows through the system, in order to analyze the response time until the reaction of some part of the system. For example, this would allow the detection of excessive overall latencies, for example because a system might introduce flows of information that pass arbitrary many TDMA-rounds. Another scenario that would be detected easily would be an over-restricted system, where the required information flows are restricted to specified TDMA-round-generations that cannot be scheduled at all.

Conclusion

For the aerospace industry as well as for other industrial sectors, the results of the DECOS project are needed to provide a fundamental and comprehensive basis for future innovations within utility systems.

We demonstrate the practicability of the DECOS results by implementing the safety-critical high-lift flap system. We have shown the model-based development process exemplary for this application and have outlined how UML modeling for system configurations and functional modeling on the basis of synchronous languages in SCADE get combined. Benefits stem from both worlds: Standardized tools and interfaces for UML models ease the development of integrated toolchains, while the formal semantics of synchronous languages enable formal verification and qualified code generation and fits the deterministic philosophy of the time-triggered architecture. Major goals are time and cost reductions in the development of safety-critical applications. In DECOS appropriate architectures, methodologies, associated COTS hard- and software components and comprehensive toolchains are developed which fulfil the requirements of the aerospace industry. These components and tools cover cluster design, middleware and code generators, validation and certification as well as systems-on-a-chip (SoCs) for high dependability applications. Furthermore technology-invariant software interfaces and encapsulated virtual networks with predictable temporal properties will be developed such that application software can be transferred to a new hardware and communication base with minimal effort (legacy reuse).

To summarize, we consider the synchronous modeling paradigm to be a nice complement and enhancement to the TTA approach of designing distributed real-time systems. The current state of this technology, regarding theoretical understanding as well as practical tool support, already makes it feasible to develop such systems. However, we still see room for improvement, in particular regarding the integration and presentation of information relevant for the designer.

References

[1] Mathworks Inc., Simulink - Simulation and Model-Based Design, Mathworks Inc., 2005, http://www.mathworks.com/access/helpdesk/help/pdf_doc/simulink/sl_using.pdf

[2] Esterel Technologies, Company homepage, http://www.esterel-technologies.com.

[3] D. Harel, Statecharts: A visual formalism for complex systems, Science of Computer Programming, vol. 8, no. 3, pp. 231-274, June 1987.

[4] T. MathWorks, Inc., Stateflow and Stateflow Coder User's Guide, Version 6, Natick, MA, September 2005, http://www.mathworks.com/access/helpdesk/help/pdf_doc/stateflow/sf_ug.pdf

[5] C. André, SyncCharts: A Visual Representation of Reactive Behaviors, I3S, Sophia-Antipolis, France, Tech. Rep. RR 95-52, rev. RR (96-56),Rev. April 1996,

http://www.i3s.unice.fr/andre/CAPublis/SYNCCHARTS/SyncCharts.pdf

[6] C. André, Semantics of S.S.M (Safe State Machine),Esterel Technologies, Sophia-Antipolis, France, Tech. Rep., Apr. 2003, http://www.esterel-technologies.com, in the download section.

[7] ISO 11898. Road Vehicles - Interchange of digital information – Controller area network (CAN) for high speed communication, International Standards Organisation, 1993.

[8] H. Kopetz and G. Bauer, The time-triggered architecture. Proceedings of the IEEE, vol. 91, no. 1, pp. 112-126, 2003.

[9] H. Kopetz and G. Grünsteidl, TTP - a time-triggered protocol for fault-tolerant real-time systems, Institut für Technische Informatik, Technische Universität Wien, Treilstr. 3/182/1, A-1040 Vienna, Austria, Tech. Rep., 1992.

[10] R. Belschner, R. Mores, G. Hay, J. Berwanger, C. Ebner, S. Fluhrer, E. Fuchs, B. Hedenetz, A. Krüger, P. Lohrmann, D. Millinger, M. Peller, J. Ruh, A. Schedl, and M. Sprachmann, FlexRay: The communication system for advanced automotive control systems, in SAE 2001 World Congress. Detroit, USA: Society of automotive Engineers, Mar. 2001.

[11] T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther, Time triggered communication on CAN, http://www.can-cia.org/can/ttcan/fuehrer.pdf, 2000.

[12] DECOS - Dependable Components and Systems, Research project homepage, https://www.decos.at/.

[13] DO-178B - Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE, Dec. 1992.

[14] R. Atkinson, COTS tools reduce the cost of embedded software certification, COTS Journal, 2002.

[15] J. Rushby, Modular certification, SRI Computer Science Laboratory, Tech. Rep., 2001, http://hdl.handle.net/2002/14483

[16] H. Fecher, J. Schönborn, M. Kyas, and W. P. de Roever, 29 new unclarities in the semantics of UML 2.0 state machines. in ICFEM, ser. Lecture Notes in Computer Science, vol. 3785. Springer, 2005, pp. 52-65.

[17] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, The synchronous data-flow programming language LUSTRE, Proceedings of the IEEE, vol. 79, no. 9, pp. 1305-1320, September 1991, http://citeseer.nj.nec.com/halbwachs91synchronous.html

[18] G. Berry, The Foundations of Esterel, Proof, Language and Interaction: Essays in Honour of Robin Milner, 2000, editors: G. Plotkin, C. Stirling and M. Tofte

[19] A. Bouali, B. Dion, and K. Konishi, Using formal verification in real-time embedded software, in JSAE Annual Congress, 2005.

[20] T. Neuheuser, B. Holert, and U. B. Carl, Elektrische Antriebssysteme für ein zentrales Landeklappenelement, in Deutscher Luft- und Raumfahrtkongress, vol. DGLR-JT 2002-192, Stuttgart, 2002.

[21] ARINC 664, Aircraft Data Networks, Part 7 Deterministic Networks, ARINC, Annapolis, Maryland, USA, http://www.arinc.com

[22] J.-L. Colaço, B. Pagano, and M. Pouzet, A Conservative Extension of Synchronous Data-flow with State Machines, in ACM International Conference on Embedded Software (EMSOFT'05), Jersey city, New Jersey, USA, September 2005.

[23] P. Amay and B. Dion, Combining model-driven design with diverse formal verification, in ERTS 2006 - Embedded Real Time Software, Jan. 2006.

[24] TTTech, Company homepage, http://www.tttech.com.

Email Addresses

haf@informatik.uni-kiel.de

rvh@informatik.uni-kiel.de

joern.rennhack@airbus.com

jens.koch@airbus.com