Model-based design of CANopen systems

Multiple disciplines for mechatronic system design co-exist, which hinder the utilization of software-oriented modeling principles e.g. UML. Existing modern tools may be integrated into a working tool chain.

Author

Dr. Heikki Saha

TK Engineering
P.O. Box 810
FI-65101 Vaasa
Tel.: +358-50-588-6894

Link
www.tke.fi

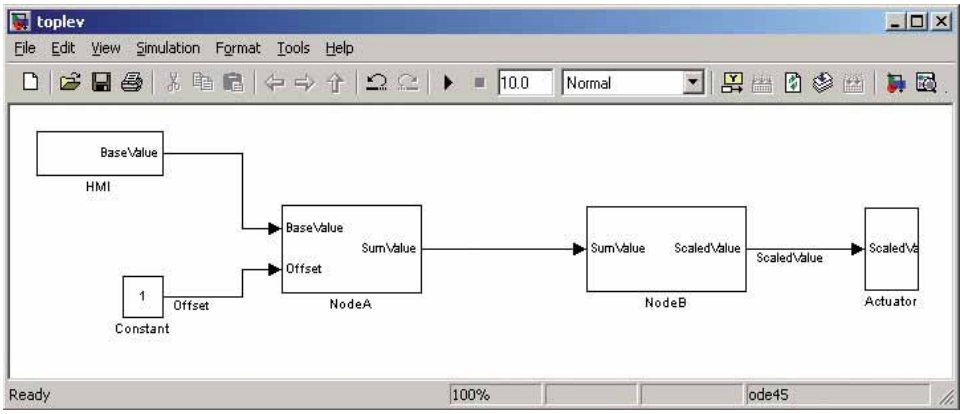

Figure 1: Example of a top-level system model consisting of two application-programmable nodes, Node A and Node B

Model-based design has become mainstream in the industry, but it has mostly been used for development of individual control functions or devices, not entire control systems. Current mechatronic systems are becoming more complex and simultaneously the requirements for quality, time-to-market, and costs have become higher. An increasing number of systems is distributed, but development is typically done device by device, without systematic coordination of system structures. Approaches to manage distributed systems with written documents have lead to inefficiency and inconsistent interfaces. Inconsistent interfaces have sometimes led to situations, where it was easier and faster for the designers to write a new software component instead of re-using an existing one. Another typical occurrence is that significant interface adjustments have to be performed during integration testing of a system. Based on such experiences, there is a demand for standardized and semantically well formed interfaces between multiple disciplines [16].

In typical mechatronic systems, multiple disciplines co-exist and none of them dominate. The multidisciplinary nature of design work makes it very difficult to utilize the modeling principles dedicated for software-oriented development, such as UML or SysML [1]. It has also been found that it is impossible to create a single tool, which is optimal for all disciplines. Instead, existing state-of-the art tools can be integrated into a well working tool chain.

The traditional way

In a typical distributed system, one function may be divided into several devices and one device may serve multiple functions. Node-centric development might be difficult because the functional distribution is not exactly known prior to development. Application-centric development and simulation provides limited efficiency because of limited testing capabilities [11]. Software-centric development without a thorough system level management will lead to serious interface inconsistencies. The old approach to managing communication interfaces is to embed communication descriptions into the application software [5]. Historically, this works with very small systems, where there is only one instance of each type of device. When devices exist more than once in a system, such an approach often leads to poor re-use of design artifacts or adoption of configuration management processes.

Model-based designs have become attractive because of the inefficiencies of the existing approaches. Though the requirement management in traditional software development has been document-centric, it has not been unusual that the requirements for the next version were collected from the source code of a ▷

previous version [18]. It has also been documented that model-based designs can reduce number defects and wasted efforts produced by current approaches.

A separate design of logical and physical structures causes challenges in managing the two parallel models and their connections without inconsistencies and still allowing incomplete models [1]. In addition, if a model-based conceptual design was used, models can be manually converted into code or control applications can be developed and tested separately, independent of each other. The main motivation for more systematic developments can be found in the assembly and service process, rather than in development, because of their higher significance [3]. Systematic configuration management enables solving serious problems e.g. during system assembly and service [3]. Systematic configuration management is required throughout the development process [18].

Existing modeling approaches

Increasing complexity of the systems requires increasing systematics during development [10]. Most defects found during the last phases of the traditional processes are caused by failures in the requirement acquisition in the early phase of the processes [10] [18]. The validation of specifications to models and model-to-code matching is easier with simulation models [9] and the use of automatic code generation with proven tools makes it possible to automate code verification and move the focus of reviews from code to models. Automatic code generation from simulation models improves the development of especially high-integrity systems [9], [10], [11]. The simulation model is actually an executable specification, from which certain documents can be generated [9], [10], [15], [18]. Higher integrity with lower effort can be achieved by validating the basic blocks and maximizing the re-use of them [15]. Conformance to corresponding standards helps to achieve required quality [15]. Simulation models can also document interfaces between structural blocks, improving consistency and enabling parallel and co-development, improving the overall efficiency [10], [12], [18].

It has been recognized that old processes produce old results [18]. New development approaches, such as a model-based design, improve the design. To achieve maximum improvements, new processes and tools are often needed. A new process with an existing, constrained design does not show benefits, but with new and more complex designs benefits can be found. A phase-by-phase approach is required to provide a learning curve. It is also important to be able to keep existing code compatible with the new code generated from models. Design re-use is one of the main things that improve productivity. The systematic management of both interfaces and behavior is mandatory in safety relevant system designs [7]. Instead of using model-based tools as a separate overlay for the existing processes and tools, automated interfaces need to be implemented between tools [18]. Connecting model-based tools with the existing legacy tools may require changes beyond built-in capabilities of the tools, increasing the effort required to maintain, develop and upgrade the tool chain.

Scope

The Simulink tool was used in the project because it is the de-facto modeling tool in research and industry ▷

and it has open interfaces. Furthermore, it solves most of the problems found in other modeling languages and approaches [1]. One of the most significant benefits is the support of dynamic simulations. Unlike e.g. executable UML, Simulink models can be used for modeling other disciplines than software. The models can be made very simple and based on behavior only. The physical structure can be included into the model by adjusting the hierarchy of the logical model. Later on, the models can be developed to cover improved dynamics too, if required.

Because of the increasing time-to-market and functional safety requirements in machinery automation applications, higher productivity and support for model verification and reuse of designs were significant reasons for using Simulink. Such features include e.g. linking to the requirement management, model analysis, support for continuous simulation during the design process, testing coverage analysis, and approved code generation capabilities [17]. The use of Simulink models enables efficient re-use of the models for various purposes.

The main reason for using IEC 61131-3 programming languages for the evaluation is that they are well standardized, widely used in the industry, and their use has continuously been spreading. Their use in especially safety critical implementations is increasing because some of the IEC 61131-3 languages, which are considered as limited variability languages (LVL), are recommended by functional safety standards [7]. A standardized XML based code import and export format has been published recently, improving systematic design processes further.

Basically the presented approach is technology independent. CANopen was selected as an example integration framework, because the CANopen standard family covers system management processes and information storage. It is well supported by numerous commercial tool chains, which can be seamlessly integrated. The management process fulfills the requirements set for design of safety relevant control systems [7]. It is also well defined how CANopen interfaces appear in IEC 61131-3 programmable devices [2]. A managed process is required to reach the functional safety targets [7]. There is also a wide selection of various type of off-the-shelf devices on the market, enabling efficient industrial manufacturing and maintenance. Especially device profiles help re-using common functions instead of developing them again and again. In addition to the design and communication services, CANopen offers extensive benefits in the assembly and service when compared to other integration frameworks.

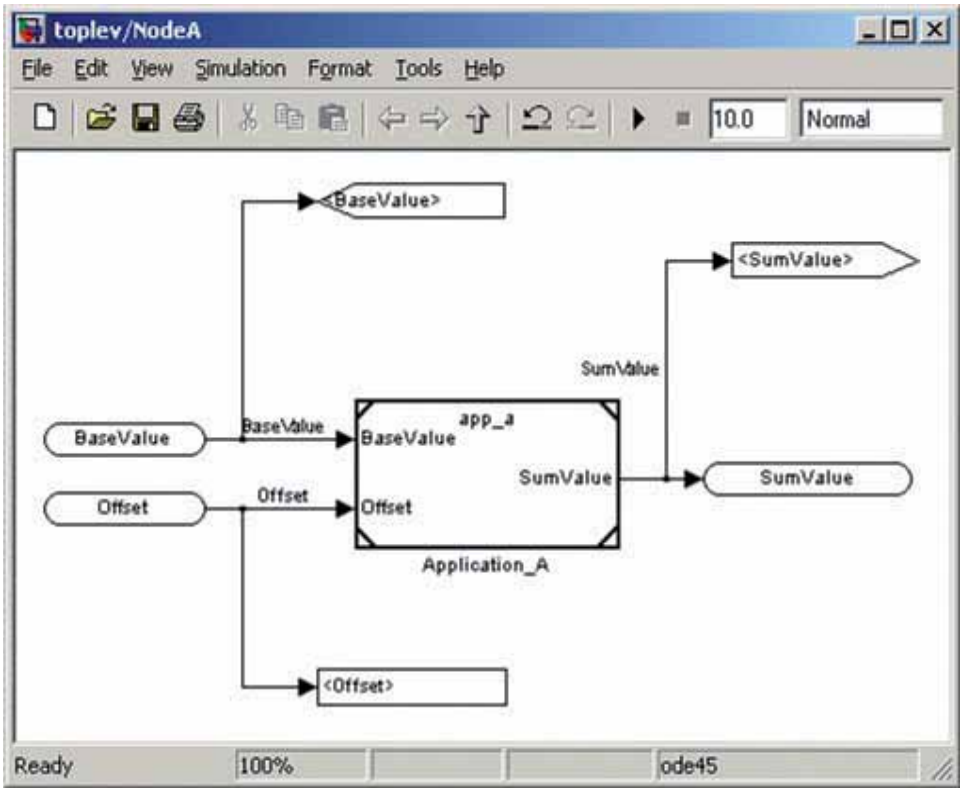

Figure 2: Example sub-model for Node A with linked application sub-model and integration interface descriptions

In this article, relevant CANopen issues are reviewed first to enable readers to understand the process consuming the presented communication description. Next, the basic modeling principles are shown. After presenting the modeling principles, the communication interface description in the model and exporting of both application interfaces and behavior are presented. Modeling details are not within the scope of this article.

CANopen issues relevant to modeling

The CANopen system management process defines the interface management through the system's life cycle from application interface description to spare part configuration download. The first task in the process is to define application software parameters and signal interfaces as one or more profile databases (CPD) [4]. Next, node interfaces defined as electronic datasheet (EDS) files can be composed of the defined profile databases. The EDS files are used as templates for device configuration files (DCF), which are system position specific and define the complete device configurations in a system. DCF files can be directly used in assembly and service as device configuration storage [19]. In addition to the DCF files, system design tools produce a communication description as a de-facto communication database format, which can be directly used in device or system analysis. A process with clearly distinguishable phases improves the resulting quality because a limited number of issues need to be covered in each step of the process [11].

Signals and parameters need to be handled differently [4] because of their different nature [14]. Signals are periodically updated and routed between network and applications through the process image [2], [4]. The process image contains dedicated object ranges for variables ▷

supporting both directions and the most common data types. The same information can be accessed as different data types. Signals are typically connected to global variables as absolute IEC addresses [2]. Signal declarations include metadata and connection information used for consumer side plausibility and validity monitoring. For parameters, metadata is used for both plausibility checking and access path declaration. All information relevant to the application development is automatically exported from the CANopen project to the software project of each application programmable device. Additionally, monitoring, troubleshooting, and rapid control prototyping (RCP) can be supported by the exported communication description. The completed CANopen project automatically serves the device configuration in assembly and service.

The process image located in the object dictionary serves also for communication between the functions or applications inside the same device [8]. It can also be shared by different field buses [6]. Software layers above the process image are not necessarily required with CANopen. The internal object access type can be defined as RWx to enable bidirectional access inside the producer device. The external access type should be defined as RWR to enable information distribution to the network. Access type RWW should always be used for incoming signals, which can be shared by multiple applications.

Parameters are stationary variables controlling the behavior of a software, their values are changed sporadically and in CANopen systems typically stored locally in each device [2], [4], [14]. Parameters of application programmable CANopen devices must always be located in a manufacturer specific area of the object dictionary. The only exception occurs if device profile compliant behavior is included. Then parameters must be located according to the corresponding device profile. It is recommended to organize application specific parameters as groups separated from the platform specific objects. Standards do not define the organization of parameter objects. Some different approaches to access parameters exist, e.g. linking global variables to objects or using access functions or function blocks.

System-level modeling in Simulink

A system model typically consists of models of a whole signal command chain, system or subsystem. The model may also contain sub-models describing behavior of e.g. hydraulics and mechanics, enabling multi-disciplinary design and simulation. The main benefit of model-based design is that errors are typically found earlier than in traditional approaches [10]. Models are executable specifications enabling continuous testing [12]. When whole command chains, systems or subsystems can be tested, more practical test scenarios can be used to reveal the problems more typically found with integration tests.

The multi-disciplinary system model can also be used for initial tuning of control behavior if dynamic behavior of e.g. hydraulics and mechanics is included. After finalizing the design and initial tuning the control behavior of each device can be automatically exported into executable programs to the final HW. Because of the ease of use and automated transformations, incremental modeling and development become ▷

efficient. A simple system model is shown in Figure 1.

Node model in a system contains CANopen mapping and a referenced application behavior submodel, as depicted in Figure 2. In early stages of development, parameter and signal descriptors are not required – they do not affect on behavior, but just tag the signal or parameter to be published. Signal names and data types are directly taken from the model to the descriptions. It is presented by literature, that simulation models are commonly used for documentation and communication of interface descriptions [10]. It is important to systematically define the interfaces, because the control functions communicate through the interfaces and any inconsistency can introduce more severe global consequences that an erroneous internal behavior.

It is mentioned in the literature, that configuration management is required for simulation models [18]. One approach to arrange a well documented and proven configuration management is to implement generic simulation models and publish the all configuration parameters. The proposed approach enables the utilization of configuration management features provided by system integration framework. If CANopen is used, various model configurations can be stored as profile databases, where parameter values can be imported to the new models. Potential conflicts can be detected and solved outside the model, in the corresponding design tools.

The main benefits of the referenced models are, that they are faster in simulation [13], they enable parallel development of sub-models and can directly be used from other top-level models [12], e.g. in rapid control prototyping (RCP). RCP can significantly speed up development,

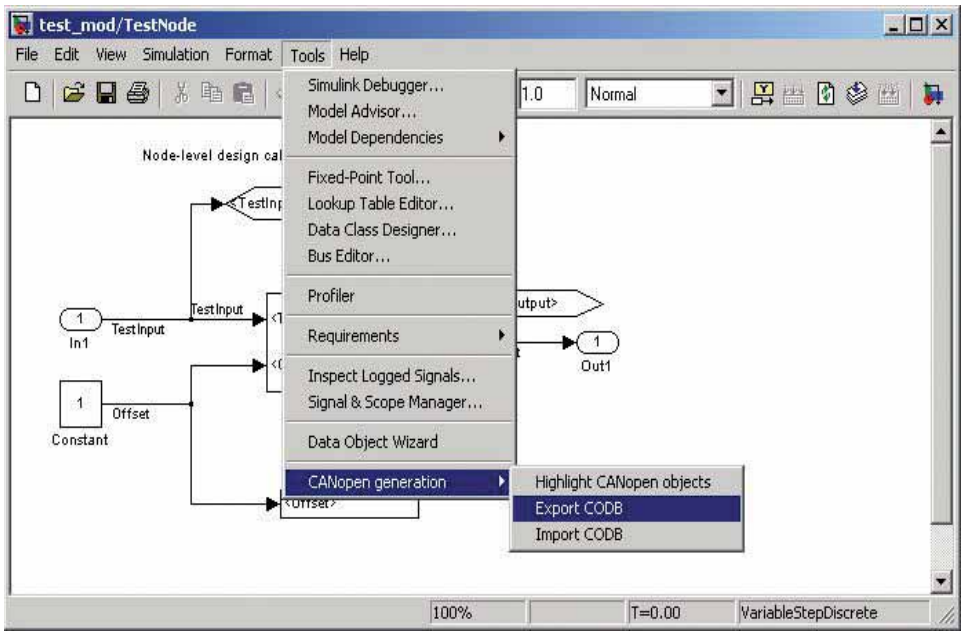

Figure 3: Example of exporting interface for Application A

because final processing performance, memory and I/O constraints do not apply [11]. Model referencing can as well be used as a reuse method of the application behavior in other models.

Preparing for export

Code generation from simulation models is a proven technology. The management of system level interfaces has not been included until now. After completing the application behavior, signals and parameters need to be defined. A dedicated blockset for such purposes has been developed. The blocks shown in Figure 2 are only markers, which are invisible to the code generation. The simulation model is independent of the integration framework and therefore only application interface descriptions are exported to framework specific tools. Such an approach enables the full utilization of the framework specific tool chain for integrating the application specific descriptions with hardware and software platform specific interface descriptions.

Signals and parameters behave differently and need to be managed accordingly [14]. Due to a thoroughly defined process image, signals may be automatically assigned into the object dictionary, but most devices have default PDO-mapping affecting the organization of the signals. Therefore it was the safest option at first to provide a manual override for automatic object assignment for the signals and parameters. The access type of signals is fixed by using direction specific blocks and the object type need not to be defined for the process image. Signals can also be introduced into e.g. device profile specific objects when standard behavior is developed. In this part of the process, compatibility with existing PLCs is as important as CANopen conformance.

Parameter management has even more deviations among different implementations. Therefore it should be possible to select the main attributes manually. The manual assignment enables parameter grouping into records and arrays, if grouping is required by applications. Access type and retain attributes are available only for parameters and their values are related to the parameter's purpose. If a parameter is intended to indicate a status, it needs to be read-only and not for retain. If a parameter's purpose is to adapt the behavior of a function, read-write access and retain are needed. Some parameters, such as output forces, require read-write access. Retain storage should not be supported, because forces should be cleared during restart for safety reasons.

Automatics can be implemented later e.g. by using target file describing object assignment rules specific to a target hardware. Development of interface standards and exchange formats will help the further development [2]. During the time of writing there are too much variations – especially in the management of parameter objects – to be covered by automatic assignment without potential need for further editing.

Minimum, maximum, and default values can be assigned for each object. They are important to be defined, because they can be efficiently re-used during further steps of the process. Those values can be given either as plain values or as variables in the Matlab workspace. Such an approach enables sharing the same metadata with ▷

application function blocks as constants linked to the same variables, but may add to the complexity of the model [14]. To speed up the modeling, value fields can be left empty, when default values are automatically used. Minimum and maximum possible values according to the object's data type are used as minimum and maximum values by default. If a default value is not defined, zero is used.

Generating exports

The generated application behavior needs to be isolated in a separate sub-model. Source code cannot be generated directly from the root of the referenced model. The structure of the generated code strongly depends on the internal structure of the source model [12]. The IEC 61131-3 code generation results in a single function block, where the behavior of the selected block is included. Depending on the model structure, other functions and function blocks may also be generated.

A completely fixed interface is mentioned in a case example presenting the application development improved by using fully automated code generation [11]. The more generic approach expects the management of the interfaces from the model [10] [12]. However, only application specific interfaces can be managed in the model and both hardware and software platform specific interfaces need to be managed according to the management process of selected integration framework. Applications can be developed as separate models and mapped onto the same physical node as part of the system design process. The level of modularity can be selected according to the application field. The configuration management [12] of the applications in the presented approach is supported by published application interface descriptions. The configuration management of the target system is done in a CANopen process supporting it better on the system level [11]. Calling of the application interface export of Application A is presented in Figure 3.

The resulting application parameter and signal object descriptions are shown in Figure 4. The file format in the example is a CANopen profile database (CPD) because CANopen was selected as an example system integration framework. Application interface descriptions are combined with descriptions of other optional applications, which will be integrated into the same device and the communication interface of the target device [4]. The resulting EDS-file can be used in system design as a template defining the communication capabilities of the device. System structure specific communication parameters are assigned during the system design process [2], [4].

Software integration

The first requirement is that all tools must be compatible with each other [11]. Based on experience, using standard interfaces is the easiest method to achieve a sufficient level of compatibility. Second, thoroughly defined interfaces are needed in co-development projects to get them working completely [12]. It cannot be assumed that all development is performed within a single company or department and with a uniform methodology. Third, outputs must integrate manually written, existing codes to enable either a smooth transition into model-based development or a flexible use of automatically generated and manually written code [11]. Fourth, although CANopen is currently the best integration framework in machin-

```
#-------------------------------------------------------------------------------
# Database export from model: test_mod/TestNode
# Created:  03.10.2014 11:39:42
# Integrated communication profiles
#
# Integrated device- or application profiles
#
# This CODB is written according to
# CiA-306, Part 2, Version 0.0.4 or higher
#
# This CODB is applicable to
# CANchkEDS version 2.2.1 or higher
#-------------------------------------------------------------------------------

# Parameter objects
2100::Offset:conditional::VAR:m:UNSIGNED8:m:rw:m::n::n:0:m:255:m:0:m:n:m
# Signal objects
A0C0::ObjA0C0:conditional::ARRAY:m::n::n:2:m:2:m::n::n::n::n
A0C0:00:MaxSubA0C0:conditional::VAR:m:UNSIGNED8:m:ro:m::n::n:0:m:254:m:1:m:n:m
A0C0:01:SumValue:conditional::VAR:m:INTEGER16:m:rwr:m::n::n:-32768:m:32767:m:0:m:y:m
A540::ObjA540:conditional::ARRAY:m::n::n:2:m:2:m::n::n::n::n
A540:00:MaxSubA540:conditional::VAR:m:UNSIGNED8:m:ro:m::n::n:0:m:254:m:1:m:n:m
A540:01:BaseValue:conditional::VAR:m:INTEGER16:m:rww:m::n::n:-32768:m:32767:m:0:m:y:m
```

Figure 4: Interface descriptions for parameters and signals of Application A as a CANopen profile database

ery applications, upgrade paths and additional supported integration frameworks should also be possible.

A generic approach does not support predefined signaling abstraction used in some implementations [11]. Instead, application specific abstractions need to be generated from the model and developed further in the CANopen process, where physical platform specific and communication specific details can be integrated most efficiently into a complete description of a device's communication interface. That includes necessary information from the rest of the system [4] [20]. Finally the communication abstraction is imported as an IEC code into a development tool. Manual coding is required only for connecting the exported application behavior into communication and I/O abstraction layers. The approach follows a standardized process enabling integration of commonly used tools, which is also recommended in the relevant literature [18]. Relying on a standardized process enables a simple adaptation natively supported by the tools and heavy tool customizations are avoided, which confirms the findings already presented in the relevant literature [18].

The remaining manual integration work is minimal, mainly consisting of connecting application signals and parameters to the communication abstraction layer. Moreover, signal and parameter metadata – minimum, maximum, default values, and signal validity – if used by application behavior, also need to be connected manually to the relevant application function blocks. Fixed connections are not performed, because such information is not necessarily required for all signals and parameters. Including complete metadata for all signals and parameters with plausibility checking may require too much memory and processing power. An automatic connection would also violate the requirements of flexible mixing of manually written and automatically generated code [18].

Discussion

An approach to including public interface descriptions into the same model with system behavior divided into multiple application has been presented. Such an approach enables an efficient system level interface management, which serves the design process by enabling the export of application specific signal and parameter descriptions. Furthermore, the be- ▷

References

1] Laakso M., Distributed System Design Flow: Fieldbus Modeling, Master's thesis, TUT, 2008, 78 p.

[2] Saha H., Improving development efficiency and quality of distributed IEC 61131-3 applications with CANopen system design, Proceedings of 13th iCC, CiA, 2012, pp. 10-15 – 10-21

[3] Saha H., Benefits of intelligent sensors and actuators throughout the systems life cycle, The Twelfth Scandinavian International Conference on Fluid Power, May 18-20, 2011, Tampere, Finland, ISBN-978-952-15-2517-9, pp. 169 – 181

[4] Saha H., Wikman M., Nylund P., CANopen network design and IEC 61131-3 software design, CAN-Newsletter 3/2009, CiA, 2009, pp. 52 – 58

[5] Tisserant E., Bessard L., Trelat G., Automated CANopen PDO Mapping of IEC 61131-3 Directly Represented Variables, Proceedings of 12th iCC, CiA, 2008, pp. 06-08 – 06-13

[6] Rostan M., Hoppe G., Generic Fieldbus Application Program Interface for Windows, Proceedings of the 7th iCC, CiA, 2000, 7 p.

[7] Safety of machinery. Functional safety of safety-related electrical, electronic and programmable electronic control systems, EN 62061, 198 p.

[8] Additional application layer functions, Part 4: Network variables and process image, CiA-302-4, CiA

[9] Conrad M., Verification and Validation According to ISO 26262: A Workflow to Facilitate the Development of High-Integrity Software, SAE,

[10] Murphy B., Wakefield A., Friedman J., Best Practices for Verificzation, Validation, and Test in Model-Based Design, SAE, 2008-01-1469

[11] Thate J. M., Kendrick L. E:, Nadarajah S., Caterpillar Automatic Code Generation, SAE World Congress, 2004-01-0894

[12] Anthony M., Friedman J., Model-Based Design for Large Safety-Critical Systems: A Discussion Regarding Model Architecture

[13] Nadarajah S., Large Scale Modeling and Simulation of Propulsion Systems, SAE, 2007-01-1645

[14] Anthony M., Behr M., Model-Based Design for Large High Integrity Systems: A Discussion on Data Modeling and Management, AAS 10-023

[15] Anthony M., Behr M., Jardin M., Ruff R., Model-Based Design for Large High-Integrity Systems: A Discussion on Verification and Validation

[16] Markkula M., Rokala M., Palonen T., Alarotu V., Helminen M., Koskinen K. T., Utilization of the Hydraulic Engineering Design Information for Semi-Automatic Simulation Model Generation, Proceedings of The 12th Scandinavian International Conference on Fluid Power, 2011, ISBN 978-952-15-2522-3

[17] Erkkinen T., Conrad M., Safety-Critical Development Using Automatic Production Code Generation, SAE 207-01-1493

[18] Dillaber E., Kendrick L., Jin W., Reddy V., Pragmatic Strategies for Adopting Model-Based Design for Embedded Applications, SAE 2010-01-0935

[19] Saha H., Accelerated transfers of CANopen projects into assembly and service, CAN Newsletter 4/2012, CiA, 2012, pp. 17-20

[20] Saha H., Experimental CANopen EEC management, CAN Newsletter 1/2013, CiA, 2013, pp. 12-18

havior of each application can be generated from the same model. Application programs with communication abstraction layers can be developed simply by combining interface descriptions and application code modules. The uniform and automated management of system integration interfaces improves the development process and enables a model-based design of entire systems instead of a design of individual applications. In addition to behavioral errors, information interchange inconsistencies can be found earlier, which reduces failure costs. Moreover, higher system-wide safety integrity can be reached through the presented approach more comprehensively than before.

The use of proven tools and standardized file formats enables an efficient re-use of design information throughout the design process. Small changes during the process are inherently made directly into the CANopen project – DCF-files. Changes can be updated backwards to the corresponding EDS-file easily with existing tools. Updated EDS-files enable node re-use of the devices with the most recent changes [4]. Application interfaces defined as CPD files can be updated by extracting the defined part of an EDS-file into the corresponding CPD, which enables application level re-use. The changes can be read back from CPD into a simulation model. The signal or parameter name and data type introduce a problem, because in export they are taken from the model. However, if additional changes are allowed, incomplete back annotations from CPD into the simulation model can be performed. The problem is not significant, because the model should be the master version for both behavior and interfaces anyway [10].

Model-based development and model referencing enables the direct re-use of application behavior as referenced models for other purposes, such as RCP and education simulators. Source code generated from the model can also be re-used indirectly in code modules. Code generation supports several programming and hardware description languages, which also enable the optimization of partitioning between hardware and software implementations.

Although systematic, system-wide signal and parameter management as an integral part of model-based designs has been implemented, further development is needed. From a process efficiency point of view, it is most important to develop the automatic assignment of parameter object indexes. Such a development should be tightly coupled with the integration framework specific standardization work. Such improvements, like an automatic connection of applications into communication and plausibility checking of signals using partial value range, will be implemented in the future. Including I/O abstractions is also an interesting topic for the future. It is also possible to add support for other system integration frameworks than CANopen. Based on current knowledge, a fully automatic software development requires such tight constraints for hardware and software components that such a development is not important. ◀