Modelling of embedded CAN applications

Dipl.-Ing. Jürgen Pisarz
Dr.-Ing. Martin Schneider

c/o *port* GmbH
Droysziger Weg 56
D-06188 Hohenthurm area Leipzig/Halle
Phone +49-34602 33 279
Fax +49-34602 33 280
E-Mail service@port.de

Abstract

The paper gives an overview over the object-oriented methodologies including related workbenches, which were developed especially for modelling of distributed reactive systems. The applicability of this tools will be discussed with a concrete motion control application based on CAN.

Also some aspects of hardware/software co-design of such kind of systems regarding to the quality and efficency of the development process will be part of the presentation.

1. Application strategy for distributed real-time control systems

The dramatically increased complexity of distributed real-time control systems and software is caused by the requirements *Timeliness*, *Dynamical internal structure, Reactiveness*, *Concurrency* and *Distrubution*.

Each of these aspects adds fundamental difficulties to the development process - that means that the development process itself is a key issue in successfully realising complex real-time systems.

Following [Tör95] the design of distributed real-time control systems requires methodologies, models and tools that facilitate systems engineering supporting the design phase to determine:

- an appropriate level of application decentralisation;
- a suitable way of organising the distributed application.

With refer to [Ger95], [Sel92], [Stü95] and many others mainly modelling and simulation can help in evaluating different design choices to manage the complexity of such kind of systems, and it is important to obtain informations about a desired system before constructing it, mostly in time and money critical market sections.

2. Development Strategies

Traditional tools are mainly used for syntactical checks of design and code, but usually not for validation of resulting behaviour and performance of a system implementation before the coding phase. They have methodological gaps in the development process and a relatively late response to system properties.

To determine the correct functionality of embedded CAN applications different bus analyser and other tools are used.

Therefore methods and tools are needed to support a better validation of system properties like functionality, behaviour and performance at the beginning of the lifecycle on understanding of the problem and on fixing the desired properties by getting feedback from what is defined.

Rapid prototyping is already used for better understanding of a problem. But unfortunately the results of prototyping are not consequently used in final implementations.

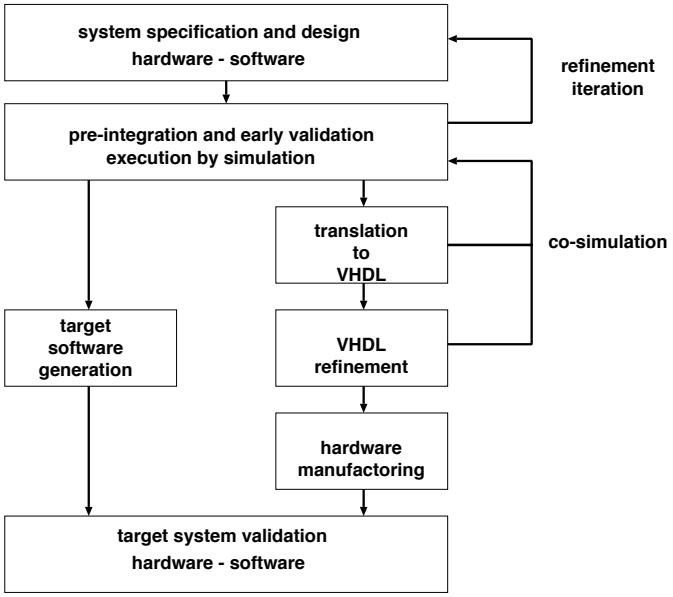

Figure 1: Systems development approach [Ger95]

The alternative lifecycle in future development of systems and software, shown in Fig. 1, is driven by formalisation of the problem and use of sophisticated methods and tools for early validation of a prototype from which the final implementation can be derived. That means to use a problem-oriented language, method or tool for expressing the problem and not a textual description [Ger95].

Also reuse is still an issue and object-oriented methods and tools have the capabilities to provide powerful mechanisms for reuse of concepts rather than only code.

The development process to generate systems that meet the requirements can be viewed as an iterative search through a tree of design alternatives. Results of the development progress can be measured in terms of the achieved part of the total specified functionality.

Based on this iterative view of development, the most of object-oriented methodologies do not part the development process into sequential phases with distinct functions performed in each. Instead of the strongly phased *Waterfall*-model, the development process breaks down in terms of activities, according to their respective objectives.

The objective of *Analysis* is to gain an understanding of the problem at the current level of refinement including the analysis and refinement of requirements and the modelling of the environment in which the problem is defined as well. The inputs to analysis are the system requirements, system models from previous levels of refinement, and domain expertise. The formal outputs are analysis models which can form a basis for the design model, and perhaps a more refined requirement specification.

The objective of *Design and Implementation* is to create a design model that provides a solution to the requirements based on the understanding captured in analysis.

In other words, the choosen analysis model and the design model are simply two parts of a system model with typical commonalities.

The objective of *Execution and Verification* is to verify a model by executing it and comparing its behaviour against a set of prepared test cases whether the model meets its requirements.

Several features distinguish this process model from previous ones:

1. It takes advantage of the fact that analysis models are executable to gain early insight into the problem;
2. The design starts in an early phase of the process to explore possible alternatives of the most difficult parts - the ability to execute the combined analysis and design specification is heavily exploited;
3. The analysis model can form a starting point for the design model;
4. Because of the continuity of concepts no distinction between design and implementation is made (all executing activities use the same abstractions that have been used to create the models).

Important for the model is, that tool support is absolutly necessary to execute or simulate models, and to manage the iterative development process.

3. Modelling alternatives

The known object-oriented toolsets for simulation of real-time systems provide complete system architectures organised in inheritance hierarchies allowing abstraction and reuse at a much higher level than it is possible with programming languages.

To describe an application concepts for capturing system properties in terms of structure and behaviour are offered.

The *Structure* (or architecture) of a system is the definition of the set of its component parts and the set of communication and containment relationships between those parts.

While structure deals mainly with the static aspects of a system, *Behaviour* captures its dynamics. This encompasses all actions within the system whether they are generated internally or in response to external stimuli.

The toolset *ObjecTime* is an implementation of the Real-Time Object-Oriented Modelling *ROOM*-methodology, which was designed for modelling of complex reactive distributed systems in the telecommunication area.

Structure description is given in *ROOM* with *Actor-Hierarchies* and *Protocol-Classes* for communication purposes between Actors shown in Fig. 2.

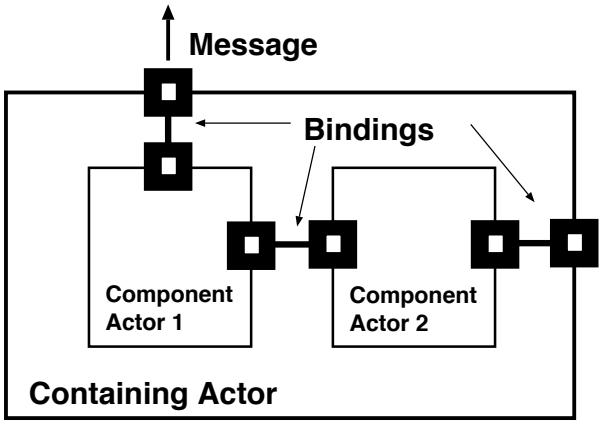

Figure 2: Actor-Hierarchies

Behaviour capturing is provided by *Hierarchical State Machines* for each Actor.

The actor paradigma is strongly focused on distributed event driven systems.

For high level analysis activities *Message Sequence Charts* (Fig. 3) provided by *ObjecTime* and *SDT* are useful to create scenarios of interaction between *Actors.*

Also *Actors* as independent active logical machines with a structure and an optional behaviour are useful for modelling CAN networks.

Unfortunately the explicit protocol based communication between ports - the basic paradigm of *ROOM* - has some disadvantages for modelling bus structures, broad- or multicast and synchronisation which are only surmountable by additional modelling effort using dynamic structure reconfiguration.

An alternative object-oriented workbench is *ControlShell*. This approach addresses the fundamental issue of how to best merge event driven reaction with cyclic feedback control.

Dataflow-systems are organised by *components* to describe cyclic control. *Finite state machines (FSM)* are used for expressing the startegic part of a system. Both are supported by a database.

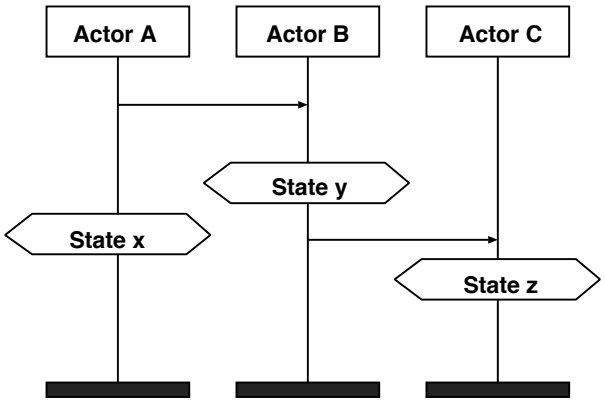

Figure 3: Message Sequence Charts MSC

Developed for motion control and robotics with cyclic and reactive capabilities and the functionality of the network data delivery service for distribution the *ControlShell* approach is suitable to model CAN*open* applications too.

A remarkable restriction for modelling CAN in this environment is the strictly periodic control of the data flow system. This disadvantage may be overcome for synchronised CAN*open* applications using the SYNC mechanism and with additional time synchronisation functions for CAN proposed by [Tör95] and advised by [Neu95] to met the requirements of the real-time market.

4. Application strategy for complex real-time control

A characteristic application area for distributed real-time control systems is advanced motion control for mechatronic systems.

Typical for such systems are heterogenous application architectures combining centralised and decentralised application components both in hard- and software with an increased number of devices coupled by CAN or other fieldbusses.

The choice of CAN for motion control is caused by the cost effectiveness and the short worst case bus access latency, which gives CAN the potential for high performance and less missed deadlines in distributed control systems. Unfortunately the design of the message priority assignment algorithm is crucial to guarantee message latencies [Ba95].

Therefore the aim of designing CAN based real-time control systems is to assign each message to a priority in a way, that each message not exeed the maximum allowed delay or deadline in that system.

To manage the product development process and to reduce the development risks it is useful to split the process in two phases

1. Rapid prototyping and
2. Product optimisation

Phase 1 aimes to save the time to market for new technologies and phase 2 provides the cost effectiveness for mass products.

The focus in phase 1 is a principal solution for control of electromechanical systems. Developing tasks are:

- specification of the distribution of functionality;
- evaluation of communication requirements;
- implementation and validation of control algorithms.

Typical tools to solve this type of application tasks are for example VMEbus and CAN components, real-time operating systems (RTOS), motion control frameworks and simulation tools, discussed above.

The aim for phase 2 is a volume dependent optimised product. The implementation of the validated prototype needs for example:

- decisions to target microcontrollers;
- development of the final hardware;
- development of firmware;
- implementation and test of the integrated system.

The used toolset for phase 2 is dependent of the targeted volumes.

For very high volumes complete hardware/software codesign methodologies based on the system description language SDL or others for development as a source for automatic generated code C/C++ or VHDL is useful. The main advantage of this strategy depicted in Fig. 1 is given with the opportunity to make design decisions for realising functionality in hard- or software relativly late in project.

For lower volumes the development tasks are concentrated on systems integration of components partly used in phase 1 or developed based on design decisions in this phase.

5. Modelling CAN integrated control

ControlShell is an open expandable object-oriented application framework for real-time control.

Objects in *ControlShell* are the FSM's and the transitions in FSM's coupled with data-flow objects, called components.

The run-time environment shown in Fig. 4 directly supports the two different programming paradigms characterised by:

1. The data-flow in a system - fundamentally synchronous and executed in a sequential manner, and
2. The events in a system - fundamentally asynchronous with non-sequential execution.

The application is built from an object hierarchy.

Components are organised in execution lists, sample habitats, and configurations to serve dynamic internal structures.
All parts of an sample habitat are listed in its execution list. The list contains activated and deactivated parts due to the active configuration. The activated parts will executed periodically with a defined sample rate. Each sample habitat has its own rate.
There is no need to have only one sample habitat in the application. The only restriction for the variety of the sample rate is their common source, normally a hardware clock.

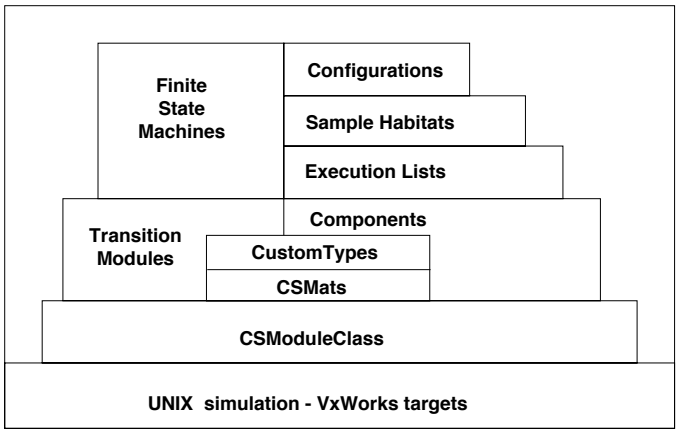

Figure 4: Run-Time Hierarchy

Transition modules are bound with the real-time state engine to form state machines.

Data expressions consist of objects of the CSMath class which contains types from simple integer up to complete matrices and the necessary operations up to matrix solving or of custom types build with CSMath types.

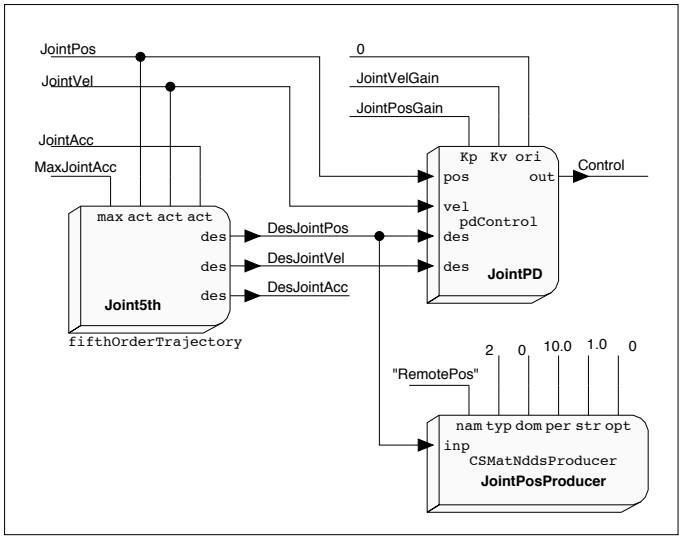

Figure 5: data flow diagram
simple controller including NDDS producer

The integrated network data delivery service *NDDS* provides a transparent network connectivity based on the so-called *Subscription Communication* with similar working principles to CAN.

This communication system builds on the producer and consumer model. Producer register information unaware of prospective consumer. It is possible to have more than one producer of the same data. Consumer subscribe to update the information they require without concern for who producing them [Par94]. In difference a CAN producer needs at least one consumer to get a valid acknowledge.

A producer component is characterised by a set of parameters:

strength	priority of producers of the same data
persistense	duration of validity of strength
option	determines what part (data, size, name, units,...) of data has to be sent
name	data name in the NDDS network

A consumer requests the distrubuted data and delivers it to the data flow it belongs to. In the *ControlShell* environment it is polling the NDDS network periodically. The highest possible rate would be the sample rate of its sample habitat.
There are some parameters too to determine the proper work of the consumer:

deadline	specifies the maximum time between data updates, old data will be marked as such
minSeparation	specifies the minimum time intervall between updates to limit network traffic
type	determines the working type: immediate or polled
name	data name in the NDDS network

Wait and deadline are parameters to regulate tradeoff beetween returning date as soon as available or waiting for better data.

The service uses UDP as a means of communication. Data is encoded with XDR to allow communications between computers with different data representations. The service is available for different UNIX systems, Windows NT and VxWorks.

StethoScope is another tools to provide an effective developing. It is used for graphical real time monitoring of every available and needed data both in simulation and in the run time environement. Application fields are debugging, (motion) tuning or visualisation.

There is also an interface to MatLab/SimuLink available. It allows modelling of complex mechatronic systems.

6. Modelling practice

For modelling CAN integrated real-time control the *ControlShell* framework is suitable without any CAN specific module like shown in Tab. 1 which compares CAN*open* with *ControlShell* elements.

ISO/OSI	CAN*open*	*ControlShell*
	object dictionary	CSMath
	function blocks	components
	finite state machines	FSM
	communication profile	
7	CAL	NDDS
0-2	CAN bus	TCP/IP, Ethernet

Table 1: Comparison of CAN*open* and *ControlShell*

NDDS-coupled *CS* systems/devices are useful to analyse appropriate levels of application decentralisation constisting of a master (for example on VMEbus) and different numbers of nodes.

This approach offers also some capabilities for modelling message priorities by parameterising the producer and consumer components of *NDDS* as shown in the simple example for a remote controller in Fig. 5 and 6.

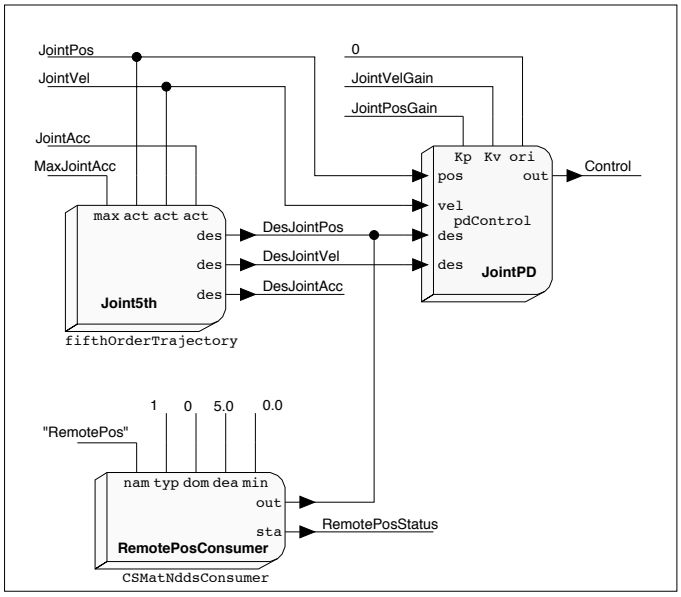

Figure 6: data flow diagram
simple controller including NDDS consumer

The organisation of the distributed application and the evaluation of communication requirements will be done by:

1. CAN bus models in the way of [Stü95] for communication assessments and
2. CAN*open* configurations on distributed development systems using a CAN driver on each system and *NDDS* as monitoring facility in parallel for a complete verification of an application.

Additionally a CAN simulation daemon for the development system for behaviour analysis of an application with different *ControlShell*'s is possible.

For case 1 multiplexer and demultiplexer components and a burst silence generator component must be defined using the component editor.

The needs for case 2 are CAN hardware and driver for VxWorks systems available on the market and CAN producer and consumer components to connect the network to the other application components of *ControlShell*.

These components under development are structered similar to *NDDS* components with refer to CAN*open*.
At this time they get a set of parameters listed below:

name	data name in the network
identifier	specifies the indentifier and priority of data
deadline	specifies the maximum time intervall between updates
minSeparation	specifies the minimum time intervall between updates to limit network traffic
sync	determines the sync type: none, sample rate, external, other
type	determines which data parts have to be sent

This way provides an easy portation of applications between several hardware or fieldbus architectures by only changing a small number of similar components.

7. Summary and Conclusion

With *ControlShell* models and tools that facilitate systems engineering for the design of distributed CAN integrated real-time control systems are available.

Because the object-orientation of the approach higher productivity and increased software quality is reachable comparing to more traditional techniques.

Different runable models can be created for different development purposes, and with the CAN components the system development process is supported up to turn key solutions running under VxWorks.

Applying the fast serial link FSL technology for complex motion control [Pi96] also the hardware dependencies are solved.

By using the producer/consumer model and the appropriate components there is an easy way to change and adapt the communication medium in an application if needed.

References

[Ba95] M. D. Baba and E. T. Power: Scheduling Performance in Distributed Real-Time Control Systems. Proceedings of the 2nd international CAN Conference, Published by CAN in Automation 1995

[Neu95] Dr. K.-Th. Neumann at al.: TOUCAN: A new CAN Communication Module for Embedded Microcontroller. Proceedings of the 2nd international CAN Conference, Published by CAN in Automation 1995

[Pi95] J. Pisarz: Frameworks for designing motion control applications. Proceedings of the PCIM'96, International Intelligent Motion Conference, Published by ZM Communications GmbH 1996

[Pi96] J. Pisarz: Digital signal processing and high speed communication for embedded motion control. proposed for the Proceedings of the ECC '96, Embedded computing conference Paris.

[Stü95] M. Stümpfle, J. Charzinski: Simulation of Heterogeneous CAN-Systems. Proceedings of the 2nd international CAN Conference, Published by CAN in Automation 1995

[Tör95] M. Törngren: A Perspective to the Design of Distributed Real-Time Control Applications based on CAN. Proceedings of the 2nd international CAN Conference, Published by CAN in Automation 1995

[Ger95] R. Gerlich, C. Jorgensen: An alternative Lifecycle Based on Problem-Oriented Methods and Strategies. ESTEC Symposium on "On-Board Real-Time Software", Noordwijk 1995

[Par94] G. Pardo-Castellote and S.A. Schneider: The Network Data Delivery Service: A Real-Time Data Connectivity System. Conference on Robotics and Automation, IEEE 1994

[Sel92] B. Selic, G. Gullekson, J. McGee, I. Engelberg: ROOM Real-Time Object-Oriented modelling. CASE'92 Fifth International Workshop on Computer-Aided Software Engineering, Montreal 1992

[Sn95] S.A. Schneider, V. Chen, G. Pardo-Castellote: The ControlShell Component-Based Real-Time Programming System. Conference on Robotics and Automation, IEEE 1995