A Model-Based Approach to the Development of Distributed Control Systems*

Manuel Bernardo Barbosa (mbb@di.uminho.pt)
João Miguel Fernandes (jmf@di.uminho.pt)
Departamento de Informática, Escola de Engenharia
Universidade do Minho, Braga, Portugal

Abstract

Distributed Control Systems (DCS) are a class of application with specific characteristics. This type of system is used in industrial environments to control manufacturing processes. Usually they comprise a controller, a fieldbus network, and a set of Of-The-Shelf (OTS) components, interfacing process signals with real-time QoS requirements. In this paper we present a Model Driven Development (MDD) method that targets this category of systems. This method focuses on the critical stages of DCS development. Namely, the specification of system requirements, the choice of OTS modules and fieldbus system, and the validation of the design using real-time analysis tools. This MDD method uses the Unified modelling Language (UML) as support notation, including the extensions defined in the UML Profile for Schedulability, Performance and Time Specification.

1 Introduction

The use of MDD methods greatly improves the productivity and reliability of software systems. There is no reason why this type of methodology should not bring similar benefits to the development of software for embedded systems. Previous experiences show that this is indeed the case.

These issues are particularly relevant in the development of embedded systems with real-time QoS requirements. In this paper we present a MDD method that targets DCS, a type of real-time system common in industrial environments.

A DCS is composed of several intelligent modules connected to sensors and actuators that provide an interface to the process being controlled. These mod-

*Work partially supported by Fundação para a Ciência e a Tecnologia (FCT) and Fundo Europeu de Desenvolvimento Regional (FEDER) under project “METHODES: Methodologies and Tools for Developing Complex Real-Time Embedded Systems” (POSI/37334/CHS/2001).

ules cooperate by communicating through a network system, typically a fieldbus. They may be custom-built for a particular application but, in most cases, standard OTS components are used. The overall operation of the system is managed by one or more controller devices, which may also operate as gateways to other information systems.

The need for this type of methodology became apparent in our interaction with an industrial partner that develops DCS. Traditional software development MDD methods and real-time system development methods are not able to solve all the problems raised by the peculiarities of this type of system.

Our MDD method addresses specific aspects of the development of DCS: the identification of process output/input signals, i.e. system input/output signals; the selection of the OTS components and fieldbus system; the assignment of input/output signals to OTS components; the association of input/output signal events with network message transfers; and the usage of real-time analysis tools.

It uses the Unified Modelling Language (UML) and it is based on the UML Profile for Schedulability, Performance and Time Specification [7].

2 Background

2.1 Real-Time System Modelling Using UML

The central difference between modelling a software application and modelling a real-time system is that it is not possible to model the latter unless quantitative data can be depicted in the model. In fact, for most software applications, it suffices to model functional requirements. For real-time systems, it is also necessary to model real-time QoS characteristics consisting of time-related quantitative information. Even though UML is particularly well suited to develop real-time software [9], a careful adaptation of UML is required due to the particular characteristics of this type of software.

The usage of UML in the embedded field dates from 1998 [1]. In his book, Douglass introduces techniques for developing embedded real-time systems with UML. Another important line of research is the UML-RT proposed in [8] [11]. UML-RT is strongy based on ROOM [10] and proposes an approach based on collaboration diagrams. The adoption of UML-RT to model embedded real-time systems in the telecommunication domain is discussed in [5]. Other experiences, such as that in [4], show that UML is suitable to provide notational support in tool-chains for real-time embedded system design.

Recently, another step forward has been taken. OMG defined the UML Profile for Schedulability, Performance, and Time Specification [7]. This profile was written to be applicable to a wide range of application areas. It does not attempt to define a complete set of real-time modelling concepts. This would be

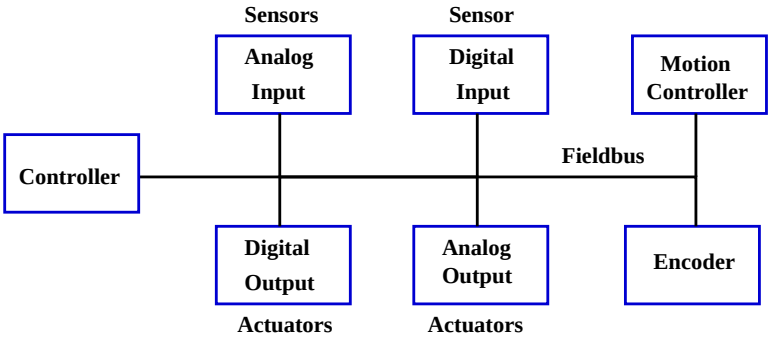

Figure 1: Structure of a generic DCS

virtually impossible given the variety of systems that fit this category, the many design styles that are used and the different modelling approaches that can be used. For example, only the designer can determine what is an acceptable level of detail when modelling a particular system. The concepts that are defined in this profile are kept to a minimum and at a level of abstraction that allows designers to take advantage of the "power of UML".

This profile also takes into consideration a very important part of real-time system development: automatic system analysis. System analysis consists of producing a specialized view of the system that can be processed automatically for one of two purposes: calculating the values of parameters still missing in the model; or assessing a particular characteristic of the system depicted in the model.

2.2 Distributed Control Systems (DCS)

This work focuses on the development of DCS, a type of process control system that is common in the manufacturing industry. These systems are used to control manufacturing processes by collecting process data and acting on the process according to a specific control algorithm. Figure 1 shows the typical structure of this type of system.

Systems like this present a set of common characteristics that permit tailoring a development process to them:

Self-containment Restrictions imposed by the control functionality usually imply that the control loop operation must be isolated from external interaction. In most cases, the whole system will present to the exterior a very simplified interface to allow for the collection of operational data, and the setting of global operational parameters.

Real-time QoS requirements Invariably, the specifications for this type of system include real-time QoS requirements. These will apply to sensing

and acting on process signals and, consequently, to all the processing that must be carried out to acquire, process, generate and transfer the associated data.

Fieldbus communication system The geographical locations where data collection and actuation must be performed are spread around the manufacturing floor, separated by distances ranging from a few to several hundreds of meters. Fieldbus standards such as Profibus and CANopen have been developed specifically to address the needs of this type of control systems [2].

OTS embedded components The economic benefits of using OTS components compatible with a particular fieldbus standard are well known. They include reduced system development time and cost, minimal downtimes through the replacement of damaged or faulty components, etc. However, the use of OTS components introduces a new problem in real-time system design: the degree of confidence that may be placed on the overall system greatly depends on the confidence placed on the reliability of the components and on the degree of knowledge that exists regarding their internal operation.

Fault-tolerance QoS requirements Some kind of fault tolerance guarantee is usually implied in the system specification and, often, it will be stated explicitly. Our project has not yet reached the point where fault-tolerance issues are addressed and this work does not cover fault tolerance requirements.

3 Development Process Overview

The following technological or architectural levels can be distinguished in the development of a DCS:

1. Hardware/software partition and hardware design
2. Real-time operating systems
3. Real-time network and OTS device software
4. Controller device software
5. Application software

Clearly, from the system characterisation in Section 2.2, the technological solutions for the first three levels will be very much constrained by the system specification itself. This in fact means that the entire architecture of the system

will be a rather direct consequence of the system specification. At these levels, the designer will be left with choosing and customizing suitable commercial products: OTS components and fieldbus system.

Software development will, in most cases, be concentrated on the the upper levels. It will comprise solely the software running in controller devices and connecting the control system to the outside world e.g. user interfaces, database connections, gateway connections, etc.

A development process for this type of system does not have to be built from scratch. In fact we chose to base our work on the embedded system development process presented in [3]. However, it was necessary to customize this process to address the peculiarities of DCS development. We have defined six development stages that will be described in the following sections.

3.1 Black-box system requirements

A fundamental aspect of system analysis is a correct characterisation of the requirements for the system's interaction with the environment.

In the case of DCS three types of external entities may interact with the system:

- Human users – They observe and adjust system operational parameters. In this case, typical user interaction requirements, common to other areas of software development, are at play.

- Input and output process signals (or logical groups of signals e.g. machines) – This is the process interface itself. The system requirements include the real-time characteristics of the signals that the system must acquire from sensors and generate for actuators.

- Other systems – These may collect monitoring information and adjust operational parameters. In this case requirements may be of two types. If the peer system is a higher-level system, then the application is functioning as a gateway, and the requirements are as for human users. If the peer system is at the same level as the one being designed, it is possible that real-time requirements apply to this interaction as well.

For each input or output process signal, a minimum set of parameters must be identified: the physical nature of the signal, the timing characteristics of the interaction, and the operations that must be perform on that signal.

3.2 OTS device and fieldbus selection

One immediate consequence of using OTS components is that the selection of the commercial components and fieldbus system that will be used in the system becomes a fundamental stage in the design process.

The selection of this type of equipment for a particular application may be subject to constraints that are out of the scope of a software development process e.g. economic and strategic issues. It is clear that the choices made by the designer, at this level, also depend on restrictions imposed by the physical environment on which the system will operate. An extensive discussion of the aspects that must be considered in this design stage can be found in [2].

More relevant for this discussion is an assessment of how the parameters resulting from the system analysis described in the previous section affect this design stage. The identification of the required OTS components implies that these components are assigned to a set of input and/or output signals which they will be interfacing. The characteristics of these signals will be determinant in the selection process. The nature of the signal will of course require that a compatible module is selected e.g. if we want to sense an analog signal, we must use an analog input module. The real-time QoS requirements for sensing or actuating on the signal will function as minimum performance requirements for the OTS module. Whether or not the device will be capable of doing that in run-time can only be assessed by analysing the entire system, as will be discussed in Section 3.6.

3.3 OTS device configuration and real-time model

OTS components are themselves complex subsystems, usually designed by a third party. These components present some degree of configurability, which the system integrator must customize to ensure that the overall system will operate correctly.

The configuration of each component will be performed in an implementation-specific manner. However, the designer will have to derive the configuration data for all of the components from the requirements associated with the signals that each of them will be interfacing. This is done using an algorithm that treats all devices uniformly. Usually this type of algorithm is specific to a particular fieldbus standard. For example, for a particular type of network, the messages that a device will be transmitting or receiving are configuration parameters that apply to all devices.

We believe that, to a great degree, the derivation of the configuration data for OTS devices can be automated. Nevertheless, for this to be possible, all devices compatible with a particular fieldbus will have to be modeled consistently, at a suitably high level of abstraction, based on coherent configuration parameters.

Furthermore, a very important phase in the design of DCS is the real-time analysis of the entire system. This type of analysis requires all of the system's components to be described in sufficient detail, in order to determine their influence in the system's behaviour. A more detailed model of the system will allow for a more precise analysis and, consequently, for less pessimistic design choices.

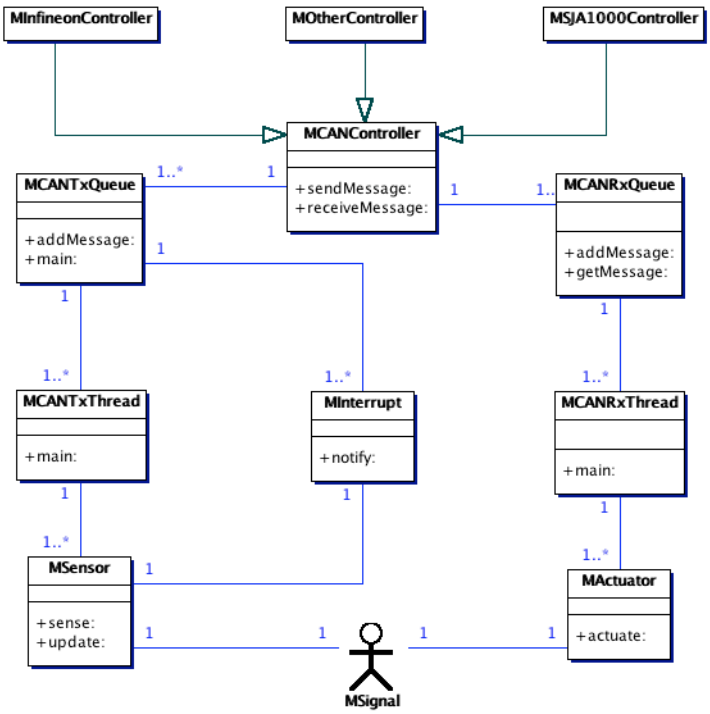

Figure 2: Model of an OTS CAN device

For the reasons explained in the previous paragraphs, our development method requires a generic model for the OTS components that are used in the DCS. In this paper we will be focusing on components that use the CAN network [6] as a communication link. This discussion will apply to any higher-layer protocol [2] using CAN, and it can also be easily adapted to other fieldbus technologies.

The generic model we use is shown in Figure 2. It reflects a view of a CAN-based OTS device where its internal behavior is structured according to the CAN messages the device is configured to receive and transmit.

This model is unlikely to be a truthful description of the internal structure of every component. However, it provides just enough detail to permit analysing the influence that each component will have on the overall operation of the DCS. The simplicity of this model also reduces to a minimum the number of performance parameters that must be known about a given implementation. Ideally, these parameters should be provided by the manufacturer, but they can also be measured by the system integrator, if required.

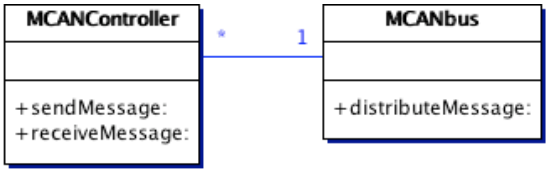

Figure 3: Model of the CAN network

3.4 Network configuration and real-time model

One of the determinant factors in the development of DCS is the influence of the communication infrastructure (fieldbus system) in the operation of the system.

Fieldbusses are communication protocols which operate over a shared medium, typically in bus configuration. A fieldbus influences the operation of the system due to the serialisation that it imposes on message exchanges: typically, only one device can send one message at a time. This implies that other devices wishing to transmit at the same time will be subject to delays in their operation. Conflicts are usually solved using a priority system.

This type of operation can be modeled as a resource sharing process that is common in concurrent systems. Conceptually, the network can be seen as a shared resource, residing on an independent node. When a device wishes to transmit a message, it must access the shared resource. The resource will be blocked until the message is transmitted. Bus access conflicts can be emulated through the priority system that is usually associated with this type of resource sharing mechanism.

The class diagram in Figure 3 shows the class that is used to represent the CAN bus, and its association with multiple CAN controller objects.

For each CAN bus that is used in the DCS (usually there is only one), an instance of the `MCANbus` class will be present in the model. Each CAN interface connected to the network (usually one per device) will be represented by an instance of one of the CAN controller classes shown in Figure 2. The network activity in the system will be represented in the model through message exchanges between these object instances, as will be seen in Section 4.

3.5 Controller development and real-time model

The application running on the controller(s) coordinating the operation of the DCS will be the only parts of the software composing the system that will actually be designed by the system developer.

Typically, a controller in this type of environment is itself a complex embedded system. The description of a complete development process for this type of module is not the purpose of this paper. Suitable MDD methods exist [3], and can be used without change.

One additional point that must be considered is that the controller's influence on the global behavior of the system must also be taken into account. This means that, similarly to what was described in Section 3.3 for OTS components, a suitable model of the controller must be produced.

Typically, the resulting model will not be too different from the class structure shown in Figure 2. Controllers produce and consume messages like all the other nodes. The difference being that the data contained in these messages is usually generated and consumed by the controller itself. There will however be additional functionality associated with the controller's interaction with the outside world, which must also be included in the model, in order to accurately characterise the workload carried out by the corresponding node.

3.6 Global real-time analysis

Global real-time QoS requirements appear as end-to-end time constraints, that apply to complex sequences of interactions, between multiple object instances in the complete system model. These end-to-end time constraints put limits on the time that may elapse from the time instant at which a triggering event occurs, until the system's response to that event is completed. These limits may represent hard or soft deadlines for the system's response.

Real-time analysis is a fundamental stage in the design of a DCS with real-time QoS requirements, where we validate that the solution meets its requirements. In some cases this analysis may even include the generation of operational parameters that can only be calculated based on a global view of the problem e.g. the assignment of priorities to messages circulating on the network to guarantee response times.

Our model-driven approach must support this type of analysis. This means that the models must include all the information that is required to carry out the necessary validation. Moreover, it should be possible to show, within the model, relevant results obtained during the real-time analysis.

The real-time analysis is usually carried out by specialised tools, which present an application-specific interface. Nevertheless, it is reasonable to assume that, as long as such tools are able to import and export analysis information to data files, it is possible to implement a design environment such as the one shown in Figure 4. An example of such platform integration can be found in [4].

A real-time analysis tool for CAN-based systems is being developed within the Methodes project. It will implement a data import/export feature that will allow it to read/write real-time analysis data from/to XML files. This data will be extracted from and reinserted into XMI representations of UML models by an additional tool, currently being developed at Universidade do Minho.

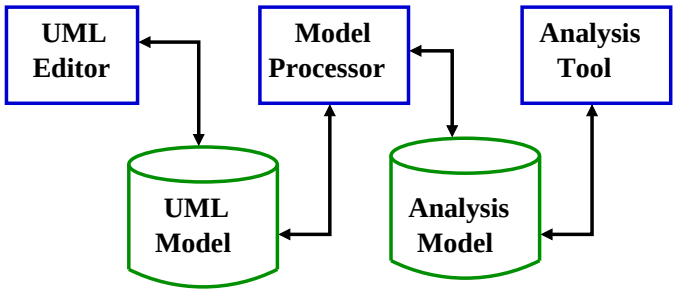

Figure 4: Design environment

4 modelling Approach

One trait that will be common to all stages of the design process is the need for a language to document system requirements and design solutions at all technological levels. For the reasons presented in Section 2, UML is a natural choice as the notational support for this MDD method. The following sections describe our use of UML throughout the development stages presented in Section 3.

The diagrams that will be presented constitute a small example that intends to demonstrate our development process. Due to limitations in the UML editing tool that was used to produce these diagrams, stereotypes are shown within note elements, together with the tagged values to which they are related. For the same reason, whenever note elements are associated with messages, this association is indicated by including the name of the message in the appropriate note element.

4.1 Black-box system requirements

At this level, the modelling is done through Use-Case Diagrams, Sequence Diagrams and Activity Diagrams. These diagrams are used to depict the intended interaction of the system with external entities or actors.

4.1.1 Use Case Diagrams

Use Case diagrams can be used without any adaptations to depict the system's interaction with the exterior. Each external entity appears as an actor, interacting with the system in a way that is put into context by the use case's name.

For use cases where no real-time QoS requirements apply, usually those relating to user interaction or gateway operation, the modelling process is the classical one used in the UML. Sequence diagrams can also be used in the usual way to further detail the functionality associated with the use case.

For process interactions, two stereotypes called *MInputSignal* and *MOutputSignal* were defined. They modify actors to represent input and output signals (from the system's point of view). Figure 5 shows a use case diagram where

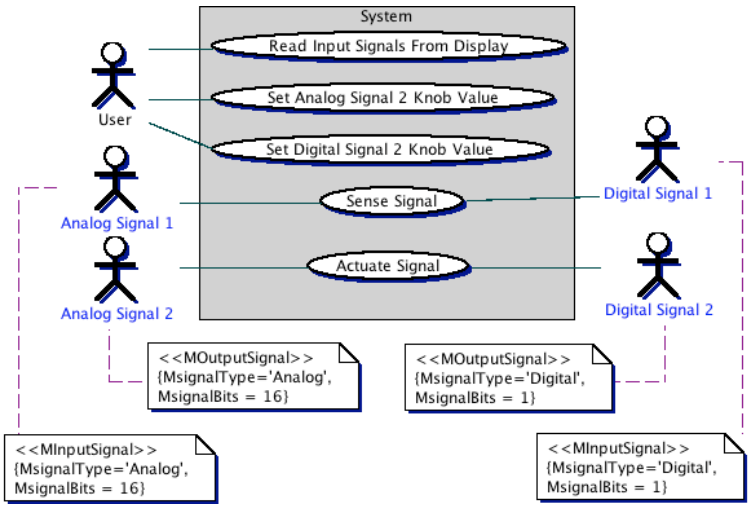

Figure 5: Use-case diagram

a system interfaces four process signals. The actors representing the signals are modified with a stereotype that specifies the nature of the signal. Associated with the stereotypes that identify the process signals are the *MsignalType* and *MsignalBits* tags, which are self explanatory.

This type of information can also be represented using a signal table that includes all relevant signal characteristics. In this case, if the timing characteristics for the signal are also included in the table, sequence diagrams may be omitted for these use cases. This approach would be similar to the one described in [1].

4.1.2 Sequence Diagrams

Sequence Diagrams are also used without major changes to provide detailed descriptions of the system functionality represented by a particular use case. However, for use cases with real-time QoS requirements, additional information must be included. This information consists of performance QoS requirements and, therefore, we chose to use the notational extensions defined in the performance analysis part of [7]. Each sequence diagram depicts an instance of an actor/system interaction. Represented in the diagram are the system itself, plus all actors representing related signals.

Input (resp. output) signals are represented as a message from (resp. to) the actor to (resp. from) the system. These messages are called `update` and they represent signal value events that must be read by (resp. produced by) the system. They are stereotyped as *PAopenLoad* and an occurrence pattern and required response time are also indicated. Figure 6 shows two examples.

If there is a cause/effect relationship between two signals, or if their inter-

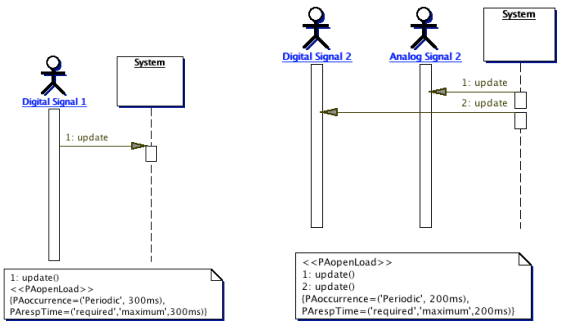

Figure 6: Sequence diagrams: input signal (left), output signals (right)

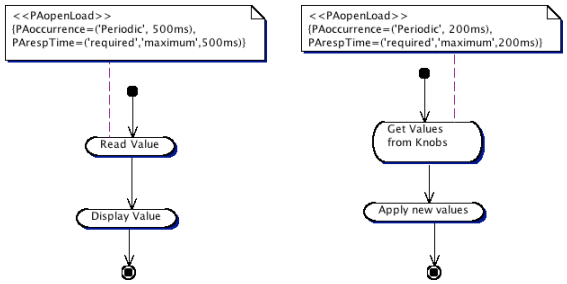

Figure 7: Activity diagrams: input signal (left), output signals (right)

action with the system is somehow related, they should be shown in the same sequence diagram.

4.1.3 Activity Diagrams

For each signal acquired or applied to the process, it is necessary to specify what the control system will do with the input signal values it consumes, and how these are used to produce new output signal values i.e. the control algorithm. This is best achieved using activity diagrams.

The examples in Figure 7 show how activity diagrams can be tagged using the stereotype described in the previous section, to show required end-to-end execution times.

4.2 Representing DCS architecture

The internal architecture of the system, showing the network infrastructure and the selected OTS devices can be represented using Deployment Diagrams. In our approach, each device is represented as a node. Stereotype *MOtSComponent* has been defined to identify OTS components, as shown in Figure 8.

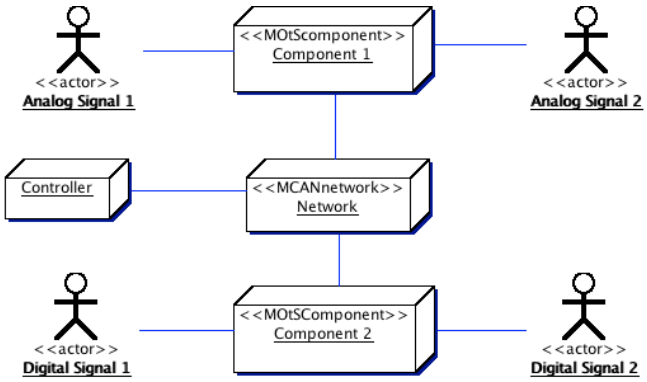

Figure 8: Global deployment diagram

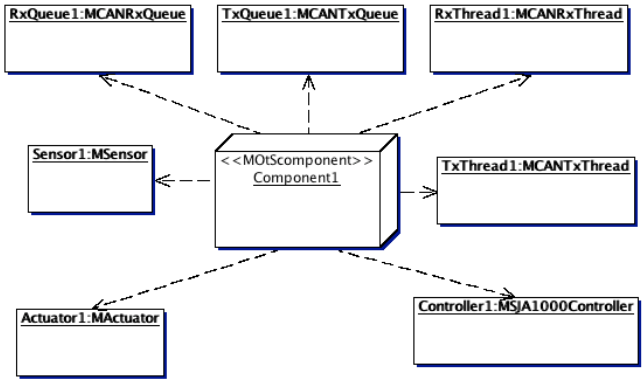

Figure 9: Deployment for an OTS CAN device

Note that the process signals identified in the requirements analysis are also shown in this diagram as actors associated with one of the OTS components. This is how signal assignments are represented. Also note that the communication infrastructure is represented by a special node stereotyped as *MCANnetwork*. Other nodes shown in the diagram in Figure 8, and not stereotyped, will be hosting controller software developed during the design process.

The (modeled) internal structure of each OTS component (introduced in Section 3.3) is described through additional Deployment Diagrams such as the one shown in Figure 9. This diagram shows the objects that emulate the OTS device internal software. In this case, the OTS component interfaces one input and one output signals, receiving one message, and transmitting another one.

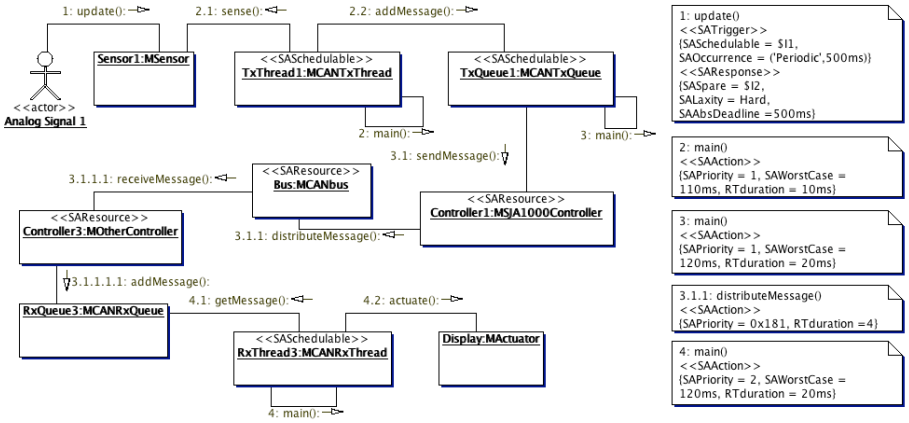

Figure 10: Collaboration diagram

4.3 Real-time modelling

The real-time QoS requirements and characteristics of a system depend on run-time factors, the most relevant of which are the processing platforms hosting the running software and the network activity on the fieldbus system. Additionally, the real-time characteristics of a system apply to interactions between object instances. These properties are not inherent to the classes involved, but to each instance itself [3]. In this paper we will be focusing on the interactions that are relevant for the real-time characteristics of the system, although many more interactions will exist in the model.

A real-time analysis consists of a schedulability analysis over a specialised view of the developed system. This is why the tags and stereotypes that we use to represent the real-time analysis data are taken from the schedulability part of [7]. The diagrams presented in this section were tailored for a specific real-time analysis technique. A small description of the parameters required by this technique are included in Section 4.4. Nevertheless, these models are sufficiently generic to be easily adaptable to other real-time analysis tools and even generic schedulability analysis tools that are compatible with [7].

Two UML diagrams can be used to depict interactions between object instances: Sequence and Collaboration Diagrams. Although they are interchangeable, here we will be using only the latter. Also, in collaboration diagrams it is possible to position object instances so as to emphasise deployment associations.

The collaboration diagram in Figure 10 shows the real-time parameters associated with one of the CAN messages in our example. Note that message exchanges are stereotyped differently.

One collaboration diagram similar to this one is included in the model for each CAN message transferred on the bus. Each of these diagrams depicts

the complete sequence of object interactions associated with the corresponding CAN message. Two types of messages can be used as triggers:

- Process signal events that give rise to CAN message transmission (either event-driven or timer-driven).
- Internal device events that, for some application-specific reason, trigger the transmission of CAN messages on the bus.

The workload associated with the scheduling job is composed of several message exchanges stereotyped as *SAAction* . There are two different uses for this stereotype:

- Software execution, as in the case of the `main` methods, which represents computational load.
- CAN message transmission, as in the case of the `distribute` function, where the load is associated with bus access and transmission delays.

The objects stereotyped as schedulable resources correspond to independent threads of execution inside a particular device. Collecting the different threads that are identified in the collaboration diagrams created for each CAN message, and correlating this information with the architecture information presented in Section 4.2, an analysis tool is able to enumerate all the threads operating concurrently in each device, their relative priorities and timing characteristics.

Finally, the objects stereotyped as shared resources in this diagram are all related with the CAN communication infrastructure. The object instance representing the CAN bus appears in all collaboration objects involving CAN communication in that particular bus. This resource is shared by all CAN controller object instances to which it is connected. The combination of the bus access (the `distribute` method) operations throughout these collaboration diagrams allows an analysis tool to work out bus access conflicts, task blocking due to collisions, and bus utilisation parameters. The object instances representing CAN controllers are shared, within the host device, by message transmission and message reception routines.

The object instance representing the CAN bus is conceptually deployed on a node stereotyped as *MCANnetwork*, as shown in the diagram in Figure 11. The *MCANNetwork* stereotype is derived from the *SAEngine* stereotype defined in [7]. It is seen as a computational resource with a particular processing speed (in this case the bus rate), schedulability and utilization. Messages transmitted on the bus are modeled as processes executing on this conceptual processor.

4.4 Interfacing real-time analysis tools

The collaboration diagrams presented in this paper constitute a small example of a DCS. This example was created for the real-time analysis tool being developed

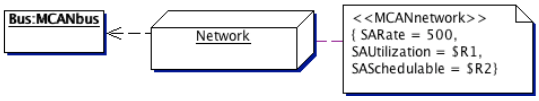

Figure 11: Deployment Diagram: CAN bus

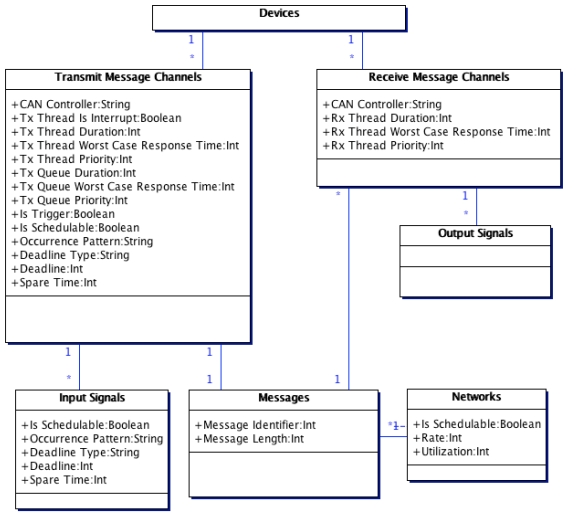

Figure 12: Real-time analysis data

within Methodes. The information exchanged with this tool is structured as shown in the class diagram in Figure 12.

In our implementation, this information is extracted from an XMI coding of the UML model, into an XML file which can be imported into the analysis tool. This extraction is performed by combining the structural information provided by the deployment and collaboration diagrams presented in Section 4 with the quantitative data specified using the stereotypes and tagged values also there described.

Note that the analysis results can be put back into the model using the reverse procedure. For example, in Figures 10 and 11 there are several tags that do not present numerical values, but variable names beginning with $. Analysis results can be traced back to these variable names and presented as the values of the corresponding tags.

5 Conclusions

DCS are a class of application with specific characteristics, such as the use of OTS components and fieldbuses, and the existence of real-time QoS requirements. In this paper we presented a MDD method that targets this category

of systems. This method focuses on the critical stages of DCS development. Namely, the specification of system requirements, the choice of OTS modules and fieldbus system, and the validation of the design using specialized real-time analysis tools.

The MDD method that was presented uses UML as support notation, as well as the extensions defined in the UML Profile for Schedulability, Performance and Time Specification [7]. The diagrams shown in this paper constitute a small example that illustrates the use of UML throughout the DCS development process. They apply to CAN-based systems and to real-time analysis tools developed within the Methodes project. Nevertheless, the approach that was taken is sufficiently generic to permit a simple adaptation to other fieldbus standards and analysis tools. Present and future work to be undertaken within this project includes the evaluation of the use of this methodology by industrial development teams and the automation of some of the steps in this development process.

References

[1] B. P. Douglass. *Real-Time UML: Developing Efficient Objects for Embedded Systems*. Object Technology. Addison-Wesley, 1998.

[2] M. Farsi and M. Barbosa. *CANopen Implementation: Applications to Industrial Networks*. Research Studies Press, 2000.

[3] J. M. Fernandes, R. J. Machado, and H. D. Santos. Modeling Industrial Embedded Systems with UML. In *8th ACM/IEEE/IFIP Int. Workshop on Hardware/Software Codesign (CODES 2000)*, pages 18–22. ACM Press, May 2000.

[4] Zonghua Gu, Sharath Kodase, Shige Wang, and Kang G. Shin. A Model-Base Approach to System-Level Dependency and Real-Time Analysis of Embedded Software. In *IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2003.

[5] D. Herzberg. UML-RT as a Candidate for Modeling Embedded Real-Time Systems in the Telecommunication Domain. In *2nd International Conference on the Unified Modeling Language (UML'99)*. Springer-Verlag, 1999.

[6] ISO11898. Road Vehicles, Interchange of Digital Information - Controller Area Network (CAN) for high-speed Communication, 1993.

[7] OMG. UML Profile for Schedulability, Performance, and Time Specification, 2002.

[8] B. Selic. Using UML for Modeling Complex Real-Time Systems. In Frank Mueller and Azer Bestavros, editors, *Languages, Compilers, and Tools for Embedded Systems, ACM SIGPLAN Workshop LCTES'98*, volume 1474 of *Lecture Notes in Computer Science*, pages 31–40. Springer-Verlag, June 1998.

[9] B. Selic. Turning Clockwise: Using UML in the Real-Time Domain. *Communications of the ACM*, 42(10):46–54, 1999.

[10] B. Selic, G. Gullekson, and P. T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley & Sons, 1994.

[11] B. Selic and J. Rumbaugh. Using UML for Modeling Complex Real-Time Systems. Technical report, ObjecTime Limited & Rational Software, 1998.