DATA ACCESS IN DISTRIBUTED CONTROL SYSTEMS

Matjaž Colnarič, Domen Verber

Univeristy of Maribor, Slovenia
colnaric|domen.verber@uni-mb.si

Abstract: When dealing with distributed control systems, the notions of nodes, messages, timetables, etc. are usually not transparent from the control application designer view. For that reason, in the IFATIS platform, the distributed shared memory model has been introduced to decouple the control application design from the issues of the hardware implementation.
To cope with that, for the control application, the underlying communication system and peripheral devices are only exposed as a set of data cells. Two processes can use such a cell to communicate with each other regardless of their current location. If they reside in different nodes, data written to a cell is transparently distributed through the system by means of statically scheduled TTCAN messages. For automatic data transformation and fault detection, each data cell can be associated with a validation and/or transformation routine.
To utilize this approach control application development tools (Matlab/Simulink and ControlBuild) have been adapted. *Copyright © 2005 IFAC*

Keywords: distributed control systems, local area networks, CAN, time triggered, distributed replicated shared memory.

1. INTRODUCTION

The ever-increasing complexity of the control systems today calls for distributed implementations. Control applications are executed over a set of distributed processing nodes that cooperate among themselves, with sensors, actuators, plant devices, etc. The fault-tolerant requirements of some of those systems additionally increase the complexity and difficulty of its implementation. For these reasons, it is very important that the dependability of such systems is systematically considered at all stages of the development and is not left to the experience and the intuition of the designers.

The issues in the implementation of fault-tolerant distributed control systems were investigated in the Intelligent Fault Tolerant Control in Integrated Systems (IFATIS) project. The major goals of its Workpackage 4, from which this paper is originating, are to explore the existing methodological and technological background, to develop the solutions and to implement an experimental fault-tolerant hardware platform. Its purpose is to allow for testing different fault-tolerant techniques provided by the partners in the project. Finally, a corresponding software tool was devised and adapted to the HW architecture (IFATIS ,2004).

As a part of the implementation of the platform a distributed replicated memory model has been implemented. There are several reasons why this model has been introduced into the control application design.

First of all, there is usually a logical gap between the representation of a control system and its implementation. From the design point of view, a typical control system consists of function blocks and different resources. Function blocks in control systems usually represent some sort of mathematical transformation (e.g. derivation, integration, etc.) although more sophisticated algorithms can also be included. Further, function blocks can be hierarchically organized or they can be decomposed into the simpler elements. Resources are used by function blocks to perform their operations. Usually these are some physical properties of the observed system (e.g. environment temperature). However, from the implementation point of view, the distributed control system consists of hardware components, application software and system software. Hardware components are processing modules, sensors, actuators and interconnecting busses. Beside some correlation between the abstract resources and the sensors and actuators, there is no obvious transformation from one representation to another. Processing modules can be represented by pre-built computing components, custom built components or special plant devices. Because of this, the translation of the control function into the equivalent code is heavily target dependent and no development tool can effectively support all combinations. By using the intermediate layer both issues can be observed and dealt with separately.

Further, the design of a complex control system is usually performed by engineers and scientists with good knowledge in automation control domain and with less knowledge in computer system domain. And vice versa, engineers that are responsible for the control system implementation usually lack knowledge about all the specifics of control systems. The development environment should thus provide support for the participants from both domains.

2. BRIEF DESCRIPTION OF THE IFATIS HARDWARE PLATFORM

IFATIS hardware platform consists of several processing and peripheral (I/O) modules (see Figure 1). The modules are interconnected with two time-triggered communication channels. For the communication and fault-detection special hardware was designed. To adequately support the execution of the control application on the target hardware, proper system software has been developed.

Processing nodes execute control application. Sensors and actuators are connected to the I/O nodes. For development and monitoring purposes, one of the nodes is connected to the host development platform.

Time-triggered communication allows for temporally deterministic distribution of all safety critical messages in the system. In the aerospace industry, time-triggered applications are today already state-of-the-art. Time-triggered automotive applications will be part of the next generation of products. In the Time-Triggered protocols, all messages between the nodes are sent in the predetermined time frames. One of the nodes on the communication channel periodically submits a so-called reference message and thus synchronizes the communication. Each message is sent within the specific frame relatively to the instant when reference message was received. In simple scenarios each possible message is sent in every cycle. However, in more general situations, a certain message may only be sent every second, third, etc. cycle. This way, the same message slot can be shared by different messages in different basic cycles. The reference message includes the information which basic cycle will be transmitted next. In Figure 2 an example of time-triggered communication with two basic cycles is shown. Message no. 1 is more volatile and must be sent in each cycle.

The configuration of timetables for basic cycles must be generated in advance (off-line) and all nodes in the system must be initialized with the same configu-

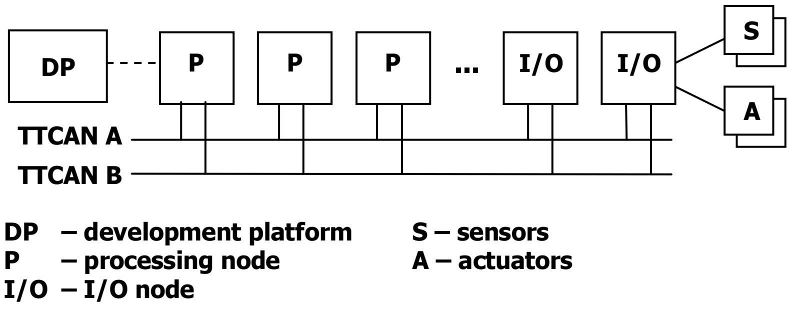

Fig.1. Diagram of the IFATIS hardware platform.

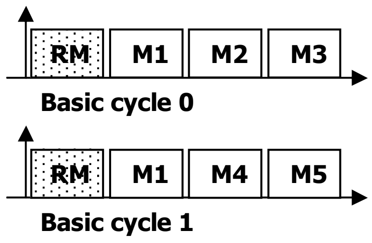

Fig.2. Example of TTCAN communication with two basic cycles

ration. This represents a drawback in the case when nodes should be reconfigured, because the timetables must also be reinitialized. As a consequence, an appropriate timetable must be prepared in advance for each of the possible reconfiguration scenarios.

There are several competitive solutions of time-triggered control busses. Many of them are based on CAN bus protocol as the network data layer because of its inexpensive infrastructure. One of the approaches is time-triggered CAN (TTCAN) protocol and is used with the IFATIS platform (Robert Bosch, 1991, ISO/CD 11898-4, 2004).

To avoid single point of failure, two communication channels have been implemented, although this also significantly increases the number of possible reconfiguration scenarios of the system.

The detailed description of the IFATIS hardware platform can be found in (Verber, et al., 2003).

3. DISTRIBUTED SHARED MEMORY MODEL

In control application, different function blocks communicate with each other, read data from sensors and produce signals for the actuators. In distributed control applications it is also necessary to know on what processing node the specific function block will reside or where and how to address specific I/O device.

From the application designer point of view, however, it is much more convenient if the communication is performed transparently. In this sense, a model of distributed shared memory has been devised. The distributed nature and other particularities of the hardware are thus hidden from the control application design. The specifics must be considered only in the latter stages of the development.

Distributed shared memory consists of a set of cells that serve as sources or sinks of the virtual communication links between the application, hardware platform and the environment (see Fig. 3). Two processes can use a cell to communicate with each other regardless of their current location (i.e. on which processing nodes the processes run). The same mechanism is used for the communication between control application and sensors or actuators.

When both source and destination peer for the communication resides on the same processing node, simple memory transfer of data can be used. On the other hand, when the communication spans over two or more nodes, the data written into a cell in one node must be transparently distributed to all appropriate nodes in the systems by means of some sort of middleware, see Fig 4.

The middleware is a well-recognized approach in modern computer systems. There are well known distributed communication subsystems like CORBA or DCOM that are being used in classic computer systems (OMG Group, 2002 and Microsoft 2002). However, these protocols usually introduce too much overhead into the system architecture to be used in the embedded real-time control applications. Because of that, for the IFATIS project, time-triggered communication as described above has been used. Memory cells that must be distributed through the system are mapped into the related TTCAN messages and sent periodically to other nodes in the system. To achieve temporally predictable behaviour, the overall operation of the system is synchronized with those time intervals. Each execution cycle starts with the reception of the reference message. Then, all outputs of the previous cycle are transmitted to the other nodes in the system and serve as an input to a new cycle. The behaviour is similar to the traditional PLC operation and is used by most control application development tools.

However, there is a problem using time-triggered communication. Generating the optimal schedule for specific control applications has exponential complexity. The complexity is even larger when the system is requested to be fault-tolerant. In the case of a fault the control application must swiftly adapt to the new situations. This can include a requirement for rearrangement of the communication pathways in

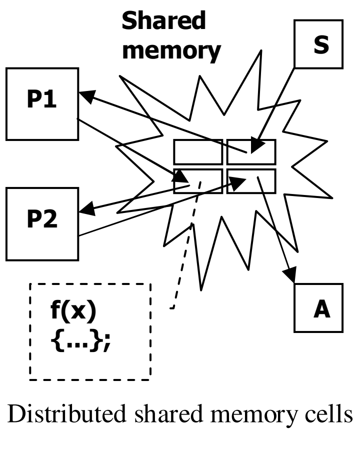

Fig. 3: Distributed shared memory cells model

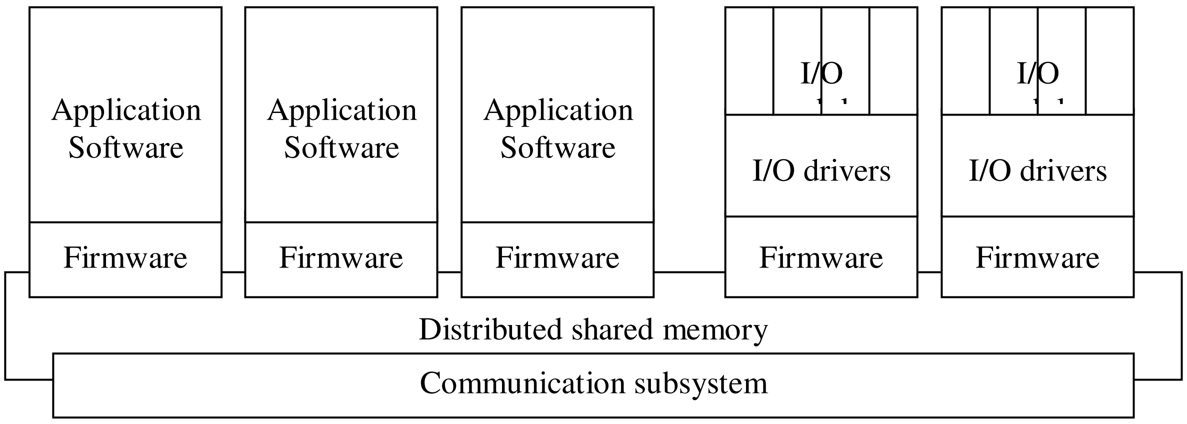

Fig 4. Implementation of DSM model

the system. Thus, for time-triggered protocols, a new message schedule must be used. Because the timetables cannot be generated dynamically, all possible resolutions must be prepared in advance. To cope with this problem an off-line tool for TTCAN timetable generation has been implemented as a part of the IFATIS project. Based on the information of all possible producers and consumers in all possible configurations of the systems appropriate timetables by means of C-language header files are generated to be included later in the application code.

The underlying communication protocol also introduces some other limitations. In general, the maximum size of the message also limits the amount of data that can be put into a single cell. This can be overcome if several messages are used for a single memory cell. Further, the communication speed confines the overall performance of the application.

This drawback can be diminished if a faster communication protocol is used. Or, if redundant communication channels are available, they can share the load to increase the overall throughput of the communication. In normal operational mode the communication busload is distributed over all available channels. However, this solution increases the complexity of message's timetable generation because the schedules for those buses must be synchronised. And, of course, if some of the communication channels fail, the busload must be rearranged and the communication time increases.

There are other possibilities to utilize the proposed shared memory model. First, because the data between the control application and peripherals are transferred indirectly, some sort of data transformation can be applied. Control application usually deals with abstract representation of some physical quantities from the environment (e.g. temperature represented in the degrees of Celsius). On the other hand, the acquisition and the actuation of those quantities are performed by simple input/output devices (e.g. A/D converter connected to the temperature sensor). Those devices use their own kind of data representation. To simplify control application development, the transformation between different value domains can be done transparently by the system software.

As an example, an integer value provided by a temperature sensor can be transformed into the appropriate floating-point value for the application. For this, each memory cell can be associated with two transformation functions. The first is applied when data are written to and the second is applied when data are read from the cell. If, for some reason, the temperature sensor must be replaced with a different one only the transformation functions must be updated accordingly.

In addition to the application data, the memory cell can embrace some other attributes that are provided by the application (e.g. quality of the information generated by the producer) or the systems software (e.g. timestamp when the data has been generated). System software can also monitor when data in a cell actually change and signals the control application, etc.

By using shared memory model it is also possible to perform a simple and unified kind of tracing and diagnostic of all important state variables used by the control application. Because the information written to a cell is transparently distributed to all nodes in the systems, it is possible to have a dedicated monitoring node that acquires the current state of the system.

Current version of the IFATIS system software supports the use of maximum 32 shared memory cells with five basic data types (byte, word, integer, float and double). This functionality can be expanded to a larger number of memory cells and for using custom data types with maximum data size of eight bytes. Communication layer manages the replication of memory cells in the background as a separate task.

4. SUPPORT FOR FAULT-DETECTION AND FAULT-TOLERANCE

There are several techniques to deal with faults in the control systems. However, the first step is to detect the fault. To do this some kind of dependable monitoring system must be implemented, which detects abnormalities in the system and triggers appropriate corrective actions.

Using the shared memory model, some faults can be detected by evaluating the values in the memory cells. E.g., the data transformation routine (described above) can be extended to check the plausibility of data written into a cell. The simplest kind of validation is range checking. It is expected that some variables can only hold values in a certain range (e.g. the temperature of the environment should not be below 5 degrees or above 50 degrees Celsius). The out-of-range values signal some sort of fault in a system. In similar way more sophisticated validation can be performed. For example, the dynamic of the changing of values can be observed if a previously written value is compared to a new one, the minimum or maximum frequency of data arrivals can be monitored if time-stamps are observed, etc. Although it is more efficient if those tests are done by dedicated hardware it is much more flexible to use software routines. In the case of floating-point values this is usually the only feasible solution.

The shared memory model may not only be used for fault-detection; it can also simplify the fault-tolerance of the overall system in the case of faults. Fault-tolerance of the hardware is usually dealt with by redundancy and diversity. When a fault occurs, the system must be reconfigured. With shared memory model approach this can be implemented in such way that the reconfiguration is transparent to the control application code.

For example, one of the possible faults in the system is when one of the nodes in the system fails. In this case the tasks performed by the failed node must be reallocated to other ones. Because of the shared memory model, the transition from one configuration to another is simplified. It is not important on which processing node the initiator for the message is allocated. In most cases the timetable for the message scheduling can remain the same. In some cases, however, because of the reduced resources of the system, some degradation in the performances will be necessary and must be considered in the configuration of the application (e.g. by switching off some non-important tasks, by increasing some response times, by gracefully degrading the accuracy of some calculations, etc.).

In the same way the faults of sensors and actuators can be dealt with. By using redundancy, several sensors can be used to measure the same physical quantities, but only one of them (the one that operates most accurately) is used to produce the value that is distributed through the system. It is even possible to replace the physical sensor with the software component that produces (i.e. calculates or estimates) appropriate values for the system without the need for changes in the control application.

However, there is also a difficulty by using shared memory model. As mentioned above, several communication channels can be used to increase the overall throughput of the shared memory related messages. But, in the case of failure on one of the channels, the communication must be performed by the others and some degradation in the communication response times can be expected.

5. SUPPORT FOR CONTROL APPLICATION DEVELOPMENT

Use of the shared memory model would have no practical value, if it would not be integrated within the control application development life cycle. For this, the application development tool should be adapted to support the shared memory model. As a part of the IFATIS project the official development tool, ControlBuild, has been expanded in such a way. Also, one of the most frequently used development tools for control application development, Matlab/Simulink, has been upgraded.

To implement the shared memory model two additional function blocks have been added to the development tool library: one for reading from and one for writing into the specific memory cell. The custom dialog window is used to define specific parameters: type of data that will be sent or received, shared memory slot identifier, status if the same data should be sent over both TTCAN channels (redundancy), how frequently data should be updated, etc. Some of the memory blocks are used for the communication with the peripheral devices. For those blocks several additional parameters must be set. For example, parameters for mapping between application specific values and device specific values can be defined to support data transformation. To allow for this, two value intervals can also be defined. The first interval defines the expected range of the value produced by the application and the second one defines the range of the value expected by the hardware device. The given range also represents the minimum and maximum values of the data and serves as some kind of limiter or it can be used for the data validation. As a shortcoming, only linear transformation can be described at this time. More complex transformation and other fault-detection mechanisms can be (at this time) implemented in C and added later to the application code.

Because the redundancy and the diversity can be utilized, it is possible that more than one function block produces the data for a specific memory cell, however not at the same time. Thus, there may be more than one output memory block with the same ID. To assure that only one of the replicas actually produces the value, the sending block has an additional enable signal. This signal is used to enable and disable the data transmission in different operation modes. The signal is set according to the current execution state of the application.

The process proposed for building a fault-tolerant control application by using the solutions described above would be as follows.

First, the development of the control application is performed as usually without considering the target platform. Only the limitations on what function blocks are usable for the implementation should be considered. The model should be tested and validated by means of all usual practices from the control application domain.

Later, the model must be prepared for execution in the distributed environment. The control application must be divided into modules that will be executed on different processing nodes. In the case of the IFATIS tool this is done by division of the model into the sub models. Because all code is preloaded onto the processing nodes prior to execution, also all possible execution modes of the system must be prepared. To simplify this, each subsystem can be further divided into several smaller ones that represent the model of execution in different execution modes. Then, the shared memory blocks are introduced to describe the connection links between different subsystems. After that, the application can be simulated and validated again as a distributed system. Sampling effects due to latencies introduced by the TTCAN bus can also be simulated accurately. Later, when the result of the testing is satisfied, the specific blocks are augmented with more detailed implementation attributes. Based on the information about the nodes and the memory cell blocks, appropriate timetables for TTCAN messages are generated. Finally, the code is generated and loaded into the target where additional testing and validation can be performed directly on the target.

6. CONCLUSION

During the design and implementation of the IFATIS experimental platform several approaches to implement a reliable and fault-tolerant distributed computer system were tested. One of the solutions to partial problems was the distributed shared memory model that has been represented in this paper. Although this approach still exposes some difficulties it proved to be a very efficient solution to transparently deal with data interchange and with faults in the system almost independently of control application design and implementation.
In the future work the distributed shared memory approach will be further elaborated to increase the dependability and to reduce the number of required TTCAN timetables for all possible configurations of the system. As an experiment, for massive data transfer, an Ethernet communication channel will be added in parallel to the TTCAN communication to increase the overall throughput and response times of the system.

REFERENCES

Bosch R. (1991). *Controller Area Network (CAN) Specification, Version 2.0.* http://www.can.bosch.com

ISO/CD 11898-4 (2004) Road vehicles – Controller area network (CAN) – Part 4: Time triggered communication.

IFATIS (2004). Intelligent Fault Tolerant Control in Integrated Systems. IST-2001-32122. http://ifatis.uni-duisburg.de

Microsoft (2002). *Distributed COM (DCOM) specifications.* http://www.microsoft.com/com

OMG Group (2002). *Corba specifications.* http://www.corba.com

Verber D and Colnarič M. (2003) "Issues in the implementation of a fault-tolerant hardware platform", in 5th IFAC Symposium on fault detection, supervision and safety of technical processes, Washington, D.C., USA, June 9-11, 2003 p. 561-566.

Verber D., Colnarič M. and Halang W. A. (2002). Fault detection in safety-critical embedded systems. In: Design and analysis of distributed embedded systems (Kleinjohann B. (ed.)), p. 113-119. Kluwer Academic Publishers, Boston.