Martin Törngren

*Address:* Division of Mechatronics, Dept. of Machine Design, The Royal Institute of Technology, S–100 44 Stockholm, Sweden. *Fax*: +46–8–202287, *Email*: martin@damek.kth.se, *WWW*: http://www.damek.kth.se

# A perspective to the Design of Distributed Real–time Control Applications based on CAN

**The Controller Area Network (CAN) constitutes a good low level base for distributed real–time control systems. The priority mechanism allows for tailoring by choice of scheduling policy. It is however unfortunate that the error handling policy of CAN is fixed, this limits the solution space for the designer. Clock synchronization should be added to the basic CAN system in order to obtain a global system clock. Designing a distributed system around CAN requires the identification of the appropriate level of system decentralization and a suitable system organization. These design aspects are presented and the use of CAN as a basis for distributed control systems is discussed.**

## 1. Introduction

The work on communication over the past years, reflected by available protocols, clearly indicate the wide applicability of distributed control system approaches. Communication protocols can coarsely be grouped into *fieldbuses* (e.g. FIP and PROFIBUS), *automotive buses* (e.g. CAN), ”other” *machine buses* (e.g. 1553B and the IEC train network), *general purpose networks* (e.g. IEEE LAN’s) and a number of *research protocols* (e.g. TTP), Törngren (1995). It is obvious that the wide applicability of distributed system approaches for applications with different characteristics has yielded a multitude of concepts and components for the design of such systems.

Two important problems facing designers of real–time control systems are due to the impact of non real–time techniques and systems, to which the majority of (research) work has been devoted, and the inherent inter–disciplinarity of real–time control systems. These gaps have probably contributed significantly to delaying the introduction of distributed control systems. Further, the gap between academia and industry has delayed the transfer of knowledge regarding the design of distributed real–time control systems. The main purpose of this paper is to present the Controller Area Network protocol (CAN), developed by Bosch and Intel for the automotive market and an ISO standard, from a number of perspectives on distributed real–time control systems. Section 2 provides the perspectives and in Section 3 CAN is presented and compared with FIP, the factory instrumentation protocol. Section 4 briefly presents essential design aspects for distributed control systems. Section 5 discusses the use of CAN as a basis for distributed control systems and Section 6 gives conclusions.

## 2. Perspectives to distributed real–time control systems

The design of distributed real–time control systems requires skills and knowledge from several engineering disciplines. True parallelism and real–time operation together with dependability requirements can make design difficult. In this regard, perspectives are very important.

**A brief non–technical perspective.** Much insight, intuition and information can be gained by comparing distributed real–time systems with human distributed systems. Fox (1981) shows that such analogies are useful by comparing distributed computer systems with distributed human organizations and by attempting to apply organization theory to computer systems. Interesting aspects including the selection of organizational structures (i.e. entities and communication paths), the selection of a "control regime", encapsulation, control delegation and coordination are discussed. One conclusion is that decentralization is a highly appropriate form of organization for complex distributed computer systems. There are many other further common life analogies that can be drawn. This topic is further elaborated by Törngren (1995).

**A brief historical perspective.** User needs and technological advances paved way for the introduction of *direct digital control* systems in 1963 (i.e. a system where a computer is directly interfaced to and controls a particular system), *digital multiplexing* in serial communication in the early 1970s and the first *distributed computer control systems* in the middle and late 1970s. At this period of time, research activities were consequently increased, evident from the fact that IEEE and IEE conferences on distributed computer control systems and distributed processing were started. With lower costs, increasing performance and reliability of digital technology, distributed control systems were soon thereafter applied in integrated manufacturing. Papers treating the use of distributed control in machinery also appeared at this time, see Duffie (1982). Excellent work dealing with some of the fundamentals of distributed control systems was produced in the early days of distributed processing. For example, arguments for establishing a global clock as a fundamental base for distributed applications was put forward by Kopetz (1983) who also discussed the use of datagrams for real–time applications instead of conventional positive acknowledgement and retransmission protocols. In an early work on distributed processing by Jensen and Boebert (1976), the partitioning and allocation phases are discussed. Enslow (1978) and Jensen (1981) clarified levels and degrees of decentralization. A communication model similar to that used by CAN was presented by Guth and d'Epinay (1983) in the context of process control applications (further discussed in section 3) and various design aspects discussed by Ramseyer (1979), Wolff (1977). In a historical perspective, machine embedded control applications constitute the most recent area for distributed computer systems.

It is interesting to note the following quote by Motus and Rodd (1994): *"Embedded or real–time software was for a long period left to non–professional programmers, such as control and mechanical engineers".* This indicates the gap from computer science to control applications. It has also been stated that computer science lacks a historical perspective, Laplante (1995). This is related to inter–disciplinarity since real–time control systems based on mechanics, pneumatics, hydraulics, analog electronics and humans have existed for ages. What is fairly new is the art of computer engineering and science, and the design of computer based real–time systems. Nevertheless, in parallel to industrial control systems and spurred by the classical Liu and Layland (1973) paper, in particular real–time scheduling theory became a vital research field in academic computer science. The focus on general purpose computing systems combined with the lack of inter–disciplinary interaction is believed to have lead to an uncertainty regarding important design parameters for distributed real–time control systems. It is therefore appropriate to continue with an area perspective.

**Real–time control systems and the inter–disciplinary gap.** Performing real–time control means that it is no longer possible to rely on conventional abstraction mechanisms that provide an abstract view of the underlying hardware through the use of operating systems and communication protocols. This is so since the actual timing behaviour depends on the implementation of these abstractions, which therefore need to specifically address real–time. The basis for the design of general purpose (non real–time) distributed computer systems, however, is made up of components that provide *high average performance.* The focus on average performance and the major difference in approach compared to real–time systems is well illustrated by Tannenbaum (1995): *"On account of the constraints on real–time distributed systems, their protocols are often quite unusual. In this section we will examine one such protocol, TTP (Time– Triggered Protocol) (Kopetz and Grundsteidl, 1994), which is as different from the Ethernet protocol as a Victorian drawing room is from a Wild West Saloon."*

Significant time–variations typically occur with the use of general purpose contemporary computers. That is, the *dynamic factor* $\delta_{max}/\delta_{min}$ will be large. The factor is a measure of the time–variations that can be expected where $\delta$ denotes the time from start to end of a particular computer activity (program, process, etc.). Time–variations typically have the effect to deteriorate control performance for control applications. This fact has spurred research activities in two fundamentally opposite directions. The ”real–time branch” of computer science deals with e.g. real–time scheduling theory and the design of predictable computer and communication components. Recent work in control theory, on the other hand, deals with the design of feedback control systems that provide robustness towards time–variations, data–loss and other transient failures. These directions indicate the interesting trade–off and interaction between control and computer engineering regarding timing problems in real–time control systems; solutions can typically be provided either by computer *or* control engineering design, see Törngren (1995) for more on this topic.

The overemphasis on bitrate performance of communication links and the influence of OSI oriented communication is worth noting. For a distributed real–time control system, bit rates at the physical layer do not really give much information. The effective bandwidth of a protocol is often significantly lower than the bitrate. Furthermore, the transmission delay is only a fraction of the end–to–end delay, including delays at sending and receiving nodes. Therefore, system wide execution strategies and end–to–end delay considerations are essential. The OSI model, services and protocols were originally designed to connect larger computers in a wide area network, not requiring real–time communication and aimed towards connection–oriented services between a pair of agents. The transmission of, for instance, short but frequent and time–critical data is not until very recently considered by ISO standards, Stallings (1995).

## 3. Properties of CAN and comparison with FIP

Among the very large number of serial communication protocols, CAN and FIP, the Factory Instrumentation Protocol, Leterrier (1992), are exceptional since they were explicitly designed for real–time control applications, Leterrier (1992), Kiencke (1994). Dissimilarities include their origin, automotive vs. process control/factory automation. The relevance of these protocols is further justified by the fact that CAN is commercially highly competitive and that FIP includes an application layer and tools for distributed real–time applications. CAN and FIP are compared in Tables 1 and 2 (data–link layer and above) and further discussed in the following.

| | *CAN* | *FIP* |
|---|---|---|
| *Layers:* | 1, 2 | 1, 2, 7 |
| *Scheduling policy/mechanism:* | None[1] /Preemption & priorities | Static & run–time / [5] |
| *Error detection:* | FCS and much more[2] | FCS |
| *Error handling:* | Retransmission | None[6] |
| *Effective bandwidth[3]* | $\sim$ 0.49 | $\sim$ 0.06–0.89 |
| *Data length periodic messages:* | 0–8 bytes | 0–126 bytes |
| *Maximum update frequency[4]:* | $\sim$ 6 kHz | $\sim$ 200 Hz |

TABLE 1: Some properties of CAN and FIP

1) The CAN specification leaves the choice of scheduling policy open. See further below.
2) FCS=Frame Check Sequence, monitoring, frame check, bit stuff and error signalling.
3) Effective bandwidth = [Number of data bits]/[total number of frame bits + scheduling overhead]. Figures for small and large periodic messages (FIP) and 8 byte messages (CAN).
4) Approximate maximum update frequency for periodic messages with a bitrate of 1 Mbit/s.
5) Local clocks for centralized static scheduling. Aperiodic messages are discussed below.
6) Acknowledged and datagram services are available.

| | *CAN* | *FIP* |
|---|---|---|
| *Application layer support for real–time (RT) systems* | *+/–* | *+* |
| *Suitability for RT–event triggered systems* | *+* | *–* |
| *Suitability for RT–time triggered systems* | *+* | *+* |
| *Error detection mechanisms* | *++* | *+* |
| *Error handling* | *–* | *+* |
| *Cost* | *++* | *–* |

TABLE 2: Qualitative comparison of CAN and FIP.

**The communication model used (application layer).** ”Communication model” is here informally used to refer to the communication view that is provided to the application by the communication subsystem. This view is primarily related to the logical behaviour of communication and includes connectivity (e.g. *1–1* or *1–n*, uni– or bidirectional, connection–oriented vs. connection–less); whether communication is blocking or not and the basic addressing principle (sender–receiver vs. source addressing only).

The model provided by FIP and CAN has a similar basis. It has been referred to as the *distributed data–flow communication model*, formalized by Guth and d'Epinay (1983), or alternatively as the Producer–Consumer model. The model is most notably based on source addressing and broadcasting of information. It can further be classified as unidirectional, connection–less and non blocking. Receivers by use of an associate receiver select the information of interest for reception. The communication abstraction provided can be viewed as logical channels connecting distributed application functions, forming a distributed database. Communication can be performed based on *periodic updating* (time–triggering) or *event–controlled updating* (data change triggering). The communication model provided is simple, allows a straight forward mapping for control applications, Törngren (1995) and promotes system flexibility in terms of decoupling communicating processes. It is interesting to note that a similar model has been developed in non real–time distributed systems, termed *event–based, implicit invocation,* Garlan and Shaw (1993). Stated advantages include strong reuse support and ease of system evolution. For non real–time systems, stated disadvantages include the lack of control from the point of view of source components. For real–time systems this is less of a problem since control–flow anyway must be specified and managed. Further, error detection is appropriate at receivers of data, see e.g. Rodd et al. (1990).

The lack of a standardized application layer protocol in CAN has caused a lot of activities related to different applications, see e.g. CiA (1994) for a survey of protocol proposals. As indicated in Table 2 for CAN, this lack can either be seen as a merit, since it enables the design of application specific higher layer protocols, or as a drawback due to the lack of standard and stability.

FIP, on the other hand, provides an application layer which includes support for periodic updating and specific consistency and synchronization services (which ensure that a number of nodes have received the same information and that its use can be synchronized). A special mechanism and policy is provided for aperiodic messages which are granted access by the communication master. The master can accept requests during ordinary TDMA communication and can then provide any spare capacity, in terms of empty TDMA slots, to these aperiodic requests. Additionally, FIP provides tool support for distributed application development. This is an advantage compared to CAN.

**Communication scheduling.** One of the central demands of a real–time computer control system is the need of predictable timing behaviour. In a distributed control system a number of nodes are connected through a shared communication network. Therefore, resource sharing or scheduling (termed medium access control for communication) is a topic of major importance. If several time–critical control activities are multiplexed onto a computer system it is not sufficient to provide a fast computer system. *Scheduling policies* determine which entity (e.g. process or message) that at a given time instant can use a given resource. A policy consists of a number of rules and algorithms

which implement the aim of the policy by use of a number of ”low–level” *mechanisms*, compare for instance with preemption and dispatching mechanisms in an operating system. Scheduling can of course to some extent be avoided by using more resources, but cost always tend to introduce some degree of multiplexing. A scheduling policy is furthermore characterized by its suitability for event vs. time–triggered systems.

CAN provides a controlled access protocol by use of message priorities and a priority resolution (preemption) mechanism. I.e., CAN provides a mechanism but does not specify how priorities should be assigned to messages or impose any restrictions on when messages can be transmitted. Unfortunately this together with the launching of the protocol as an ”event–driven one” has created misconceptions about the use of CAN, Törngren and Backman (1993), Tindell and Burns (1994). Consequently CAN has sometimes been considered to have stochastic delay properties for all but the highest priority. Clearly, from the extreme point of view of an entirely event–driven system where events can occur as often as possible and priorities are assigned to messages without any specific rules, the system does become as stochastic as the assumed environment. By use of concepts developed in 70s and 80s, such as rate–monotonic scheduling, critical instant analysis (Liu and Layland (1973)) and sporadic servers (Mok (1983)) predictable real–time scheduling is easy to achieve by use of the priority mechanism provided by CAN (see further below and section 5).

Consequently, it is in principle possible to apply a number of policies to a CAN system, including *static scheduling (*e.g. some form of time division multiple access – TDMA), a *rate monotonic style of scheduling*, or the *waiting room protocol*. This allows CAN to be tailored towards event– or time–triggered systems by choice of scheduling policy, compare with Table 2. The policies are briefly introduced below.

In TDMA, time slots for communication are statically assigned to nodes. Even though this policy does not require a priority mechanism, such a mechanism can potentially be used to implement a hybrid scheduling policy, e.g. using one priority for a static schedule and higher priorities for certain alarm or mode change messages. TDMA can be performed based on *centralized (*master/slave or polling*)* or *decentralized* (master/master) *control*. In either case, transmission rights are granted to nodes at predetermined time instants according to the static schedule. Centralized control is exemplified by FIP, which as opposed to traditional polled systems allows communication slaves to communicate directly with other slaves/nodes. Because FIP primarily uses static scheduling it is less suited for event–driven systems. Drawbacks for centralized control schemes include the overhead caused by master polling and the single point of failure that the master constitutes. In decentralized control a global clock is required and used to implement distributed static scheduling. This approach eliminates the disadvantages in master/slave communication systems. The Time Triggered Protocol – TTP is one example, Kopetz and Grundsteidl (1994).

Rate monotonic scheduling was initially developed by Liu and Layland (1973) for the scheduling of independent periodic processes on a single processor. With rate monotonic assignment, priorities are assigned such that processes get a priority corresponding to their frequency: high frequency yielding high priority. A preemption mechanism is required to ensure that the process with the highest priority always executes. Under these conditions a sufficient utilization condition exists to check whether all processes will finish their computation requirements each period before their current period expires. Since then, rate monotonic theory has been extended to deal with aperiodic processes, communication scheduling and processes where deadlines are shorter than the period, etc., see Audsley et. al for an overview (1995). This collection of theoretical work is here referred to as the rate monotonic framework. Tindell and Burns (1994) illustrated the application of this framework to CAN scheduling.

In the Waiting Room Protocol (Ramamritham (1987)) a logical waiting room is established. Each node maintains its own local queue of messages. At the end of a transmission round the first message in each local queue enters the waiting room. During a transmission round all messages in the waiting room are transmitted one by one. A message round is implemented by means of prioritized arbitration, where each node is assigned one unique priority. Messages from the waiting room will always have higher priority than messages not in the waiting room. The waiting room

protocol is advantageous compared to token passing scheduling. There is no token which can be lost and if a node does not have a message to send it will not consume any bandwidth.

**Communication (logical) reliability – error detection and handling.** Specific attention must be given to *error handling policies* and the provision of suitable *error detection mechanisms*. A prerequisite is clearly that a very high degree of communication faults are detected and reported as early as possible. Fault tolerance support may additionally be required (for reliability or availability reasons), in order to provide higher probability of message delivery in spite of failures. In a real–time application the availability of time is limited and thus e.g. the number of retransmissions possible before the data–item becomes invalid are also limited. Different scheduling and error–handling policies need to be applied depending on application requirements and characteristics.

CAN does provide excellent error detection mechanisms, see e.g. Charzinski (1994). Several mechanisms complement a frame check sequence as indicated in Table 2. If a node detects an error it immediately transmits an error frame to notify other nodes. CAN also provides a hardware acknowledge which indicates to the transmitter that there is at least one other functioning node on the network. The acknowledgement, however, does not say that the message actually has been received by any CAN–controller. This, it can be argued, would have been more useful information. Compared to CAN, FIP only provides a frame check sequence.

Unfortunately, the CAN protocol specifies a *fixed error handling policy.* I.e. when a transmitting node has detected an error regarding a transmitted frame, the CAN protocol specifies that immediate retransmission of the message must take place. For a real–time application it would have been more preferable that the maximum number of retransmissions was a configuration parameter, thus leaving the policy open, as for the scheduling policy. The consequence of the fixed policy is that the designer is left with fewer degrees of freedom. For example, a transmission error can often be handled by the application by use of e.g. prediction in control applications, or by use of oversampling. While these options remain, the hardwired policy reduces the available bandwidth. Further, the alternative to use static scheduling (as discussed above) is made very complicated. In FIP, datagram and acknowledged services exist. For periodic broadcasting no retransmission is specified.

## 4. Essential design aspects for distributed real–time control systems

In a wider perspective and from a technical point of view, the design of distributed real–time control systems requires methodologies, models and tools that facilitate systems engineering to determine an appropriate level of application decentralization and a suitable way of organizing the distributed application. Up till now, practical work has focussed on tools supporting on–line analysis of already designed systems rather than supporting the design phase. Most existing methodologies do not address decentralization but the problems have been addressed in research. A *decentralization methodology* for machine control applications by Törngren (1995) is illustrated in Fig. 1 and briefly outlined in the following.

The appropriate level of decentralization depends on a number of factors, including costs, load balancing, reliability and real–time requirements. Load balancing is an important system property that will improve system flexibility and utilization margins. Real–time requirements include bounded and/or constant delays as well as time synchronized actions. A typical requirement can be to minimize control delays (the time from sampling to actuation), for given sampling frequencies, to increase control performance. Modularity and flexibility are desirable properties, which however are difficult to quantify. Use of natural decomposition and allocation approaches has advantages in this regard. Decentralization is typically suitable to formulate as an optimization problem.

The methodology adds a number of design steps to conventional design methodologies. The steps specifically deals with implementation oriented application structuring, the problem of finding a suitable match between the application and the resource structure (”mapping”), and with execution strategy considerations. The steps can be applied in different stages, for systems and subsystems

and early as well as late in design. They are interdependent, related through iterations, and rely on knowledge about application and implementation characteristics. The need for modelling in order to exploit the solution space and understand the system is essential, but not further treated in this paper.

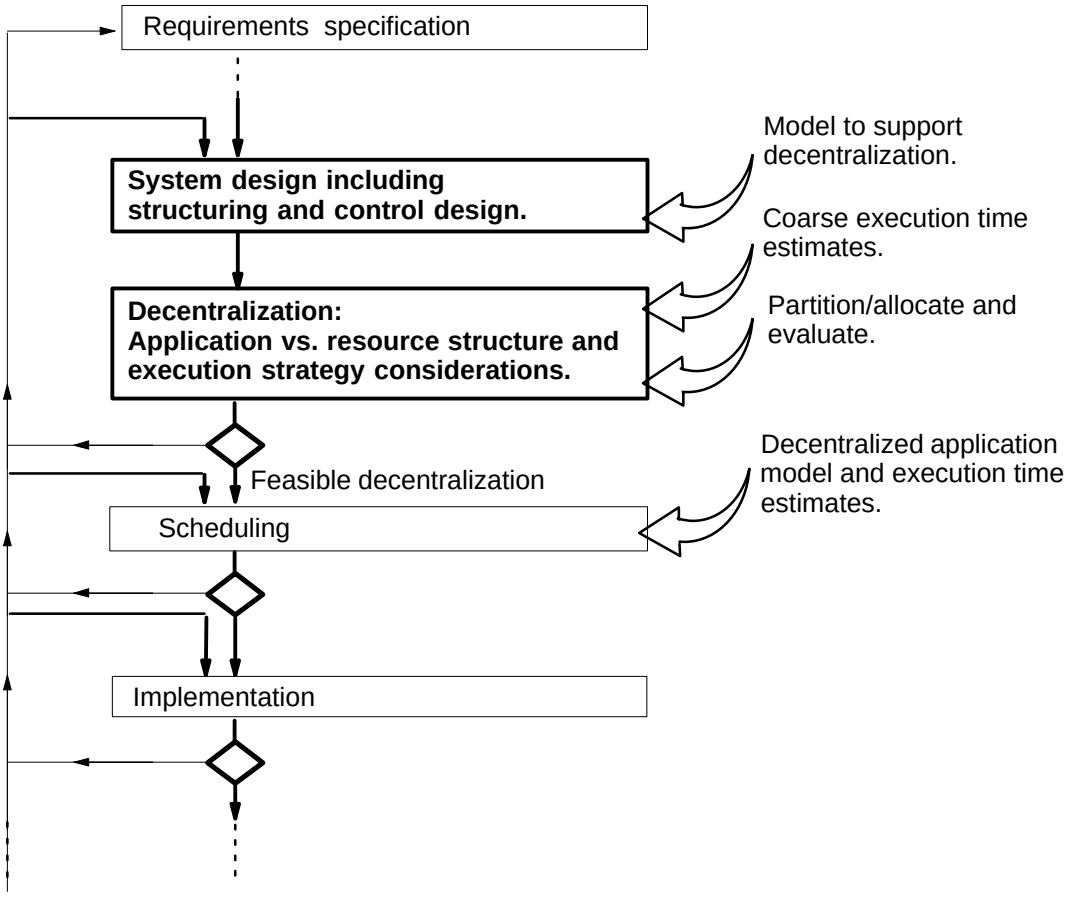


**Fig. 1.** Decentralization considerations for control applications.

**System design including structuring and control design.** Structuring, through decomposition, yields elementary functions which are the basic units that can be considered for distribution and parallel execution. I/O bounds can be used as a basis for decomposition and allocation. An I/O bound refers to a clear data–flow relation between an elementary function and one or more I/O points (sensor and/or actuators). An allocation strategy that is based on the I/O bounds of functions is referred to as *locality based distribution* – basically implying that local functions are executed locally.

Natural ways of decomposition include functional, spatial, dynamical and organizational decomposition. The second has a clear relation to I/O boundness, as introduced previously. The third, has relations to system hierarchical structuring, but also to the design organization. I.e. decomposition may be given by the company/subcontractor etc. structure. Dynamical decomposition arises in control system design since the controlled objects often constitute multivariable systems in which there may be dynamic couplings between several actuators (subsystems). Further, the decomposition of centralized into decentralized functions is essential for distributed systems. It is interesting to note that these decompositions tend to coincide in machine control applications. I.e. for example a position servo function is potentially identifiable based on all three ways of decomposition.

**Decentralization: Application vs. resource structure and execution strategy considerations.** In this step, application characteristics and the above mentioned aims are used as a basis for identifying natural partitioning and allocation approaches (in view of considered resource structures). Different approaches are evaluated with respect to resource utilization, extensibility and cost.

Execution strategy considerations are used to ensure that timing requirements of control activities can be met and for comparing remaining alternatives. That is, scheduling considerations are integrated in this step. Partitioning and allocation derive *distribution units* that contain *executable processes* that are allocated to the computer system.

Locality based distribution has been shown to be a good approach towards reduced control delays, load balancing and modularity. Locality based distribution however comes in several flavors. For example, several allocations can qualify as locality based and the resource structure may require functions local with respect to for example one joint to be implemented in several nodes. Therefore, heuristics from general purpose real–time allocation problems are also used, including considering the weights of bounds/communication (some may be neglected) and the grouping of strongly connected functions into objects to hide internal communication.

Given a number of potential hardware and application structures, one basic idea is to apply two levels of distribution: Low level and locality based distribution. This is very useful as a first attempt since they stress different potential bottlenecks. The former approach stresses communication and centralized controller computation requirements, whereas the latter approach to a more or lesser extent reduce communication but increase computational (and memory) requirements on I/O nodes. Necessary constraints on utilization are evaluated. If both approaches satisfy necessary constraints and leave a sufficient utilization margin, decentralization should be straightforward.

If both approaches fail in the evaluation, several possibilities are at hand. Application optimization and implementation can be altered and enhanced, e.g. through algorithm optimization and further decomposition. Alternatively, increased hardware performance can be considered. Subsequent iteration failures to meet basic constraints may require iteration back to change system requirements (if possible), for example, by reducing a particular sampling rate. Remaining candidate allocations can be compared with respect to control delays, extensibility and cost.

A set of selected policies with respect to triggering, scheduling and synchronization are together referred to as an *execution strategy.* An implemented control activity may involve the execution of sequences of elementary functions in several nodes. If full control of triggering, scheduling and synchronization policies is not possible or not properly considered, undesirable time–variations may result. Consider the execution of a pair of elementary functions that need to exchange data. Execution scenarios with respect to triggering, scheduling and synchronization are as follows:

– The functions are either time– or event–triggered. These two principles are well known computer system principles, e.g. to detect changes in the environment either by *polling* (sampling) or by *events* (e.g. external interrupts).

– The time–triggering is based on a local or a global time base.

– An event–triggered function is either truly aperiodic or sporadic. The former type is always enabled for execution once a relevant event occurs. The latter is only enabled provided that a specified minimum period has expired since the last event occurred. I.e., for sporadic activities the maximum event occurrence frequency must be bounded, Mok (1983).

– The system is completely statically scheduled, completely scheduled based on static priorities, or scheduled using a mix of these and possibly other policies.

In distributed application design, this altogether provides many different execution scenarios for control activities and cooperating processes. For example, a distributed producer/consumer (compare sensor, controller and actuator) pair could be implemented as two periodic processes with a constant phase relation (e.g. by use of a global clock) in order to achieve a constant control delay, or as a periodic – aperiodic/sporadic pair. In either cases the network may be statically or run–time scheduled. Based on a model specifying real–time requirements and implying a desired execution strategy, execution strategy considerations (”transformations”) can be used in design situations where the system hardware architecture and resource management policies may be more or less fixed, to improve the timing behaviour of the distributed application (e.g. reduce time–variations).

## 5. The use of CAN as a basis for distributed real–time control systems

**Establishing a global clock.** The provision of a global clock as a fundamental basis of a distributed control system drastically facilitates all aspects of system design, implementation and management. As described by Kopetz and Grundsteidl (1994), a sparse time base can be created on top of local synchronized clocks, providing a time grid with sufficient granularity, relative accuracy and precision for the application. External synchronization, if required, can be included in the clock synchronization. There exists many proposals and implementations for master/slave clock synchronization. Distributed clock synchronization should however be used to increase reliability and to eliminate the need to consider a faulty clock master. One example of a protocol with distributed clock synchronization (and based on static scheduling) is TTP. A global clock is not specified by CAN and FIP, but both provide a base for implementing clock synchronization. For priority based scheduling, a distributed solution may well exist but has not been found in the literature.

**Scheduling.** As described in section 3, several scheduling policies can principally be used, however, requiring specific considerations due to the fixed error handling policy. Priority based scheduling requires priorities to be assigned to messages and updating principle to be determined. The use of a rate (or deadline) monotonic style of assignment is considered to be straightforward, see Tindell and Burns (1994). The separation of message priorities from message contents is important. As the urgency of a particular data–item clearly can vary from application to application, a standard imposes a strong limitation if the standard prescribes the object numbers, i.e. priorities, to be used for specific data entities such as e.g. pressure or temperature.

**Dimensioning.** Communication system dimensioning in terms of utilization and real–time requirements can be based on the rate monotonic framework. For coarse dimensioning the utilization contributions from nodes can be computed. For detailed dimensioning a critical instant analysis, i.e. the worst–case instant where the maximum number of messages are transmitted simultaneously, is useful.

**Periodic vs. event controlled updating and datagram vs. acknowledged services.** There is much to be said about this topic. It is important to base these decisions on application (message) requirements with respect to consistency and real–time behaviour, e.g. to distinguish between continuous and discrete data–flows and to identify the consequences of data–loss and/or non consistent updating, see further Rodd et al (1990), Kopetz and Grundsteidl (1994), Törngren (1995). CAN guarantees with high probability that the network delivers a message, or alternatively signals an error. This then requires consideration of the ”weak acknowledge” together with the fact that high level acknowledgement is not necessary for continuous data when periodic updating is used.

## 6. Conclusions

The paper has provided several perspectives towards the design of distributed real–time control systems and indicated some important design trade–offs and problems. The need for design and modelling tools which can be used early in design and which specifically address decentralization has been emphasized. The choice of application decentralization level and execution strategy largely determines the timing behaviour of a distributed application. This is an important reason to incorporate the discussed design aspects into system design strategies and standards.

Both CAN and FIP have good real–time properties compared to many other protocols. The major drawbacks of CAN include the fixed error handling policy and that the maximum cable length is limited. The major drawbacks of FIP include the single point of failure due to centralized control, the less stringent error detection mechanisms and the relatively expensive components. For cost–critical applications, therefore, CAN is a natural choice. To eliminate the disadvantage regarding error handling, a CAN chip where the retransmission handling can be turned off or enabled independently for messages would be desirable.

To use CAN as a basis for distributed control systems, clock synchronization should be implemented (preferably in hardware together with the communication controller). Decentralized clock synchronization using priority based scheduling is a topic for further work. Specific attention should be given towards choice of scheduling policy, the fixed error handling policy, the choice of communication model and the fact that CAN acknowledgement does not state that a message has been received.

## References


Audsley N.C., Burns A., Davis R.I., Tindell K.W., Wellings A.J. (1995). Fixed priority pre–emptive scheduling. *J. Real–Time Systems,* 8, 173–198. Kluwer. ISSN 0922–6443.

Charzinski J. (1994). Performance of the error detection mechanisms in CAN, *Proceedings of the first Int. CAN Conference.* Published by CAN in Automation.

CiA (1994). *Proceedings of the first Int. CAN Conference.* Published by CAN in Automation.

Duffie N.A. (1982). An approach to the design of distributed machinery control systems. *IEEE Trans. on Industry Applications*, Vol. IA–18, No. 4, July/August 1982.

Enslow P.H. (1978). What is a distributed system. *Computer*, January 1978, pp. 13–21.

Fox M.S. (1981). An organizational view of distributed systems. *IEEE Trans. on systems, man, and cybernetics.* Vol. SMC–11, No. 1, Jan. 1981.

Garlan D., Shaw M. (1993). An introduction to software architecture. *Advances of software engineering and knowledge engineering.* Vol. 1, World scientific publishing company.

Guth R., Lalive d'Epinay T. (1983). The Distributed Data Flow Aspect of Industrial Computer Systems, *Proc. of 5:th IFAC workshop on distributed computer control systems*, Sabi–Sabi, South Africa.

Jensen E.D., Boebert W.E. (1976). Partitioning and Assignment of Distributed Processing Software. *IEEE COMPCON* fall 1976, pp. 490–506.

Jensen E.D. (1981). Decentralized Control, *Distributed Systems: An Advanced Course*. Springer Verlag.

Kiencke (1994). Controller Area Network – From Concept to Reality. *In Proc. of the first Int. CAN Conference.* Published by CAN in Automation.

Kopetz H. (1983). Real time in distributed real time systems. *Proc. of the 5:th IFAC Workshop on Distributed Computer Control Systems*, Sabi–Sabi, South Africa, 1983.

Kopetz H., G. Grundsteidl, (1994). TTP – A Time Triggered Protocol for Automotive Applications, *IEEE Computer,* January.

Laplante P.A. (1995). Guest editor introduction. *J. Real–Time Systems,* 8. Kluwer.

Leterrier P. (1992). The FIP protocol, Centre de Competence FIP, Nancy, France, 1992.

Liu, C.L, Layland, J.W. (1973). Scheduling Algorithms for multiprogramming in a hard–real–time environment. *J. of the Association for computing machinery*, 20, 46–61.

Mok A. (1983). *Fundamental Design Problems of Distributed Systems for the Hard Real–Time Environment.* PhD thesis, Massachusetts Inst. of Technology, U.S.A.

Motus and Rodd (1994). *Timing analysis of real–time software.* Pergamon, ISBN 0 08 0420257.

Ramamritham K. (1987). Channel Characteristics in Local–Area Hard Real–Time Systems, *Computer Networks and ISDN Systems,* 3–13, North–Holland, Amsterdam.

Ramseyer R.R., Arnold R.G., Applewhite H.L., Berg R.O. (1979). The modular missile borne architecture from the requirements and constraints point of view: An overview. *IEEE Proc. 1st Conf. on Distributed Computing Systems*. Oct. 1979. pp. 747–756.

Rodd M. G., Zhao G. F., Izikowitz I. (1990). An OSI–Based Real–Time Messaging System, *Journal of Real–Time systems*, 2, 213–234, 1990 Kluwer Academic Publishers.

Stallings W. (1995). *Data and computer communications.* Prentice–Hall. ISBN 0–13–326828–4.

Tannenbaum A.S. (1995). *Distributed operating systems.* Prentice Hall. ISBN 0–13–143934–0.

Tindell K. and Burns A. (1994). Guaranteeing Message Latencies on CAN. *In Proc. of the first Int. CAN Conference.* Published by CAN in Automation.

Törngren M. (1995). *Modelling and design of distributed real–time control applications.* PhD thesis, Dept. of Machine Design, The Royal Institute of Technology, Stockholm, Sweden.

Törngren M., Backman, U. (1993). Evaluation of Real–Time Communication Systems for Machine Control*. Proc. of the Swedish National Association on Real–Time Systems Conference*, August 1993. Royal Institute of Technology, Stockholm, Sweden.

Wolff T.O. (1977). Improvements in real time distributed control. *IEEE COMPCON* fall 1977, pp. 409a–409g.