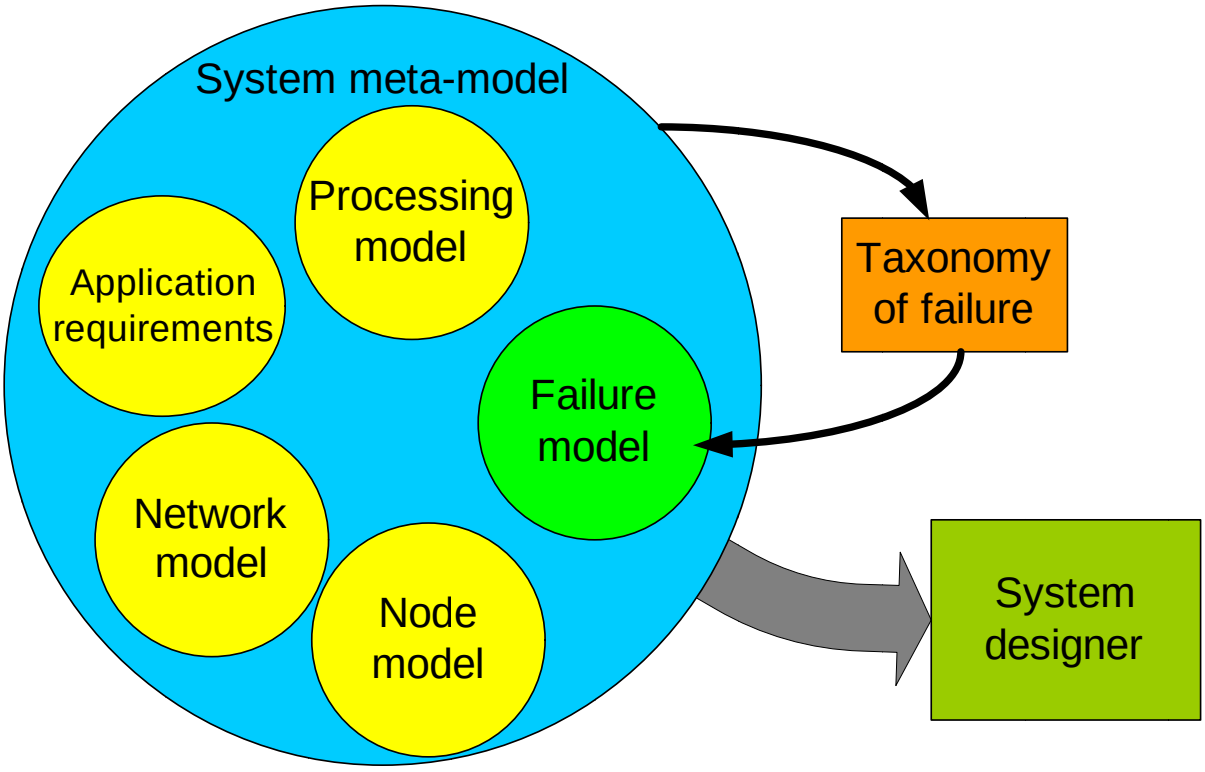
A System Model for Distributed Real-time Systems

Carl Bergenhem, Johan Karlsson

SP Technical Research Institute of Sweden



Electronics

SP Report 2012:31

A System Model for Distributed Real-time Systems

Carl Bergenhem, Johan Karlsson

Abstract

A System Model for Distributed Real-time Systems

Svenska: Denna rapport presenterar en systemmodell för en typ av distribuerade realtidssystem. Målet är att underlätta utformningen av protokoll för feltolerans som membership agreement. Systemmodellen innehåller också en felmodell som beskriver de fel som rimligen kan uppstå i systemet. Dessa fel kan sedan hanteras av feltoleransprotokollet. Rapporten bidrar med en taxonomi med vilken fel kan beskrivas. Den resulterande felmodellen påverkas av de andra delarna i systemet. Typen av system väljs beroende på kraven i säkerhetskritiska tillämpningar såsom x-by-wire. Processningsmodellen består av CDR-operationer (Beräkna och distribuera Resultat) och innebär att en producent-process beräknar ett resultat och distribuerar resultatet via kommunikation till konsument-processer. Fel i CDR-operationer beskrivs med fyra aspekter: typ, symmetri, detekterbarhet och tidsegenskap, beroende på vilket system som komponent som är felaktig. Vi jämför våra definitioner av fel-typer med kommunikationsfel enligt IEC 61784-3.

Engelska: This report presents a system model for a class of distributed real-time systems. The goal is to assist the design of fault tolerance protocols such as membership agreement. The system model contains a failure model that describes the failures that can plausibly occur in the system. These failures can then be addressed by the fault tolerance protocol. The report contributes with a taxonomy by which failures can be described. The resulting failure model is affected by the model of the system and by the model of how processing is done in the system. The class of system is assumed to be strongly partitioned which provides a high degree of error containment for real-time processes executed in the same node and also for processes executed on different nodes. The smallest unit of failure is therefore the process. The system model uses a broadcast communication network similar to Flexray, i.e. it supports both time-triggered and event-triggered communication. The class of system is chosen based on the requirements of safety-critical applications such as x-by-wire. The processing model for the system is presented in which operation is divided into sequentially executed primitive operations, called CDR-operations (Compute and Distribute Result operations). A CDR-operation involves a producer process which computes a result and distributes the result via broadcast communication to consumer processes. Failures of CDR-operations are characterised by four aspects: type, symmetry, detectability and persistence; depending on which system component that is faulty. We compare our definitions of failure types with communication errors according to IEC 61784-3.

Key words: System model, processing model, node model, network model, failure model, protocols for redundancy management, safety-critical distributed real-time systems, IEC 61784

SP Sveriges Tekniska Forskningsinstitut
SP Technical Research Institute of Sweden

SP Report 2012:31
ISBN 978-91-87017-45-2
ISSN 0284-5172
Borås 2012

Contents

Preface

Work with this report was done during my PhD-studies 2005-2011 together with Professor Johan Karlsson which were during my employment at SP. The report is also available as a Technical report from the Chalmers Library at CPL 2008:18

1 Introduction

Designing distributed real-time systems to function in the presence of faults is challenging due to the complexity of the task. Such a system is composed of many interacting components. An activated fault, either systematic or random, can lead to failure of the affected component and if unhandled can propagate and lead to failure of the entire system. The role of each component must be understood and be clearly stated by the designer. This includes finding a failure model that describes what failures may occur in the system. When the failure model is formulated the failures can be dealt with by proper design of the application or introduction of a redundancy management protocol. To support work in this area we propose a model of a distributed real-time system and pay particular attention to the failure model of the system. Focus is on applications and especially protocols which implement fault tolerance and redundancy management such as a membership agreement protocol.

A system model includes several sub-models which describe the node, network, processing model and also failures in the system. The system model in the report describes a class of distributed real-time system. This class of systems consists of a fixed number of computing nodes, possibly with actuators and sensors, which are interconnected with a network. The node and network are described in respective models. Each node includes a CPU, which executes real-time processes, and a run-time environment. In each node there are incoming and outgoing communication links. It is assumed that the addressed class of systems use a broadcast network (similar to Flexray) which provides both time-triggered messages and event-triggered messages between nodes. The run-time environment in each node consists of a real-time operating system and various system services such as that provided by a redundancy-management protocol such as membership agreement.

The system is assumed to be strongly partitioned. This implies that it provides a high degree of error containment between real-time processes executed in the same node (intra-node error containment), and between processes executed on different nodes (inter-node error containment). The failure model hence considers process failures (the smallest unit of failure due to error containment between processes), node failures, failures of incoming and outgoing communication links, as well as failures of the network. This system model is chosen based on the assumed requirements of a safety-critical application; such as a by-wire function.

Failures are described according to a taxonomy that is presented. The taxonomy describes failures in a structured manner according to four aspects: type, symmetry, detectability and persistence. The taxonomy of failures is independent of the system model, i.e. can be used to describe failures in different systems. The failure model is a result of the effects of the particular system model (including node model, network model etc.). If a different system model is regarded, e.g. due to other requirements from the application, another corresponding failure model will result. We compare our definitions of failure types with communication errors according to IEC 61784-3 – Industrial communication networks Part 3: Functional safety fieldbuses [2].

1.1 Organisation

In the following text **bold** characters are used to emphasise the introduction of a new concept. The report is organized as follows:

- A system model and basic assumptions of the system are presented in Section 2.
- A taxonomy by which failures are described is given in Section 3.
- A novel processing model is presented in Section 4.
- The failure model is presented in Section 5.
- A comparison of communication errors according to IEC 61784-3 and our definitions is given in Section 6.
- A survey of failure models and system models that are used for other redundancy-management and fault-tolerance protocols is given in Section 7.
- Conclusions are drawn in Section 8 and finally a short discussion is given on the potentially wider applicability of the system meta-model to describe other systems in Section 9.

2 System Model and Basic Assumptions

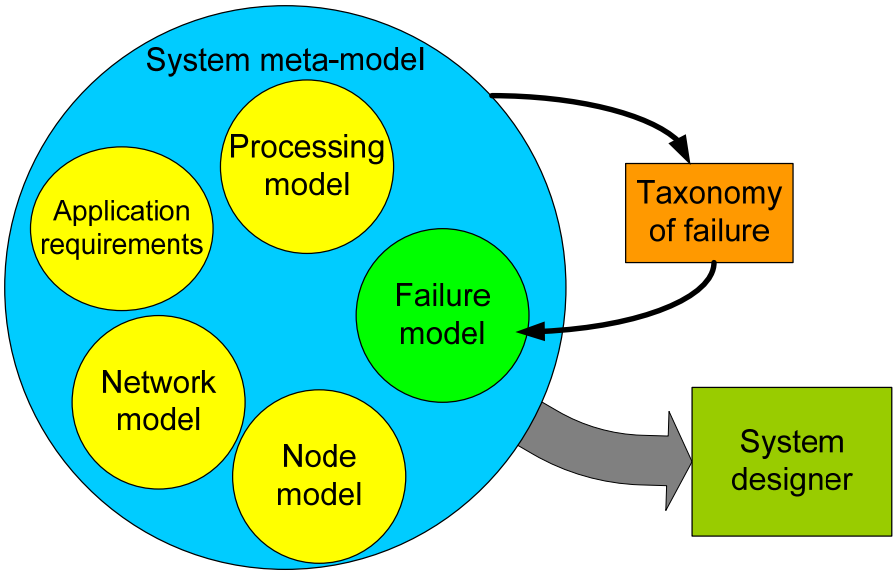


Figure 1: System meta-model and taxonomy of failure

An overview of the work in this report is given in Figure 1 in the form of a meta-model. A meta- model defines what an instantiated model will contain and how it is described. In this case instantiation takes place when the meta-model is applied to describe our class of strongly partitioned systems. This system model and its sub-models are described during course of the following sections.

The system model contains a network model, a node model, a processing model etc., depending on the nature of the application. Each of these sub-models describes the respective parts of the system model in further detail. The processing model describes how the system performs its work. There may also be requirements on the system that stem from the application and the environment that the system will work in. Three of the sub-models (the node model, the network model and the processing model) are presented here to describe the properties of the system.

A failure model is part of the system model and it describes each failure mode that is plausible in the system. Each failure mode in the failure model is described according to the taxonomy of failure that is provided in this report. The failure model is affected by the system model (node, network etc.). Each of the failures in the failure model can be handled means such as fault tolerance to achieve sufficient dependability of the system [3]. The failure model is presented in Section 0.

We assume that the target application for the system is safety-critical or requires high reliability. This implies that malfunction of the system may result in serious harm to life or cause financial loss. Typical applications which are performed by the system include by-wire steering and braking, stability control and collision mitigation (in ground vehicle system) and flight-control (in avionic systems). These functions are safety-critical (especially by-wire functions) in that malfunction may critically affect e.g. the dynamics and controllability of the vehicle and thereby lead to loss of life or damage to equipment etc. The application is assumed to be distributed; implying geographically dispersed presence of nodes that control sensors and actuators and processing functions. Processing is decentralised in that there is no central node that has authority over all parts of the system. Finally the application requires that the performance system in the presence of faults must be known. This can imply that the system is designed to detect and tolerate a

finite number of faults or that service from a faulty system degrades in a controllable manner. The system can also feature redundant components. To support the safety-critical application, protocols for fault-tolerance such as redundancy-management protocols are used. Requirements on the protocols are assumed to match the requirements on the function, hence the protocols are also safety-critical and distributed.

2.1 System Architecture and Node Model

The node and network models together with the application software give a view of the system and is denoted the system architecture, see Figure 2. The application software is described at a simplified level with its processes. We assume that the system consists of a fixed number of nodes (*N*) which are interconnected via a broadcast network. In this figure the architecture of the system can be seen. Each node includes a CPU (single or multi-core), main memory, a communication controller for the broadcast network and optionally I/O interfaces for sensors, actuators and other peripheral devices. The communication controller is assumed to support clock synchronisation to offer a globally consistent notion of time at each node. The network can optionally feature multiple redundant links to increase dependability and/or performance, depending on configuration. As an example the Flexray communication network includes support for dual network links. Multiple link networks are not further investigated. For simplicity we also assume that the network does not connect to other networks, e.g. via a gateway.

A simplified node model is described which is divided into software and hardware **components**. A node consists of the following hardware components: a **processor**, which consists of the CPU, memory and optional I/O, an **incoming link** (**IL**) and an **outgoing link** (**OL**). The incoming and outgoing links represent the input and output portion of the communication controller including the buffer memory in which the process reads and writes messages. We assume that the outgoing link has a **bus guardian** (**BG**). Bus guardians prevent faulty nodes (sometimes called “babbling idiot” nodes) from interfering or disturbing the network traffic produced by other nodes, blocking failure. Bus-guardian techniques for time-triggered communication are investigated in [4] for a local guardian approach for networks with bus topology and in [5] for a centralised guardian approach for networks with star topology. A technique for use of bus-guardians with event-triggered communication is investigated in [6]. Bus-guardian techniques for time-triggered communication are to the author’s knowledge currently more well-established than for event-triggered communication. Components for the former exist for commercially available networks such as TTP/C [7]. Therefore we assume that the strongly partitioned system provides bus guardian protection only for the time-triggered portion of the communication.

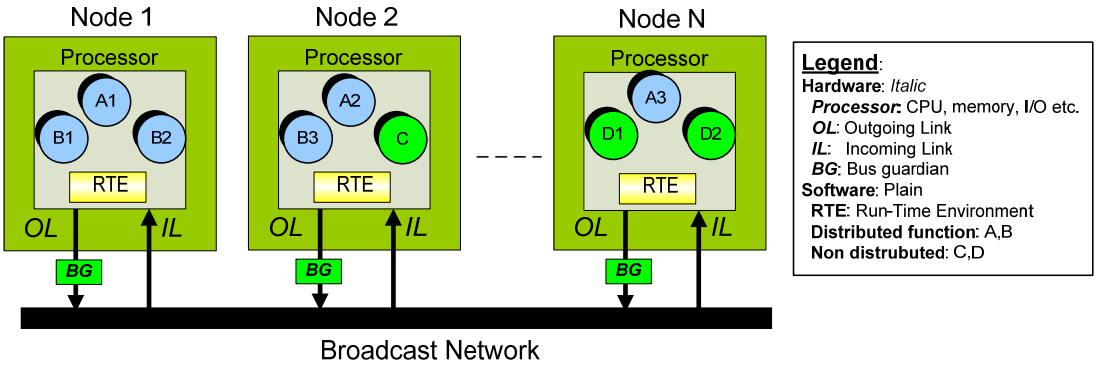


Figure 2: System architecture. Nodes, network and processes can be identified. Hardware is denoted in italic, while software in plain text.

Software components in a node are divided into: a **run-time environment** (**RTE**) and multiple **real-time processes**. Cooperating real-time processes executing on at least two different nodes constitute a **distributed real-time function**, such as A and B in Figure 2. Function C and D are not distributed, although D consists of two processes. The run-time environment[1] consists of an operating system, device drivers and a set of system services. In particular, the run-time environment includes a **sender service** and a **receiver service** which handles all communication of a node. We assume that the RTE in nodes have sufficient knowledge of the allocation of processes to nodes. Local time at a node is synchronised (with a clock synchronisation algorithm) to the other nodes in the system.

2.2 Network Model

The network model presented in this section is aimed to broadly resemble a class of communication networks similar to FlexRay [8] albeit with simplifications. An example of another suitable communication networks for this setting is TTCAN [9].

The communication network model supports both time-triggered and event-triggered broadcast messages. Broadcast implies that a sent message is normally received by all nodes except the sender; hence there are normally *N-1* receivers. In Flexray a sending node cannot form an opinion on the status of its own message, e.g. whether the message that it sent was disturbed by network disturbance or not. In CAN the sending nodes monitor that the intended bit of the message being sent is the same as bit read from the network, hence it is possible for the sender to form an opinion of the sent message status [10].

The reception-time of messages is noted in the communication controller and is used by the clock-synchronisation algorithm to maintain synchrony of communication controllers and nodes. Communication in the network is organised into periodically repeated **cycles** of equal length and schedule. A communication cycle consists of a static and a dynamic segment and a period of network idle time, see Figure 3. Note that the scheme in the figure for the dynamic segment is simplified compared to a real network, such as Flexray.

The static segment contains a fixed time-triggered schedule of **slots** of fixed length for static message. Messages have a fixed length and are sent at known times. Each slot is associated with a specific process in a node. All nodes have sufficient knowledge of the communication schedule and are configured in a consistent way. To protect against corruption and aide detection of corrupted message, a CRC which covers both payload and header/trailer is included in the message. Despite this measure corruption may still be undetectable, especially if the sending node is the source of the fault.

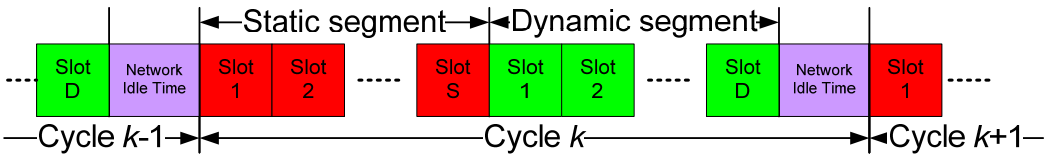


Figure 3: Communication cycles with both event- and time-triggered communication. Cycle *k* is shown in its entirety

[1] Our definition of RTE has a different scope than the RTE defined by AUTOSAR .

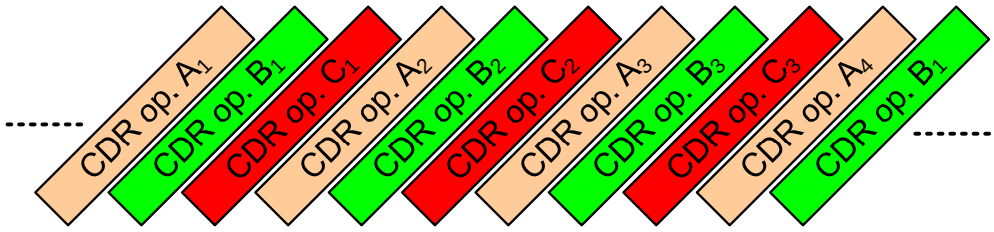


Figure 4: Consecutive CDR operations. Each independent chain of operations

Communication during the dynamic segment is contention based according to the priority of each message. The priority of a message is fixed during design-time. Due to the uncertainties in the event-triggered mode of medium access, the jitter of the delay of dynamic messages is greater than static messages. The size of the jitter depends on the chosen network and its configuration. The timeliness of each dynamic message can be configured by appropriate allocation of its priority in relation to the priority of the other dynamic messages. Dynamic messages are configured to be either guaranteed to be sent (albeit with jitter) or are "best-effort", i.e. only being sent if there is unused capacity. In the figure, the dynamic segment contains slots 1 to *D* to illustrate that *D* dynamic messages are guaranteed to being sent. We assume that critical dynamic messages can be given high priority over other messages and hence are guaranteed to be sent during a single communication cycle. The dynamic segment of the network model resemble the medium access scheme in Byteflight [11].

2.3 Processing Model

We assume that the operation of distributed functions in the proposed class of system can be divided into sequentially executed primitive operations called **CDR-operations**, where CDR means Compute and Distribute Result[2]. Such an operation involves the computation of a result by a **producer process** (denoted producer) and the distribution of that result, via broadcast communication as payload in a message[3], to one or more **consumer processes** (each denoted consumer). For simplicity (but not necessity) we assume that a message only contains one result. A message is sent by the sender service in **sender** node and received by the receiver service in **receiver** nodes. The receiver service takes the payload, the result, from the message and passes it to consumers within the node. Each node can have zero or more consumers. In the case of a system with broadcast communication, a message is received by all nodes under failure free conditions. Other components involved in the distribution of the message are the outgoing link, the network and incoming link, see Figure 5. The software and hardware components in the figure are represented as circular and rectangular shapes respectively. The message in the CDR-operation is sent in a communication slot that is dedicated to the particular producer process. The computation that is performed in the producer and consumer node does not necessarily occur within the time-bounds of the communication slot in which the message is sent. The distribution of results in the CDR-operation can take place with either time-triggered or event-triggered messages.

A single CDR-operation is to be perceived as an elementary service provided by the distributed system. Elementary implies that all other services from the system are based on this concept. During a CDR-operation all involved system components must be fault-

[2] In an automotive context, result can be referred to as a signal.

[3] In an automotive context, message can be referred to as a frame.

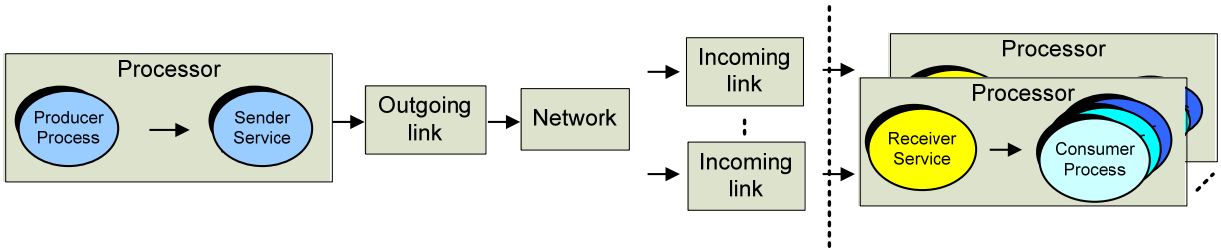


Figure 5: One compute and distribute result (CDR) operation from producer to consumer.

free for the system to provide a successful (CDR-operation) service. A **CDR-failure** is therefore a failure to successfully perform the CDR-operation, and is considered to be an internal service failure; i.e. that the CDR-failure does not propagate outside the system bounds. A CDR-failure implies that one or more consumer processes do not receive the result correctly due to a fault in a component that is involved in the operation. Fault-tolerant distributed systems are designed to mask the effects of CDR-operation failures. Their ability to do so is determined by the failure modes of the CDR-operations. A CDR-failure can be caused by a fault affecting any software or hardware component that is used in the CDR-operation. The fault will lead to an error and eventually lead to failure at system level if left unhandled. Some examples of faults that affect the system and potentially cause a failure are a bit-flip inside a sender or receiver node, crash of a producer process or a burst of external disturbance that affects the network. In the following text a failure refers to CDR-failure and operation refers to CDR-operation unless stated otherwise.

A system typically offers several services, where each is realised by a chain of individual CDR-operations. The chain has different lengths depending on the realised service. An example of consecutive CDR-operations is depicted in Figure 4. Each CDR-operation is staggered, i.e. execution and communication overlaps in time with the previous and next operations. In this example there are three independent chains of CDR-operations, A, B and C, where each realise a service. CDR-operations within a chain are dependent on previous operations unless it is the initial operation in a chain. Operation A_3 depends on operation A_2 which in turn depends on operation A_1 (the initial operation). A failure of a single CDR-operation will affect following dependent CDR-operations. Other operations are unaffected and hence continue to deliver correct service unless the fault has a major effect on the system such as a critical fault in the CPU in a node. Processing of consecutive CDR-operations is not constrained by communication cycles, but may overlap several cycles.

Propagation of a CDR-failure to following related CDR-operations is depicted in Figure 6. The operations in the figure are depicted as being not staggered or overlapped for clarity. A fault in operation Z_1 (indicated with a bold X) causes a **primary failure**, i.e. a failure which is caused directly by the original fault. The failure remains dormant in the

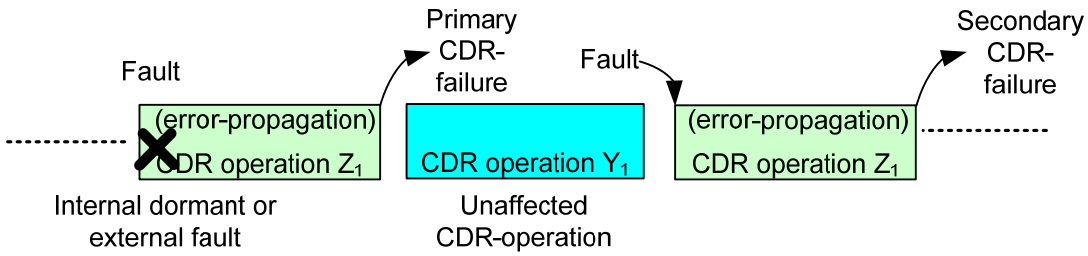


Figure 6: CDR-failure leads to partial system failure of service Z

system until it is activated by the execution of CDR-operation Z_2. Activation leads to a **secondary failure**, i.e. a failure which is caused by failure of a previous CDR-operation. In the example Z_2 depends on Z_1. **Tertiary failure** and so on, can result unless the failure is detected and the dependency chain is broken by e.g. detection of the failure and restarting service Z at operation Z_1. In the example, CDR-operation Y_1 is unaffected and continues to correctly contribute to service Y. A failure of a single chain of CDR-operations only affects a single service of the system – **partial system failure**. If all CDR-operations are affected due to a major failure, then all service of the system is disrupted – **complete system failure**. A failure of a specific resource in a node, such as dedicated hardware, can cause failure of multiple CDR-operations that share the resource. In the example a partial system failure has occurred in service Z, realised by CDR-operations Z_1, Z_2, etc. Scenarios of more complex dependencies between CDR-operations and the effects of a single fault are probable in a real system.

An advantage of modelling system operation in terms of CDR-operations is that details in the path from producer to consumer are abstracted away. This simplification removes the need to keep track of nodes and propagation of errors such as an erroneous result. For example, a process in a node can act as consumer in one CDR-operation and then act as producer in the next CDR-operation. Individual component faults, such as faults in incoming or outgoing links, do not have to be handled separately, but are rather handled as a failure of an entire CDR-operation. In the next section we present a detailed categorisation of CDR-failures based on the characteristics of failure that are described in Section 2. A CDR-failure will have different characteristics depending on the cause of the failure.

3 Taxonomy of Failures

In this section a taxonomy of failures is presented. It is aimed to be applicable to the class of strongly partitioned distributed real-time systems. Failures are characterised according to four main aspects: **type**, **symmetry, detectability** and **persistence**.

The type of the failure is further distinguished into:

- **Value failure.** Corruption in the payload (data value) of a message.
- **Timing failure.** The delivery time of an expected message deviates from the specification, i.e. the message is either early or late.
- **Omission failure.** The consumer does not receive a result or message.
- **Signalled failure**. The affected component in the system is unable to perform its service and instead sends an error message.
- **Blocking failure.** A node jams other traffic on the network, e.g. by sending too many messages, untimely messages or disturbance.
- **Erratic failure.** Delivered service at a node is not according to specification. Erratic failure implies timing and value failure or a combination of these.

- **Addressing failure.** Corruption in the message that affects source or destination address, e.g. message masquerading.
- **Insertion failure.** A spurious unexpected message is received, e.g. message comission failure.
- **Repetition failure.** An old result or message is repeated.
- **Sequence failure.** Messages are received in the wrong sequence.

The first seven failure types are generally applicable to all type of communication systems. The last four failure types are believed by the authors to be mainly applicable to event-triggered communication systems, and less to time-triggered communication systems. The applicability of failure types in relation to the system model is discussed in Section 5.

The symmetry aspect of a failure decides whether nodes in the system perceive the failure in the same way or not. Note that the symmetry aspect is defined according to an “oracles viewpoint”, i.e. we make the assumption that the status of the CDR-operation at the nodes can be instantly known to the oracle. This aspect is only meaningful if there are two or more nodes in the system. Perception of the failure is thus either **symmetric** or **asymmetric**. A symmetric failure is perceived identically by all non-faulty nodes, e.g. a message is lost at all nodes. An asymmetric failure occurs when nodes have different perceptions of the failure, e.g. a message is lost at some but not all nodes. Depending on how the send and receive operation is defined, the sender may or may not also normally receive its own message, i.e. the sender is also part of the group of receiver. For example, in a system of N nodes, where the sender receives its own message, there will be one sender and N receivers. Hence the sender is included in the definition of symmetry. However, in this report we assume that sender nodes cannot and therefore do not listen to their own messages, hence the number of receivers is generally $N-1$.

The symmetry aspect is related to the notion of **consistency**. This is attribute is used to describe the relevant internal states of the nodes in the system, i.e. an attribute that has a larger implication than only the taxonomy of CDR-operations. As with the symmetry aspect, consistency is defined according to an “oracles viewpoint”. The consistency of a

system state is either **consistent** - identical internal states; **inconsistent** – differing states. A symmetric failure normally leads to a consistent system state. That is all nodes have perceived the failure in the same implying that the relevant system states in the nodes are consistent and thus nodes proceed identically. Note that this implies that the system state in the nodes may be erroneous, i.e. other than intended, although its is consistent. An asymmetric failure can potentially lead to inconsistent system state. However, inconsistent system state can be avoided if the system utilises mitigating protocols such as atomic broadcast to handle the failure. (This protocol ensures that a message is delivered correctly to all or none of the receivers.) We avoid the notion of consistency since it describes the internal states of nodes, i.e. consistency is the consequence of a failure, rather than an aspect of a failure.

Detectability decides whether the failure is detectable with **individual assessment** by an individual node e.g. by the consumer process or receiving service in the node. The failure is thus either **detectable** or **undetectable**. An undetectable failure implies that individual assessment based only on an incoming message in a single CDR-operation does not suffice to resolve the failure. The detectability aspect of a failure is referred to as "I-detectability", i.e. detectable by individual assessment, in work by Torin et al. in the NFFP-project report [12].

Persistence specifies the timing characteristic of a failure; the failure is either **temporary** – a single CDR-operation is affected; or **permanent** – consecutive CDR-operations are affected. A permanent failure of CDR-operation can, for example, be caused by a crashed producer process. The producer process will generate an incorrect or no result every time until repair and therefore imply failure of all CDR-operations involving the particular producer process.

The taxonomy of failures given above is based on work by Avizienis et al. [3] but detail has been added and it is adapted to be applicable for processing in distributed real-time systems and to assist the design of such systems and protocols for fault-tolerance.

The aspects of CDR-failures are summarised in Figure 7. A failure is characterised by combinations of the symmetry, detectability and persistence aspects and type. When the taxonomy of a failure is stated, an omitted aspect implies that it is not defined. For example, in "Detectable Temporary Value Failure" the symmetry aspect is not specified.

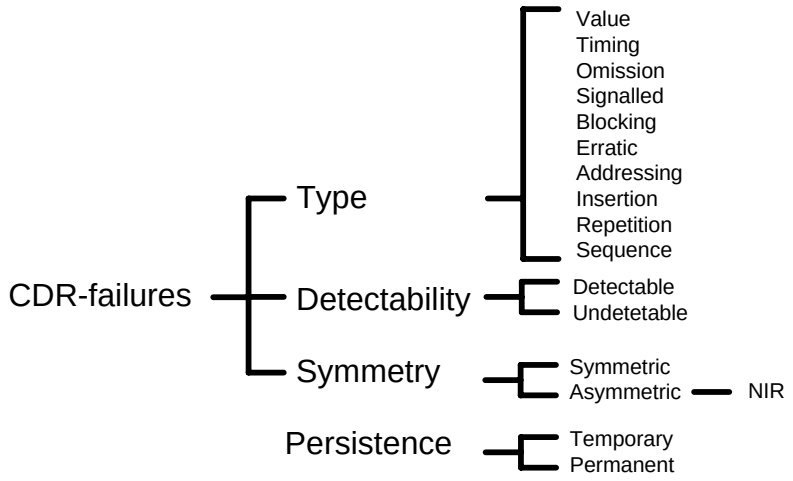


Figure 7: Overview of the aspects of CDR-failures

3.1 Degree of asymmetry – Number of incorrect results

In section 0 the aspect of asymmetry was introduced. This aspect is now further detailed to describe the asymmetry of a failure in terms of degree of asymmetry. The degree of asymmetry is expressed by the **number of incorrect results** (**NIR**) that are received as part of one CDR-operation. By incorrect we imply a deviation from the intended; the correctly computed result from the producer process being available at consumer processes in receiving nodes. An incorrect result can for example be due to no message at all being received at a node hence no result is available (e.g. due to receive omission failure); or a corruption of the result (e.g. due to a value failure). The degree of asymmetry concept is used to describe the effects of a fault as a function of the NIR. The NIR is expressed as a natural number between 0 and *R*; where *R* is the number of nodes that receives the message containing the result. For the description of NIR we make an assumption that each node receives one result as part of the received message and that the result is then delivered internally potentially several consumer processes within the node. We assume that a distributed result is received by *N-1* nodes under fault-free conditions; i.e. all nodes in the system except the sender. Depending on the type of network actually being used, the sender may also receive its own message – hence in this case there would be N receivers.

A low NIR implies that a minority of received results are incorrect at nodes, i.e. a minority of nodes are affected by the failure. A high NIR implies that a majority of received results are incorrect at nodes, i.e. a minority of nodes are affected by the failure. A NIR of 0 implies no error at any receivers, i.e. no failure. A NIR of *R* implies error at all received results. Both NIR of 0 and R imply symmetric perception at the group of receivers. Interesting degrees of asymmetry are summarised in

Table 1. A NIR of *R/2* implies that there is an equal number of receivers that correctly received the result as receivers that did not. Failure with this degree of asymmetry can be challenging if e.g. receivers are expected to vote based on the result. This situation may cause a “hung vote” and must be resolved with care.

Table 1: Examples of different degrees of asymmetry

NIR	Comment
0	No error at any receiver, i.e. symmetric perception. No failure of CDR-operation.
1	Low degree of asymmetry. Error in one receiver. For example a fault in the incoming link of a receiver.
>0 – <R/2	Low degree of asymmetry. Error at minority of the receivers
R/2	Tie situation. Error at half of the receivers.
>R/2 – <R	High degree of asymmetry. Error at majority, but not all receivers
R	Error at all receivers, i.e. symmetric perception. All receivers are affected by the failure. For example a fault in the outgoing link of the sender.

4 Discussion on Failure Modes

This section is organised as follows. Firstly, limitations of the failure modes are discussed, i.e. failure modes that are considered irrelevant and these limitations are motivated. The plausibility of the various failure modes are then commented based on properties of the system model of the strongly partitioned class of systems. The discussion is presented from two perspectives: CDR-operations using time-triggered communication in the static segment and CDR-operations using event-triggered communication in the dynamic segment. Examples of related failure modes and other failure models are then given. Finally, a number of related and commonly used failure modes such as fail-silent and Byzantine failure are commented and compared with our taxonomy of failures and failure model.

4.1 Discussion on the plausibility of various failure modes

This section discusses failure modes that are not applicable to our system and processing model. Failure modes are commented and limitations in the failure model are briefly discussed and motivated. The communication network supports both event-triggered and time-triggered communication modes. These two modes manifest different failures due to faults and are therefore discussed separately where appropriate. Unless specifically stated, we assume that a failure that is plausible in time-triggered communication is also plausible in event-triggered communication, but not vice-versa.

A CDR-failure is caused by a fault in one of the components that are involved in the operation. Since an individual CDR-operation is a “single shot” the persistence aspect does not apply here. A fault is always modelled as a single occurrence during one CDR-operation and the fault leads to one CDR-failure. If the fault remains in the affected component, CDR-operations that are dependent on the component will also be affected. When consecutive CDR-operations are regarded it is however meaningful to apply the aspect of persistence – the persistence of a failure of the service provided by the consecutive CDR-operations. A **service failure** can be temporary or permanent. The former means that the service can be recovered, e.g. by using default data or reset of the service. The latter failure remains e.g. until repair is performed.

Sequence failure causes messages to arrive at a node out of order and can be a problem if sequential messages are required by the user. This can be a problem Similar reasoning as above for persistence is applied to sequence failure in a “single shot” CDR-operation. That is, for individual CDR-operation the concept of sequence of messages does not apply. However the concept of sequence is applicable for consecutive operations. Sequence failures can be detected by the addition of a sequence numbers in the message. This can be done by the RTE. An out-of-sequence message during a single CDR-operation can be detected and will not be accepted at the receiver node. A consumer process will perceive this as an omission failure, i.e. that the result is not available.

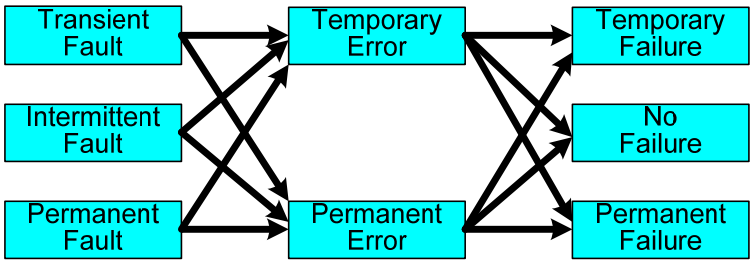


Figure 8: Activation of faults as a consequence of system activity. (Adapted from [1])

In general, the persistence of a fault can be either **transient** – a single occurrence of the fault, **intermittent** – recurring presence of the fault or **permanent** – presence of the fault is continuous. A transient or intermittent fault does not always lead to a temporary failure, and a permanent fault does not always lead to a permanent failure as may be initially concluded, see Figure 8. A permanent fault in a memory cell which is never used during the life-time of the system is an example of a fault which does not cause failure. The relation of persistence between faults and the failures that are caused is further investigated by Aidemark in [1] (page 10). A transient fault in a component that is part of a CDR-operation will only affect a single CDR-operation. Potentially, there will also be failure of dependent CDR-operations, e.g. operations where a faulty producer process is involved. A transient fault can therefore result in multiple CDR-failures. A permanent fault affects consecutive CDR-operations (depending on severity of the fault) until masked.

4.1.1 Time-triggered communication

Value failure implies that the data payload, e.g. the result, in the message is corrupted. Unless the failure is detected, e.g. by means of the message CRC, the payload will be received by the consumer and used during processing. A value failure is undetectable, for example, if the producer introduces the corruption before the CRC is computed or when the payload is "unprotected" when the CRC is recomputed along the way to the consumer. An example of this is a network with intermediate devices such as an active star in a star-network or the switch in a switched network. A value failure can also be detectable by means of the checksum on the payload and various other mechanisms that may be used. Examples are: plausibility checks and range checks on the data.

All network communication in our system model is broadcast directly from a source node to destination nodes without any intermediate stages or gateway devices etc. In this model asymmetric value failures are therefore avoided. Further, we assume that the probability for undetected failure is minimised self checking nodes and that messages are adequately protected with CRCs. An example of fault avoidance technique is to utilise end-to-end checksums [13] which protects messages on the entire communications path e.g. from producer to consumer. Other design strategies and the adequacy of CRCs are discussed by Paulitsch et al. in [14].

Time-triggered communication avoids timing failure since the communication controller will not normally accept and send an early or late message. Also it is normally not possible to have an execution schedule in a node with time-triggered processing which produces a result too early or late. Such a processing schedule is normally devised early in the design of the system and potential faults are found by checking with design-tools. In the event that a node, due to failure, attempts to send out of slot, the message will be prevented all together or truncated by the bus guardian, e.g. the message is intentionally corrupted by the bus guardian such that it is detectably corrupt at the receiving node. Timing failure is therefore disregarded for time-triggered messages in further discussions.

A send omission failure implies a failure of the sending node or producer to send a message. We assume that such a failure generally has a symmetric manifestation. Conversely, a receive omission failure, i.e. a failure of a receiving node or consumer to receive a result, generally has an asymmetric manifestation. We refer to both these failure types as an **omission failure**, i.e. loss of a result or message, and they are instead distinguished by the symmetry of the failure: symmetric or asymmetric.

Omission failures of time-triggered messages are assumed to always be detectable since all nodes have a common time-base and knowledge of when messages are expected. The detection latency is the length of the communication slot during which the message is expected. The assumption of detectability implies that a receiving node will always

correctly conclude that an expected but not received message is due to a failure. A signalled failure is assumed to always be detectable and also symmetric. Due to the assumptions made on omission and signalled failure, detectability is applicable only to value failures, i.e. a value failure is not necessarily detectable by the receiving node.

A potential cause of a blocking failure is interference from a faulty node especially when another node is transmitting. Blocking failure is sometimes known as **babbling-idiot failure** or **interference failure**. Examples of interference are: a valid message sent at the wrong time or simply electrical noise. Blocking failure is avoided in our system by the use of bus-guardians which prevents faulty nodes from disturbing communication in slots that are allocated to other nodes. The notion of babbling-idiot failure is not precise enough since it is unclear in how it affects other nodes. We illustrate this with two examples. A babbling-idiot node can potentially adhere to the time triggered protocol but all its own transmitted messages are garbled, i.e. messages from other nodes are not affected. Alternately, a babbling node can continually transmit nonsense thus effectively block messages from other nodes. The latter but not the former is an example of blocking failure.

We assume that CDR-operations with time-triggered communication generally have strict specifications of what constitutes correct behaviour. A failure in such a CDR-operation will violate its specification according to a failure type other than erratic. Erratic failures are hence disregarded.

A time-triggered system with prescheduled communication avoids insertion failure since all messages are known off-line. This implies that an inserted message will violate the schedule. An attempt by a misbehaving node to insert a message in slots belonging to another node will be blocked by the bus-guardian. Unscheduled messages cannot be sent in prescheduled slots, even if the true owner of the slot is silent due to other failure. This avoids insertion failures.

In a time-triggered system, the sender and also receiver identification of a message are distinguished by the slot in which the message was sent. This implies that messages cannot be forged with a fake sender or be received by the wrong node, hence addressing failures are avoided.

Repetition failure can be caused by a fault in a sender service or node hardware which prevents the correct update of outgoing communications buffer. This can cause an old message to be sent. We assume that this failure type can be avoided by the use of sequence counters in messages to detect old or repeated messages. Sequence failure can be avoided in the same way. Sequence and repetition failure is assumed to be plausible if the sending node is faulty. Both these failure types are treated as value failures.

4.1.2 Event-triggered communication

In time-triggered communication omission failures are assumed to be detectable and we disregard timing failures. This is, however, not valid for event-triggered communication. In this communication mode it is generally not possible to distinguish with certainty whether an unreceived message (omitted) is just late (a correct but slow sender) or is caused by a fault in the sender [15]. Schedulability analysis can be applied to event-triggered communication networks to achieve timeliness and real-time performance. An example of this is described by Tindell [16] and can be applied to an event-triggered networks such as CAN [17]. The described scheme sets dead-lines for message arrival which implies that omission can be detected. Schedulability analysis to model the timing of messages can be optionally used in our system model, but is not assumed. Therefore it is relevant to consider both timing failures and omission failures for event-triggered communication. Timing failure is assumed to always be symmetric. A motivation for this assumptions is e.g. that there are no intermediate stages in the communication network.

As before omission failure is further characterised according to symmetry: symmetric or asymmetric.

Blocking failures during the event-triggered communication segment are assumed to be possible and are hence considered. Such failures will have symmetric manifestation.

Failure in CDR-operations that involves event-triggered communication, are assumed to violate their specification according to a predictable failure type. Erratic failures are therefore disregarded.

The sender and receiver of an event-triggered message are known by the address identification in the message header. Addressing failure is avoided by means of the message id. Repetition, insertion and sequence failures are detected by means of the sequence number in the message. To protect against corruption of message id and sequence number messages are protected by CRC which covers both payload and header/trailer. Detection of these four failure types leads to omission of the service to the consumer. These four failure-types are however plausible if the sending node is faulty and are therefore regarded in the failure model.

4.2 Examples of failure modes

We assume that there are two kinds of components in the system model that are able to manifest a signalled failure: processes and run-time environments. Two examples of situations where signalled failure mode is appropriate is a process which is unable to produce a result and a run-time environment which has detected an internal fault in its processor. Instead of a result they send a message which explicitly indicates the failure to the consumers. We assume that a majority of faults that occur in a node are detected and can be manifested with signalled failure.

The symmetry aspect of a failure accounts for the “oracle” perspective of the failure at all nodes in the system. If all nodes have the same perspective then the failure is symmetric, else if nodes have different perspectives then the failure is asymmetric. The symmetry aspect is only meaningful in a distributed system e.g. where there are several receivers of a message. A fault in a node containing the producer, such as in the outgoing link, generally causes a symmetric failure. A fault in the node containing the consumer, such as in the incoming link, generally causes an asymmetric failure. A fault in the network can cause either asymmetric or symmetric failure. The two following are examples of asymmetric failures: 1) A subset of consumer processes do not receive a result while other consumer processes receive the result correctly. 2) Different consumers receive different results, none of which are necessarily correct.

The type and detectability aspect accounts for the perspective of a failure at a single node or component. Detectability decides whether the receiving node can or cannot determine that the message is erroneous by a private inspection, i.e. individual assessment based only on incoming messages. A corrupt checksum in a message that is detected by the receiver service is an example of a detectable failure. Undetectable value failure implies that individual assessment based only on incoming messages (including the payload) does not suffice to resolve the failure. A corrupt payload can be undetectable unless it is detected by e.g. a checksum.

A node potentially contains several processes that participate in CDR-operations and contribute to services. If the run-time environment fails (including sender and receiver services) or the node suffers a major failure such as a failure of the processor, then all service from the node is disrupted, i.e. **complete node failure**. This is also caused by failure of all processes in the node. Complete node failure implies failure of all CDR-operations where the producer resides in the failed node. A failure of a single process only affects a single service, i.e. **partial node failure**. This implies failure of the CDR-operations where the failed process is the producer. Loss of one or several messages from a particular process also causes partial node failure, e.g. due to network fault.

4.3 Related failure modes

Fail-silent behaviour [18] is a failure mode of a system, sub-system, node or component to only produce a correct service or nothing at all, i.e. silence. Alternately it may produce service that can be identified as being incorrect by all users of the service. This failure mode is mainly applicable to synchronous systems. **Crash** behaviour is similar to fail-silent and is often used to describe more loosely coupled system models such as timed asynchronous model [19]. Crash implies that a node stops producing results and does not recover from the failure. A similar concept to fail-silent is **fail-stop** [20]. A fail stop node that is affected by a fault will stop producing results, i.e. halt on failure. A fail-stop node also contains stable storage, typically the state of the service produced by the node, which is always accessible to correct nodes despite any failure. This can be used to resume operation of the lost service in another node which has spare processing capacity.

Applied to our taxonomy of CDR-failures, fail-silent behaviour, fail-stop and crash failure of for example a producer of a service leads to loss of the service (permanent in the crash case) at all consumers. However from the perspective of an individual consumer it is not immediately evident whether the loss of service is due to a failure at the producer or at the consumer i.e. send-omission or receive-omission. An example of the former is complete node failure. An example of the latter is incoming link failure in the consumer node. These two cases require different strategies to handle the failure, with the receive-omission being more challenging due to the asymmetric manifestation at nodes that is caused.

Fail-silent, crash and fail-stop failure modes are not limited to a single CDR-operation and potentially persist over several operations. At a later time the affected component etc. may recover and resume normal service due to recovery.

Byzantine failure is a much discussed failure mode which was coined by Lamport et al. in [21]. It is defined as the loss of a system service, in a system that requires consensus, due to a fault that presents different symptoms to different observers [22]. Byzantine failure implies that at least one receiver has received and accepted a result that is different (i.e. inconsistent) to the result received and accepted by other receivers. In our terminology we interpret a byzantine failure to correspond to an undetectable asymmetric value failure. The interpretation depends on the context of the failure, i.e. the system architecture etc. The cause of a byzantine failure can be a simple electrical phenomenon. An example is the difference in interpretation at receivers of an incoming digital signal which is stuck at 1/2 or has slow rise-time from the digital representation of 0 to 1 [22]. A marginal electrical disturbance can also cause variations in the interpretation at receivers and potentially cause a “schrödingers CRC” phenomenon [22]. This implies that bit-flips have occurred in such a pattern that the CRC is still valid for the message and the message is accepted as correct. A byzantine failure can be caused by a radiation-induced bit-flip that occurs in the memory of a consumer after the result was successfully distributed from the producer. At this stage the result in the memory may not be protected by a checksum and therefore the failure cannot be detected by individual assessment.

Closely related to byzantine failure mode is **forging failure**: a faulty node (not necessarily the sender) forges (modifies) the result in a message sent by another node. A receiver cannot determine that the message is forged by individual assessment of the result. Forging failure also implies that receivers may receive inconsistent data, i.e. asymmetrically. We interpret this in our terminology as an asymmetric undetectable value failure. **Arbitrary failure** is sometimes used interchangeably with byzantine failure. However, “arbitrary” implies that no assumptions at all can be made about the failure, i.e. contrary to the definition of byzantine failure and our interpretation.

5 A Failure Model for CDR-Failures

This section presents and comments a failure model for the class of system that has been described. The failure model contains a specification of CDR-failures that are plausible in the system. These failures should be handled e.g. by protocols for redundancy-management in the system. The aim of the failure model is to support design of a redundancy-management protocol which handles CDR-failures. All system components involved in a CDR-operation, from producer to consumers, must be free from fault for the operation to be successful. However, from the view of the redundancy-management protocol during a single CDR-operation, faults which occur in the consumer process or receiver service are out of scope. The protocol is assumed to be executed in the run-time environment of nodes. Therefore the protocol cannot detect that a consumer process in the same node is faulty. This is indicated by the dashed line between incoming link and processor in Figure 5. Faulty consumer processes are therefore (temporarily) disregarded in the current CDR-operation. However, the fault will be activated and cause failure of a following CDR-operation when the faulty consumer process is the producer process. The fault is thus eventually detected by the protocol.

Two failure models are given: One for CDR-operations with time-triggered communication in the Flexray static segment (Table 2) and one for event-triggered communication in the Flexray dynamic segment (Table 3). The two communication modes lead to slightly different potential failure modes. The tables summarises the possible mappings between components faults and failure modes. The choices are motivated. Comments made concerning the Table 2 are also valid for Table 3 unless stated. In the tables X denotes a likely mapping and – denotes an unlikely mapping. The tables show the failure modes which are relevant to a redundancy-management protocol. Some combinations of aspects are not assumed to occur in our system (motivated in earlier discussion) and are hence omitted from the table. An example of such a combination is an asymmetric value failure, both detectable and undetectable.

Table 2 summarises the possible mappings between components faults and failure modes for time-triggered communication. Asymmetric failures are assumed to occur mainly due to faults in receiving nodes (incoming link etc.) and the network. However a slightly-out-of-specification (SOS) fault [23] can also cause asymmetric omission failure. The cause can be external disturbance. SOS-faults occur in the time-domain and value domain [24]. An example of the former is deviation of clocks in nodes. The synchrony of clock affect a node's perception of what constitutes a timely message, i.e. correct, and what is a late or early message, i.e. omission. Clocks can deviate in such a combination that a message sent from one node is correctly recieved by some nodes, while other nodes do not recieve the message.

Table 2: Mapping between component faults and applicable CDR failure mode for Flexray static segment.

	Asymmetric omission failure	**Symmetric**			
Faulty component		**omission failure**	**detectable value failure**	**undetectable value failure**	**Signalled failure**
Producer	-	X	X	X	X
Sender service (or Processor)	X	X	X	X	X
Outgoing link (at sender)	X	X	X	X	-
Network	X	X	X	X	-
Incoming link (at receiver)	X	-	-	-	-

Symmetric failures (omission failure and value failure, both detectable and undetectable) occur due to faults in the sending node (process, sender service and outgoing link) and the network. All receivers are affected equally and will have the same perception of the failure. Additionally, a signalled failure informs receivers of a failure in the sender node (either producer process or run-time environment). It is assumed to be symmetric.

Table 3 summarises the possible mappings between components faults and failure modes for event-triggered communication. Event-triggered communication is prone to blocking failure, e.g. caused by disturbance from other nodes. This is assumed to have symmetric manifestation. Timing failures caused by lack of time-specification must be addressed for event-triggered communication. This failure mode can be avoided by applying a user-level scheduling scheme. Event-triggered communication is prone to insertion, repetition, addressing and sequence failures due to software faults in sender nodes. Omission failures in event-triggered communication require an additional mechanism such as time-out to be detectable.

Table 3: Mapping between component faults and applicable CDR failure mode for Flexray dynamic segment.

	Asymmetric	**Symmetric**						
Faulty component	**omission failure**	**IRAS [*1] failure**	**omission failure**	**blocking failure**	**timing failure**	**detectable value failure**	**undetectable value failure**	**Signalled failure**
Producer	-	X	X		X	X	X	X
Sender service (or Processor)	X	X	X	X	X	X	X	X
Outgoing link (at sender)	X	-	X	X	-	X	X	-
Network	X	-	X	X	-	X	X	-
Incoming link (at receiver)	X	-	-	X	-	-	-	-

*1: Insertion, repetition, addressing or sequence failure.

6 Comparisons of failure types

In this section we compare our failure types from Section 3 with communications errors according to clause 5.3 in “Industrial Communication Networks – Profiles – Part 3: functional safety fieldbuses - General rules and profile definitions” IEC 61784-3 [25]. Table 4 presents the comparison with the communication error and then relevant failure type. The effects on event-triggered and time-triggered communication are studied. Abbreviations used in the table are: TT – Time-triggered communication, ET – Event-triggered communication, BG – Bus guardians.

Table 4: Comparison to communication errors

IEC 61784-3 Comm. errors	**Failure type**	**Comment on how our system handles this failure**
5.3.2 Corruption	Value failure	Message corrupted, including message header and payload etc. Detectable or undetectable. If the failure is detected, it implies omission of message. Else the message will continue to be processed by the node.
	Blocking failure	A faulty node jams communication in the system. Avoided with BG in TT. Difficult to avoid in ET.
	Erratic failure	We assume that safety mechanisms, in both TT and ET, imply that this failure will be detected as another failure type.
5.3.3 Unintended repetition	Repetition failure	A syntactically valid, but old message is repeated. In ET this may imply insertion failure. In TT unintended repetition is a special case of value failure.
5.3.4 Incorrect sequence	Sequence failure	Individual messages are syntactically and semantically valid but several messages are received in wrong sequence. This communication error is not applicable CDR-failure since it is defined as a single operation, not a sequence (of messages). In TT unintended repetition is a special case of value failure. Detected by use of sequence numbers
5.3.5 Loss	Omission failure	Asymmetric or symmetric omission of a message. Lost message is detectable in TT. Detectable in ET if time-out is detectable. Lost message can be due to detected corruption in message, including header or payload, etc.
5.3.6 Unacceptable delay	Timing failure	Message is generally not accepted by the TT local at the nodes. In TT, a too late (or early) message is treated as an omission. If node does not handle this case, it can still be avoided with BG.
5.3.7 Insertion	Insertion failure	Syntactically valid message. In TT, slots belong to specific nodes and cannot be used by another node. Avoided with BG. Plausible in ET. Detected by use of sequence numbers.
5.3.8 Masquerade	Addressing failure	Syntactically valid, but wrong (possibly forged) sender address. Avoided in TT with a fixed communication schedule. Sender is known by transmission slot. A faulty sender node in ET- can forge sender and receiver id of the message, i.e. address.
5.3.9 Addressing	Addressing failure	Syntactically valid, but wrong receiver address. Avoided in TT with a fixed communication schedule. Receiver is known by transmission slot. A faulty sender node in ET can forge sender and receiver id of the message, i.e. address.

7 Failure models used in related work

A short survey of the failure models which are used in other related work is given. The surveyed articles describe redundancy-management protocols, such as membership agreement and diagnosis protocols and also assume distributed real-time system environment similar to ours.

An example of a common fault and system model is used by Rosset et al. in [26], Ezhilchelvan and Lemos in [27] and Kopetz and Grünsteidl in [28] (TTP). The system and fault models used in these articles are very similar to each other. The system is composed of a fixed set of processors interconnected by a broadcast network. The network offers statically scheduled communication with one communication slot per processor in each communication cycle. Processors can fail by experiencing three possible faults: Crash fault – a processor ceases operation when a fault occurs. Receive fault – a fault on reception of a message broadcast by another node. Send fault – a fault occurs when sending a message. Note that the network itself is assumed not to suffer faults. Compared to our failure model the crash and send fault are symmetric in their manifestation, while the receive fault is asymmetric. Value and signal failure is not discussed explicitly. The smallest unit of failure is the processor in contrast to the process as in our model. This is because the respective protocols are designed to monitor nodes and does not distinguish processes. The TTP fault model [28] also assumes that faults do not arrive more often than one fault per two TDMA rounds.

Table 5: Kopetz's failure modes

Failure mode according to Kopetz [29]	**Example of occurrence**	**Kopetz failure mode mapped to our model and comment**
Babbling idiot failure	An incorrect node sends untimely message(s) which blocks other nodes from sending correctly.	Blocking failure. Symmetry and persistence is not specified. Node could send one, several or indefinitely send untimely messages. This failure is assumed to be detectable since TT is implied.
Masquerading failure	A node sends a message with the identity of another node.	Addressing failure. Symmetry and persistence is not specified.
Slightly-off specification (SOS) failure	Nodes in a system have slightly varying	Either receive omission failure or value failure. Failure is asymmetric since it results in inconsistent system state. Persistence is not specified. Detectability is not specified for value failure.
Crash/omission failure	Sender node operates incorrectly and does not send message correctly. Message is disturbed in the network.	Send omission failure. Symmetry and persistence is not clearly specified. Assume that this failure is detectable since TT is implied.
Massive transient failure	External EM disturbance leading to temporary loss of communication.	Several CDR-operations are affected. Characteristic of each CDR-failure is not specified. Temporary symmetric omission failures seems to be implied by Kopetz.

Kopetz investigates fault containment and error detection in TTP/C and Flexray [29]; both communication networks are used in real-time distributed systems. In this report five critical failure modes are considered: Babbling idiot failure, masquerading failure, slightly-off specification failure, crash/omission failure and massive transient failure. **Table 5** exemplifies these failure modes and they are mapped to failures in the failure model that we present.

Barbosa and Karlsson propose a membership protocol for a synchronous distributed real-time system [30]. Processor nodes are interconnected with a time-triggered broadcast network. A failure model is used where failures can occur in the sending node, outgoing link, network, incoming link or receiving node. The two former imply that the failure has symmetric manifestation. The two latter imply that the failure has asymmetric manifestation. The network causes either symmetric or asymmetric manifestation. Failure causes messages to not be correctly received by the receivers. The smallest unit of failure is the processor.

Serafini et al. propose a tuneable diagnostic protocol for time-triggered systems [31]. Processor nodes are interconnected with a time-triggered broadcast network and the smallest unit of failure is the processor. Faults occur in messages and are assumed to be of three types: 1) Symmetric malicious - the message is semantically incorrect and is perceived identically by all correct nodes. This corresponds to symmetric undetectable value failure in our model. 2) Symmetric benign - the message is syntactically incorrect and is perceived identically by all nodes. This corresponds to symmetric (detectable) omission and symmetric detectable value failure in our model. 3) Asymmetric - the message is syntactically incorrect, i.e. benign and is not perceived identically by all nodes. This corresponds to asymmetric omission failure in our model. Benign messages are hence detectably corrupt while malicious messages are undetectably corrupt. Serafini assumes that malicious messages are assumed to always be symmetric.

The system and failure model that is presented in this report is informally described. Charrron-Bost and Schiper present the Heard-of model to describe fault-tolerant distributed computing [32] in more formalised terms. In this model the notion of transmission faults and each process tracks the other processes that it has “heard of” during each communication round. These processes are then part of the heard-of set for a particular round r and process p: HO(p,r). The failure model, system model, communication model is implicitly described as a predicate over the heard-of set.

In EAST-ADL [33, 34] is an Architecture Description Language for Automotive embedded electronic systems. One of its concerns is non-functional operating properties such as error models. Safety constraints are used to define the criticality of faults and failures. However, simultaneous failures cannot be represented and therefore not asymmetric faults and failures. The typical pattern for error modelling in EAST-ADL is to define a structure which captures the perceived failure behaviour and error propagation in the nominal architecture. For the allocation-aware design level, separate error models are typically made for hardware and software, and the hardware failures propagates to the functional architecture. EAST-ADL uses the term “error model” while the CDR concept uses “failure model” because the context is different.

Our contribution is the inclusion in the failure model of the process (and other components within the node) as smallest unit of failure rather than entire node or processor. Further, we contribute with a specific system model and the concept of CDR-operation. Our system- and failure models are more detailed than those described above for similar protocols [26], [27], [28], [30] and [31].

7.1 Comparison with composite failure models

A composite failure model categorises failures into a small number of groups. An example is a byzantine failure model. Here only one type of failure is assumed and the

nature of that failure is not immediately clear. In contrast, our failure model is differentiated and acknowledges that a system must handle failures that are different in nature and severity. Composite failure modes, such as byzantine failure, are avoided in the proposed taxonomy. Instead the aspects of a failure are decomposed. In this section we discuss the advantage of having a differentiated as opposed to composite failure model. We give three examples of composite failure models used in protocols for byzantine agreement i.e. consensus on a result in a distributed system.

Lamport et al. [21] proposed an algorithm OM (oral messages) that solves agreement in a system with byzantine faults. It uses oral messages, i.e. messages can be forged by intermediate nodes and the algorithm is denoted OM(*m*). The algorithm uses *m* rounds of communication (not including the initial round) to solve agreement despite there being up to *m* byzantine faults in nodes and provided that the number of nodes (N) in the system satisfies $N>3m$. The fault model used for OM assumes that all faults are always worst case, i.e. byzantine faults. This leads to high requirements on system architecture and high costs.

Thambidurai and Park [35] describes a hybrid fault model which acknowledges that faults which affect a system will be of different categories. The hybrid fault model partitions faults into **non-malicious, symmetric malicious** and **asymmetric malicious**. Non-malicious and malicious correspond to detectable and undetectable in our taxonomy. Non-malicious faults are also sometimes called **benign** faults.

Lincoln and Rushby [36] also use the hybrid fault model but give a slightly different naming of the same three fault classes: **manifest**, **symmetric** and **arbitrary** respectively. OMH handles a larger number of more constrained faults (symmetric and manifest) than OM, which only regards the worst case arbitrary faults. We give the following example. A system must tolerate one asymmetric malicious and one non-malicious. If it is built according to OM the worst case of the failures is automatically assumed, i.e. both are asymmetric malicious, and the system must contain at least 7 nodes. If the system is built according to OMH, then five nodes will suffice. Less components in a system also leads to a lower failure rate since there are fewer components that can fail. This is investigated in [37]. The example of the two algorithms OM and OMH demonstrate the advantage of using a detailed fault model.

We note that the hybrid fault model [35], does not capture as many details as our failure model does. Especially asymmetric omission failure, which is of great interest to us, cannot be categorised in the hybrid fault model. The hybrid fault model categorises omission (and other detectable faults, e.g. crash) as non-malicious. Further, the symmetry aspect is not used to discriminate non-malicious faults, and thus both symmetric and asymmetric non-malicious faults are seen to be equally bad. We suspect that this stems from an assumption made in [35] that a distributed system tolerates detectable asymmetric faults, e.g. when some nodes receive a valid result while others do not. Consistency of results among nodes is not needed as long as faults are detectable. We do not agree with this view since our assumption is that consistent perception of faults at correct nodes is vital in our system.

8 Conclusion

This report presents a system model for a class of distributed system that are used in safety-critical embedded application. The system model includes a node model, network model, processing model and a failure model. The failure model details the failure modes are plausible in the system. These failures should be handled by the application and/or a protocol that implement fault-tolerance functions. The report includes a taxonomy with which failures can be clearly described. The aim of the report is to support the design of protocols for redundancy-management, such as a membership agreement protocol, by investigating the affect of the system model on the failure model.

The system is assumed to provide a high degree of error containment for real-time processes executed in the same node (intra-node error containment), and for processes executed on different nodes (inter-node error containment). The smallest unit of failure in the system is therefore a process (and other components within the node) rather than the entire node. The system model uses a broadcast communication network similar to Flexray, i.e. support both time-triggered and event-triggered communication. A separate failure model is given for each communication mode.

The system uses a processing model which focuses on sequentially executed primitive operations called CDR-operations. Such an operation involves the computation of a result by a producer process and the distribution of that result as payload in a message to consumer processes. The failure model regards failure of CDR-operations. A taxonomy of CDR-failures is given. CDR-failures are characterised according to four aspects: type, symmetry, detectability and persistence.

A comparison between CDR-failures and communication errors according to IEC 61784-3 is made. A short survey of failure models and system models that are used for other redundancy-management protocols is given.

9 Discussion

The failure model that is presented in Section 0 is specifically made for the system model that is presented, i.e. containing a processing model based on CDR-operations and a network model similar to Flexray with both static and dynamic communication. Another system model for a slightly different application can have a different network model e.g. with Ethernet which will lead to a different failure model; new failures will be plausible etc. We believe that the taxonomy of failures and ideas in this report can be adapted and applied to other system models for similar application to achieve a clear failure model.

In this report a system with a network similar to FlexRay is assumed. Examples of other communication networks that are suitable for the system model are: TTCAN – which supports both time-triggered and event-triggered communication [38]; Ethernet – which is event-triggered and lacks determinism but has a large user base; TTP/C – which supports only time-triggered communication by default [7]. An alternative node model may contain nodes that lack bus-guardians as these devices are uncommon in current automotive applications. An alternative processing model processing may be more centralised than in the CDR-model.

The means for dependability to handle failure can be implemented in the application or protocol for redundancy management. An example of such a protocol is membership agreement [39]. The protocol allows a group of co-operating real-time processes in a distributed system to establish a consistent view of each other's operational status, i.e. working correctly or not. A membership protocol provides valuable support for implementing fault-tolerance and graceful degradation in distributed systems.

10 Acknowledgments

This research was conducted within the project CEDES, Cost Efficient Dependable Embedded Systems, which is funded by the Swedish industry and government joint research program IVSS - Intelligent Vehicle Safety Systems. Funding has also been supplied by SP – Technical Research Institute of Sweden.

11 References

1. Aidemark, J., *Node-level fault tolerance for embedded real-time systems*, in *School of Computer Science and Engineering*. 2004, Chalmers University of Technology: Göteborg.
2. IEC, *IEC 61784-3 series: Industrial communication networks*, in *Part 3: Functional safety fieldbuses - General rules and profile definitions*. 2010, International Electrotechnical Commission: www.iec.ch.
3. Avizienis, A., et al., *Basic concepts and taxonomy of dependable and secure computing.* Dependable and Secure Computing, IEEE Transactions on, 2004. **1**(1): p. 11-33.
4. Temple, C. *Avoiding the babbling-idiot failure in a time-triggered communication system*. 1998. Munich, Germany: IEEE Comput. Soc.
5. Bauer, G., H. Kopetz, and W. Steiner. *The central guardian approach to enforce fault isolation in the time-triggered architecture*. 2003. Pisa, Italy: IEEE Comput. Soc.
6. Broster, I. and A. Burns. *An analysable bus-guardian for event-triggered communication*. 2003. Cancun, Mexico: Institute of Electrical and Electronics Engineers Inc.
7. Kopetz, H. and G. Bauer, *The time-triggered architecture.* Proceedings of the IEEE, 2003. **91**(1): p. 112-126.
8. Flexray, *FlexRay Protocol Specification Version 3.01*. 2010: FlexRay Consortium.
9. Führer, T., et al., *Time Triggered Communication on CAN (Time Triggered CAN-TTCAN).*
10. Forest, T.M. *The FlexRay Communication Protocol and Some Implications for Future Applications*. in *SAE Convergence*. 2006: SAE Press.
11. Berwanger, J., M. Peller, and R. Griessbach. *Byteflight a new protocol for safety critical applications*. 2000.
12. Torin, J., et al., *Generic architecture for avionic systems NFFP4 S4207 (WP 1)*, in *Technical Report No. 2009:2*, T.R.N. 2009:2, Editor. 2009, Technical Report No. 2009:2, Department of Computer Science and Engineering Division of Networks and Systems, Chalmers University of Technology: Göteborg, Sweden
13. Saltzer, J.H., D.P. Reed, and D.D. Clark, *End-to-end arguments in system design.* ACM Transactions Computer Systems, 1984. **2**(4): p. 277-288.
14. Paulitsch, M., et al. *Coverage and the use of cyclic redundancy codes in ultra-dependable systems*. 2005.
15. Chandra, T.D. and S. Toueg. *Unreliable failure detectors for asynchronous systems*. 1991. Montreal, Que., Canada: ACM.
16. Tindell, K., A. Burns, and A.J. Wellings, *Analysis of hard real-time communications.* Real-Time Systems, 1995. **9**(2): p. 147-171.
17. CAN, *CAN Specification Version 2.0.* 1991, Robert Bosch GmbH.
18. Powell, D., et al. *The Delta-4 Approach to Dependability in Open Distributed Computing Systems*. 1995.
19. Cristian, F. and C. Fetzer. *The timed asynchronous distributed system model*. 1998. Munich, Germany: IEEE Computer Society.

20. Schneider, F.B., *Byzantine generals in action: implementing fail-stop processors.* ACM Trans. Comput. Syst., 1984. **2**(2): p. 145-154.

21. Lamport, L., R. Shostak, and M. Pease, *The Byzantine Generals Problem.* ACM Transactions on Programming Languages and Systems, 1982. **4**(3): p. 382-401.

22. Driscoll, K., et al. *The real Byzantine Generals*. in *The 23rd Digital Avionics Systems Conference (DACS)*. 2004. Salt Lake City, UT, USA.

23. Ademaj, A., *Slightly-off-specification failures in the time-triggered architecture,* in *Proceedings of the Seventh IEEE International High-Level Design Validation and Test Workshop(HLDVT)*. 2002, IEEE Computer Society.

24. Sivencrona, H., et al. *Heavy-ion fault injections in the time-triggered communication protocol*. in *Dependable Computing. First Latin-American Symposium, LADC*. 2003. Sao Paulo, Brazil: Springer-Verlag.

25. IEC, *IEC 61784-3 series: Industrial communication networks – Profiles – Part 3: Functional safety fieldbuses* in *COMMITTEE DRAFT FOR VOTE*. 2009, International Electrotechnical Commission.

26. Rosett, V., P. Souto, and F. Vasques. *A group membership protocol for communication systems with both static and dynamic scheduling*. in *Workshop on factory communications systems*. 2006.

27. Ezhilchelvan, P.D. and R. de Lemos. *A robust group membership algorithm for distributed real-time systems*. 1990. Lake Buena Vista, FL, USA: IEEE Computer Society Press.

28. Kopetz, H. and G. Grunsteidl, *TTP-a protocol for fault-tolerant real-time systems.* Computer, 1994. **27**(1): p. 14-23.

29. Kopetz, H., *Fault Containment and Error Detection in TTP/C and FlexRay*, in *Research Report 23/2002*. 2002, Technical University of Vienna, VMARS

30. Barbosa, R. and J. Karlsson. *Flexible, cost-effective membership agreement in synchronous systems*. in *12th Pacific Rim International Symposium on Dependable Computing, PRDC 2006*. 2006. Riverside, CA, United States: Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States.

31. Serafini, M., et al. *A Tunable Add-On Diagnostic Protocol for Time-Triggered Systems*. in *Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on*. 2007. Edinburgh, United Kingdom: Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States.

32. Charron-Bost, B. and A. Schiper, *The Heard-Of model: computing in distributed systems with benign faults.* Distributed Computing, 2009. **22**(1): p. 49-71.

33. Cuenot, P., et al., *Engineering Support for Automotive Embedded Systems - Beyond AUTOSAR.* ATZautotechnology 2009(April).

34. Debruyne, V., F. Simonot-Lion, and Y. Trinquet, *EAST-ADL—an architecture description language.* Architecture Description Languages, 2005: p. 181-195.

35. Thambidurai, P. and Y.-k. Park. *Interactive consistency with multiple failure modes*. in *Reliable Distributed Systems, 1988. Proceedings., Seventh Symposium on*. 1988.

36. Lincoln, P. and J. Rushby. *Formally verified algorithm for interactive consistency under a hybrid fault model*. 1993. Toulouse, Fr: Publ by IEEE, Piscataway, NJ, USA.

37. Thambidurai, P., Y.K. Park, and K.S. Trivedi. *On reliability modelling of fault-tolerant distributed systems*. in *Distributed Computing Systems, 1989., 9th International Conference on*. 1989.

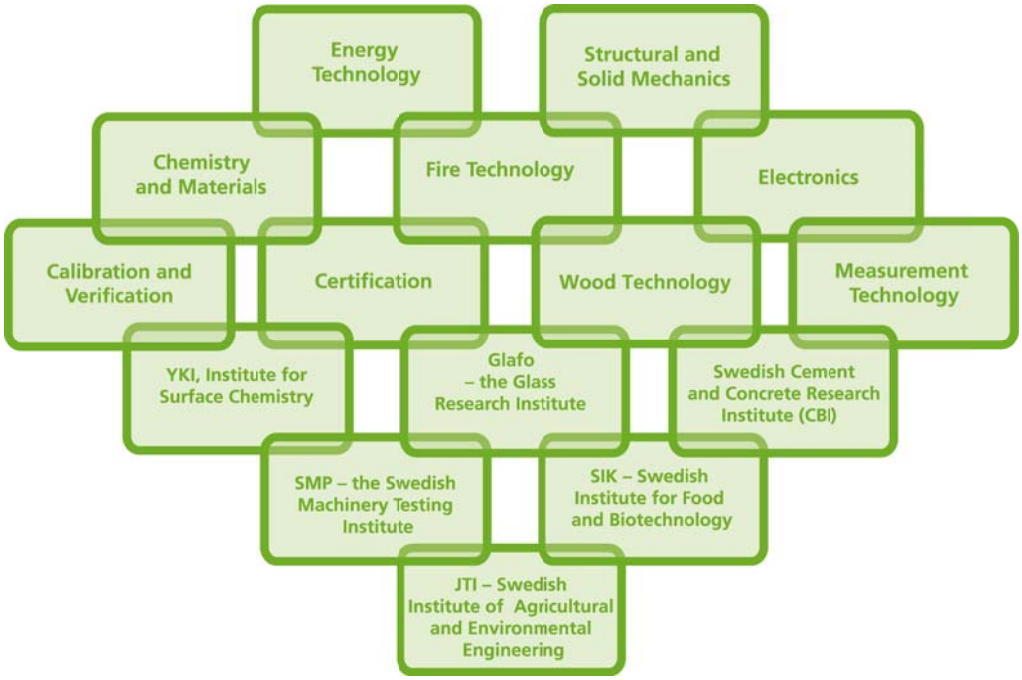
38. Zeltwanger, H., *Time-Triggered communication on CAN*. 2002. **2002-01-0437**.

39. Bergenhem, C. and J. Karlsson. *A Process Group Membership Service for Active Safety Systems Using TT/ET Communication Scheduling*. in *Dependable Computing (PRDC 2007). 13th Pacific Rim International Symposium on*. 2007. Melbourne, Australia.

12 Index of keywords

SP Technical Research Institute of Sweden

Our work is concentrated on innovation and the development of value-adding technology. Using Sweden's most extensive and advanced resources for technical evaluation, measurement technology, research and development, we make an important contribution to the competitiveness and sustainable development of industry. Research is carried out in close conjunction with universities and institutes of technology, to the benefit of a customer base of about 9000 organisations, ranging from start-up companies developing new technologies or new ideas to international groups.



SP Technical Research Institute of Sweden

Box 857, SE-501 15 BORÅS, SWEDEN

Telephone: +46 10 516 50 00, Telefax: +46 33 13 55 02

E-mail: info@sp.se, Internet: www.sp.se

www.sp.se

Electronics

SP Report 2012:31

ISBN 978-91-87017-45-2

ISSN 0284-5172

More information about publications published by SP: **www.sp.se/publ**