Distributed Fault Diagnosis for Networked Embedded Systems

Master's thesis
performed in **Vehicular Systems**

by
Dan Hallgren
Håkan Skog

Reg nr: LiTH-ISY-EX-3820-2005

December 21, 2005

Distributed Fault Diagnosis for Networked Embedded Systems

Master's thesis

performed in **Vehicular Systems**,
Dept. of Electrical Engineering
at **Linköpings universitet**

by

Dan Hallgren
Håkan Skog

Reg nr: LiTH-ISY-EX-3820-2005

Supervisor: **Mathias Jensen**
Scania CV AB
Jonas Biteus
Linköpings Universitet

Examiner: **Assistant Professor Erik Frisk**
Linköpings Universitet

Linköping, December 21, 2005

Avdelning, Institution
Division, Department

Vehicular Systems,
Dept. of Electrical Engineering
581 83 Linköping

Datum
Date

December 21, 2005

Språk Language	**Rapporttyp** Report category
☐ Svenska/Swedish	☐ Licentiatavhandling
☒ Engelska/English	☒ Examensarbete
	☐ C-uppsats
	☐ D-uppsats
☐ ____________	☐ Övrig rapport
	☐ ____________

ISBN
—

ISRN
LITH-ISY-EX-3820-2005

Serietitel och serienummer Title of series, numbering
ISSN
—

URL för elektronisk version
http://www.vehicular.isy.liu.se
http://www.ep.liu.se/exjobb/isy/2005/3820/

Titel Distribuerad feldiagnos för nätverksbaserade inbyggda system

Title Distributed Fault Diagnosis for Networked Embedded Systems

Författare Author Dan Hallgren och Håkan Skog

Sammanfattning
Abstract

In a system like a Scania heavy duty truck, faultcodes (DTCs) are generated and stored locally in the ECUs when components, e.g. sensors or actuators, malfunction. Tests are run periodically to detect failure in the system. The test results are processed by the diagnostic system that tries to isolate the faulty components and set local faultcodes.

Currently, in a Scania truck, local diagnoses are only based on local diagnostic information, which the DTCs are based upon. The diagnosis statement can, however, be more complete if diagnoses from other ECUs are considered. Thus a system that extends the local diagnoses by exchanging diagnostic information between the ECUs is desired. The diagnostic information to share and how it should be done is elaborated in this thesis. Further, a model of distributed diagnosis is given and a few distributed diagnostic algorithms for transmitting and receiving diagnostic information are presented.

A basic idea that has influenced the project is to make the diagnostic system scalable with respect to hardware and thereby making it easy to add and remove ECUs. When implementing a distributed diagnostic system in networked real-time embedded systems, technical problems arise such as memory handling, process synchronization and transmission of diagnostic data and these will be discussed in detail. Implementation of a distributed diagnostic system is further complicated due to the fact that the isolation process is a non deterministic job and requires a non deterministic amount of memory.

Nyckelord
Keywords Distributed diagnosis, OBD, Fault isolation, Embedded systems, DTC

Abstract

In a system like a Scania heavy duty truck, faultcodes (DTCs) are generated and stored locally in the ECUs when components, e.g. sensors or actuators, malfunction. Tests are run periodically to detect failure in the system. The test results are processed by the diagnostic system that tries to isolate the faulty components and set local faultcodes.

Currently, in a Scania truck, local diagnoses are only based on local diagnostic information, which the DTCs are based upon. The diagnosis statement can, however, be more complete if diagnoses from other ECUs are considered. Thus a system that extends the local diagnoses by exchanging diagnostic information between the ECUs is desired. The diagnostic information to share and how it should be done is elaborated in this thesis. Further, a model of distributed diagnosis is given and a few distributed diagnostic algorithms for transmitting and receiving diagnostic information are presented.

A basic idea that has influenced the project is to make the diagnostic system scalable with respect to hardware and thereby making it easy to add and remove ECUs. When implementing a distributed diagnostic system in networked real-time embedded systems, technical problems arise such as memory handling, process synchronization and transmission of diagnostic data and these will be discussed in detail. Implementation of a distributed diagnostic system is further complicated due to the fact that the isolation process is a non deterministic job and requires a non deterministic amount of memory.

Keywords: Distributed diagnosis, OBD, Fault isolation, Embedded systems, DTC

Preface

This master's thesis was performed at Scania CV AB in Södertälje, Sweden. Scania is a worldwide manufacturer of heavy duty vehicles, buses and engines for marine and industrial use. The work was carried out at the *Engine Software and OBD* group at the *Powertrain Control System Development* department.

Thesis outline

Chapter 1 Introduction to the thesis.

Chapter 2 Theory of model based diagnosis.

Chapter 3 Theory of distributed systems.

Chapter 4 Theory of distributed diagnosis.

Chapter 5 Proposed algorithms for distributed diagnosis.

Chapter 6 Issues with implementation in an embedded system environment.

Chapter 7 Conclusions of the thesis.

Chapter 8 Future work.

Acknowledgment

We would like to thank our supervisor at Linköpings Universitet, *Jonas Biteus*, for always taking time to answer our questions and guiding us through the project. We would also like to thank *all people at Scania Powertrain Control System Development* who supported us with guidance and special knowledge. Special thanks goes to our supervisor, *Mathias Jensen*, for all those fruitful discussions regarding DIMA and diagnosis in general, *Kristian Krigsman* for his helpfulness and always putting up with our questions regarding implementation, *Ulf (*CAN*King) Carlsson* and finally *Mattias Nyberg* for sharing his advice on both fault diagnosis and the project as a whole.

Dan Hallgren *Håkan Skog*

Södetälje, December 2005

Contents

Abstract v

Preface and Acknowledgment vi

1 Introduction 1
1.1 Background . . . 1
1.2 Objective . . . 2
1.3 Approach . . . 2
1.4 Contribution . . . 3
1.5 Delimitations and Assumptions . . . 3
1.6 Target Group . . . 3
1.7 Related Work . . . 3

2 Model Based Diagnosis 5
2.1 Introduction to Model Based Diagnosis . . . 5
2.2 Artificial Intelligence and Fault Diagnosis . . . 6
2.2.1 Behavioral Modes . . . 7
2.2.2 Diagnoses . . . 7
2.2.3 Conflicts . . . 9
2.2.4 Relations between Diagnoses and Conflicts . . . 9
2.2.5 Diagnostic Tests . . . 10
2.3 Local Algorithms . . . 11
2.3.1 Reiter's Algorithm . . . 11
2.3.2 Isolation with Generalized Fault Modes . . . 13
2.3.3 Virtual Components . . . 14

3 Distributed Systems 19
3.1 Properties of Distributed Systems . . . 19
3.1.1 Transparency . . . 19
3.1.2 Openness . . . 20
3.1.3 Scalability . . . 21
3.2 Hardware Concepts . . . 21
3.2.1 The CAN Bus . . . 22

4 Distributed Diagnostic Systems **25**
4.1 The Network Architecture 25
4.2 Current Diagnostic System 26
4.2.1 The Goal with the Distributed Diagnostic System . . 27
4.3 Components, Signals and Objects 28
4.4 Signals - Inputs and Outputs 30
4.5 Local and Global Diagnosis 31
4.5.1 Two Ways of Calculating the Global Diagnosis . . . 33
4.5.2 The Combinatorial Problem 33
4.5.3 Merging Minimal Cardinality Diagnoses 35
4.6 Centralized or Distributed Diagnosis 36
4.6.1 Centralized Diagnosis and Decentralized Diagnosis . 36
4.6.2 Distributed Diagnosis 38
4.7 Sharing Diagnostic Information 38
4.7.1 Sharing Conflicts 39
4.7.2 Sharing Diagnoses 40
4.7.3 The Information to Share 41
4.7.4 Focusing on Probable Diagnosis 42
4.7.5 Problems with Component Representation 43

5 Proposed Methods for Distributed Diagnosis **45**
5.1 Model for Distributed Diagnosis 45
5.2 Algorithms for Distributed Diagnosis 47
5.2.1 Method 1: Sharing Conflicts 48
5.2.2 Method 2: Sharing Diagnoses 50
5.2.3 Method 3: Sharing Diagnoses Extended 52
5.3 Discussion Concerning the Limitations and Assumptions . . 57

6 Implementation in an Embedded System **59**
6.1 Hardware Setup 59
6.2 Software Description 59
6.3 Processes in Embedded Systems 60
6.4 Data Transferring on a CAN Bus 61
6.4.1 Protocol Design 62
6.4.2 Transparency 62
6.5 Memory Structure 62
6.5.1 Memory Conflicts 63
6.6 Time Handling 63
6.6.1 Diagnosis Executed in a Fixed Timed Loop 63
6.6.2 Diagnosis Executed in the Background Process . . . 64
6.6.3 Synchronization 64
6.7 Using Reiter's Algorithm in Distributed Diagnosis 65
6.8 Performance of the Implementation 67

7 Conclusions **69**

8 Future Work **71**

References **73**

Notation **75**

A Proof of Method 3 **77**

Chapter 1

Introduction

The field of *distributed diagnosis* is an active topic in the world of fault diagnosis. At Scania, the subject needs to be elaborated and that is the main underlying cause for this master's thesis. In this chapter an introduction to the master's thesis is given.

1.1 Background

In modern automotive vehicles, several *Electronic Control Units* (ECUs) communicate over a local network. Each ECU is connected to a number of components, e.g. sensors and actuators that are monitored by a diagnostic system to make sure that the components are operating correctly. The diagnostic system usually consists of a number of precompiled tests, simple or complex, to perform the monitoring.

When a component becomes faulty, all tests involving that specific component should become invalidated and the diagnostic system should assign a *Diagnostic Trouble Code* (DTC) to each component that could possibly be faulty.

Tests in a specific ECU can involve components connected to other ECUs based on information shared over the network, see Figure 1.1. Each ECU generates a set of local diagnoses. The tests are thereby entangled but no diagnostic information is shared over the network. Thus the stated local sets of diagnoses are incomplete and in order to have a complete diagnosis statement, diagnostic information has to be transmitted over the network.

Due to continuous development of new environmental laws, an *On Board Diagnosis* (OBD) system is needed to detect and isolate faulty components that affect the pollution of the vehicle. In the future, the laws will demand certain actions to be taken, e.g. torque restriction, if such a fault is detected. It is thereby very important that all decisions made by the system are based on correct information, thus a sophisticated diagnostic system is necessary to meet the laws of tomorrow.

The possible designs of such a diagnostic system are many and the solution is not obvious. Different methods need to be investigated and main issues have to be discussed.

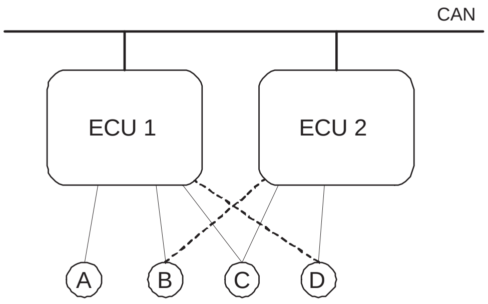

Figure 1.1: A typical layout of ECUs, components and test sensitivity.

1.2 Objective

The objective of this thesis is to present one or several methods to increase the performance of the local diagnostic systems by letting the ECUs exchange information enabling local diagnoses, consistent with the global diagnoses, to be calculated. Also the objective is to implement a distributed diagnostic system and to examine the problems that arise and if the following desirable characteristics can be fulfilled:

- The algorithms should be fast and effective since both processing power and memory are limited in each ECU.
- The diagnostic information shared over the network should be kept at a minimum, because the bandwidth of the network is limited and used for many other applications.
- The system should work independently of the system configuration, e.g. one should be able to connect, remove or exchange one ECU without affecting the diagnostic system of the other ECUs.

1.3 Approach

The main approach in this master's thesis was to first explore the field of relevant articles and literature to do some research on previous work. Special focus was on distributed systems and multi agent diagnostic systems. Later, based on the literature and previous work at Scania, algorithms for distributed diagnosis were designed. The early algorithms only served as a framework for further development and did not have full functionality.

In the second phase of the project a hardware rig was constructed and the ideas were implemented to test and investigate the main issues of a distributed diagnostic system. The methods were extended to fully suit the existing local system where general behavioral modes are used.

The work was documented using LaTeX during the whole proceeding of the project. The implementation was done in the C programming language.

1.4 Contribution

The main contribution of this thesis are the proposed methods for distributed diagnosis, described in chapter 5, and the implementation of a distributed diagnostic system, found in chapter 6. All methods that are presented comply with the objective. One of the methods focus on minimizing the diagnostic data transmitted over the network. In chapter 6 issues arising at the implementation such as memory conflicts and synchronization are discussed in detail.

1.5 Delimitations and Assumptions

The focus of this master's thesis is to perform the best possible isolation based on the test results. No attention is thus given to how the tests work or how they are implemented.

It is assumed that a component is restricted to only one behavioral mode at one point in time.

No effort is spent on optimizing the implementation w.r.t. memory consumption or execution time. The implementation should only serve as a framework for further development and to test ideas.

1.6 Target Group

This thesis is written for engineers and students with basic knowledge in vehicular systems, fault diagnosis and distributed systems.

1.7 Related Work

The main preceding work at Scania CV AB in the field distributed diagnosis is Mathias Jensen's master's thesis [Jen03]. The thesis includes detailed information about local diagnosis algorithms and some information about how local diagnoses could be used to form globally consistent diagnoses. Research in distributed diagnosis for embedded system, well suited for systems like those in a Scania truck, can be found in Jonas Biteus' licentiate thesis [Bit05a]. The foundation of the methods presented in this thesis is obtained from Biteus.

How diagnosis can be performed in large active distributed systems is discussed in Baroni et al. [PBZ98]. The approach in this article is to perform diagnosis by a modular automata technique. The main goal of this diagnostic technique is the reconstruction of the behavior of the active system starting from a set of observable events. Another interesting paper is James Kurien et al. [JKZ02], where an algorithm for distributed diagnosis in networked embedded systems is presented.

Multi agent diagnosis, both with semantically and spatially distributed knowledge, is explained by Nico Roos et al. in [NRW03a] and [NRW03b].

There are many more interesting articles in the field of distributed diagnosis. A few more worth mentioning are [JBN05], [NRW04], [NKM02] and [Pro02].

Chapter 2

Model Based Diagnosis

This chapter is intended to give a short introduction to model based diagnosis. The framework of this chapter is in particular taken from [NF05] and [Bit05a]. For more information about model based diagnosis, the reader is referred to [NF05].

2.1 Introduction to Model Based Diagnosis

The main goal of fault diagnosis is to, based on observation and knowledge, generate a *diagnosis* D, i.e. to decide whether there is a fault or not and when there is, identify the fault. The objects for diagnosis in this thesis are in particular sensors, actuators, pipes etc. The diagnosis is computed by observing inconsistencies between observed variables and what is considered normal behavior. When the diagnosis is based on an explicit formal model of the system, the term *model based diagnosis* is used. Diagnosis can be performed both on-line and off-line.

The major purpose of this thesis is aimed for emission control in automotive vehicles but the use of diagnosis in technical processes is much wider. Some examples of what have been discussed in the literature are nuclear plants, chemical plants, gas turbines, industrial robots and most subsystems of aircrafts. The use of a diagnostic system and some of the reasons why they are incorporated are:

- Safety
- Environment Protection
- Machine Protection
- Availability
- Repairability

- Flexible Maintenance

Simple and early methods for diagnosis have been performed mainly by limit checking, e.g. sensor values are checked against thresholds. Different thresholds could be used depending on the current operating point of the system.

Another traditional approach is hardware duplication (hardware redundancy), i.e. use two or more sensors to measure the same physical quantity. This is a highly reliable method for detecting faults and often used where safety and security is a critical issue e.g. aircrafts where triple redundancy often is used. Hardware redundancy could have some drawbacks though. Hardware could be expensive, it requires extra space and the weight of the system is increased. Finally, the complexity of the system is increased when extra components are introduced.

Model based diagnosis has shown to be useful either as a complement to the methods mentioned above or by its own. The models used can be for example logic based or differential equations that describe the process. Some of the advantages of model based diagnosis are:

1. Higher diagnosis performance can be reached.
2. The possibility of isolation increases.
3. Disturbances can be taken care of.
4. Model based diagnosis is applicable to more kinds of components, i.e. where components cannot be duplicated.

When models are used to compare measured values the expression *analytical redundancy* is used. Many questions arise when engineering a diagnostic system with analytical redundancy and the problems could be solved in many different ways. The different methods will not be discussed any further in this thesis apart from the approach described in the next section.

2.2 Artificial Intelligence and Fault Diagnosis

A large amount of diagnosis methodology has been developed within the field of *Artificial Intelligence* (AI). Most of the methods belongs to a part called *consistency based diagnosis*. The objective with consistency based diagnosis is to derive a set of assignments to the components in the model, so that the model, the observations and the assignments are consistent with each other i.e. an object oriented approach with behavioral modes for each component in the system rather than a global behavioral mode for the whole system. Consistency based diagnosis is beneficially used in conjunction with model based diagnosis.

2.2.1 Behavioral Modes

Each component is assumed to be in some *behavioral mode*, e.g. normal mode (OK), the abnormal mode (AB) or some specific fault mode, e.g. (F_1), (F_2) or *unknown fault*, (UF), etc.

It is sometimes preferable to only consider the AB and the $\neg AB$ mode to reduce complexity of the diagnostic system. When only these two behavioral modes are considered and there is no model for the AB mode, the *minimal diagnosis hypothesis* (MDH) is said to hold (see Definition 2.3). Minimal diagnosis is defined in Definition 2.2.

The form when only two behavioral modes are used does not cause any problems since other fault modes could be replaced with **virtual components**, see section 2.3.3. The advantages are many. One is that the diagnostic system could be represented with a **fault mode lattice**, see Figure 2.1. When a component from a set of components in the system, $c \in \mathcal{C}$, is for example in the abnormal mode, the notation

$$mode(AB, c) \triangleq AB(c)$$

will be used.

Set Notation

When representing faulty components in consistency based diagnosis and when only the AB and the $\neg AB$ mode is considered, the *set notation* is often used. This notation replaces logical expressions with sets. The sets are used when representing both diagnoses and conflicts. The following example will illustrate the notation.

Example 2.1
If two components A and B are faulty, the diagnosis expressed in logic form will be:

$$AB(A) \wedge AB(B)$$

which can be represented by

$$\{A, B\}$$

in the set notation.

2.2.2 Diagnoses

The goal with diagnosis is to find a mode assignment, or candidate, that is consistent with the *system description* (SD) and the *observations* (OBS). The SD is a set of logical rules or a model, describing the behavior of the system. The OBS is a set of observations, e.g. sensor and actuator values. In consistency based diagnosis, the following definition of diagnosis is used:

Definition 2.1 (Diagnosis). A diagnosis is a set of components $D \subseteq \mathcal{C}$ so that

$$SD \cup OBS \cup \{ \bigwedge_{c \in D} AB(c) \ \wedge \bigwedge_{c \in \mathcal{C} \setminus D} \neg\, AB(c)\} \tag{2.1}$$

is consistent.

$\diamond$

To further reduce the complexity, only those diagnoses which are so called *minimal diagnoses* are the ones with the greatest weight and thereby those which are most considered. These diagnoses are, in principal, the ones with no “simpler” diagnoses. The definition of minimal diagnosis reads:

Definition 2.2 (Minimal Diagnosis). A diagnosis D is minimal if for all proper subsets $D' \subset D$, where D' is not a diagnosis.

$\diamond$

The interest in minimal diagnosis mainly comes from reasoning like: “If one faulty component can explain the observations, there is no reason to believe that additional components also might be faulty.” Another reason why minimal diagnoses are of interest is the fact that they sometimes are a powerful characterization (representation) of all diagnoses. This is stated in the MDH.

Definition 2.3 (Minimal Diagnosis Hypothesis, MDH). The *Minimal Diagnosis Hypothesis*, MDH, is said to hold if all supersets of each minimal diagnosis are also diagnoses.

$\diamond$

MDH does not always hold and it is not easy to formulate an exact criterion when it does. One sufficient criterion is however enacted in Lemma 2.1.

Lemma 2.1. *A sufficient condition for* MDH *is that only the AB and the $\neg AB$ mode is considered and that the AB mode has no model. Further, the two assumptions also imply that conflicts (see Definition 2.5) can only contain the $\neg AB$ mode.*

Minimal Cardinality Diagnosis

Cardinality denotes the size of a diagnosis D, i.e. how many components that are included inside the brackets in D. The basic view-point is that the most probable diagnosis is the one including the least amount of components since it is much more probable that a component is not faulty than faulty.

$$\neg AB >> AB$$

Thus, the diagnosis with the least amount of components in abnormal mode is the most probable one, i.e. it is the minimal cardinality diagnosis.

Definition 2.4 (Minimal cardinality diagnosis). Let $\mathbb{D}$ be a set of diagnoses, then the set of minimal cardinality diagnoses is

$$\mathbb{D}^{mc} = \{D \mid |D| = \min_{D \in \mathbb{D}} |D|, D \in \mathbb{D}\}$$

Where $|D|$ is the number of components included in D.

$\diamond$

2.2.3 Conflicts

Diagnoses are generally not generated directly from the model and the observations. More commonly, *conflicts* are generated from tests. Compare to structured hypothesis testing in [NF05]. A conflict is an assumption that is not consistent with the observation. It will be shown later how diagnoses can be derived from conflicts. Conflicts are generally denoted Π and defined as:

Definition 2.5 (Conflict). A conflict is a set of components $\pi \subseteq \mathcal{C}$ so that

$$SD \cup OBS \cup \{\bigwedge_{c \in \pi} \neg AB(c)\} \tag{2.2}$$

is inconsistent.

$\diamond$

Similar to case of diagnoses, *minimal conflicts* can be defined as:

Definition 2.6 (Minimal Conflict). A conflict π' is a minimal conflict if there is no proper subset

$$\pi \not\subseteq \pi'$$

where π is a conflict.

$\diamond$

The set of minimal conflicts completely characterizes all possible conflicts.

2.2.4 Relations between Diagnoses and Conflicts

There is a strong connection between diagnoses and conflicts. A diagnosis state a set of components that are faulty while a conflict state a set with components that might not have proper functionality. Diagnoses can be seen as logical implications of the set of conflicts and a useful relation between the two of them is given in Theorem 2.1.

Theorem 2.1 (Conflicts to Diagnoses)**.** *Suppose that* $\{\neg\pi_1, \neg\pi_2, \ldots\}$ *is the set of* all conflicts*. Then the mode assignment* D *is a diagnosis iff*

$$\{\neg\pi_1, \neg\pi_2, \ldots\} \bigcup D$$

is satisfiable.

When the set notation is used, it is sometimes useful to represent the diagnoses with a lattice. In section 2.3.1 an algorithm for finding the minimal diagnoses from forthcoming conflicts will be shown. The procedure is easily illustrated in such a lattice.

2.2.5 Diagnostic Tests

To detect abnormalities within the system, diagnostic tests are performed to evaluate the functionality of the system's components. In a Scania truck there exist two different kinds of tests: **Electrical tests** and **plausibility tests**. The former test single components against the valid range for the component that is being tested. For example, assume that a temperature sensor is ranged between 0.4 Volt and 4.7 Volt but the reading is outside the range. If multiple fault modes are used the test result, or sub-diagnosis, could be either "out of range high" or "out of range low".

Plausibility tests use models for the functionality of the system to detect faults. If values from sensors or actuators do not coincide with the model, a fault is present and if many tests of this kind are invalidated an isolation of the plausible faults (sub-diagnoses) will be performed.

Conflicts and Sub-diagnoses

It is not always obviously how a test result should be interpreted. When only two behavioral modes are used, i.e. AB and $\neg AB$, the result of the test could easily be interpreted as a conflict (which only states components in the $\neg AB$ mode) which easily gives the diagnosis statement. But when general fault modes are used, it is not equally easy to calculate a set of diagnoses from a set of conflicts. The conflicts still only state components in the $\neg AB$ mode, (NF). The negated conflict should state a set of diagnoses, each containing the remaining possible behavioral modes. It could therefore be more convenient to interpret the test result as a **sub-diagnosis** statement, explaining some of the possible behavioral modes of the component if the test is invalidated. There is no more information however in a sub-diagnosis statement than in a conflict statement. The one is just the compliment to the other, i.e. the negated conflict should be the sub-diagnosis statement.

Decision structure

To get an overview how the faults in the different components affect the tests, a decision structure is useful to setup. A decision structure is a table containing zeros, X:es and ones describing which test is sensitive to which fault. Here, the subject will be discussed briefly, for a more detailed explanation of decision structures, see [NF05].

	$F_1(C_1)$	$F_2(C_1)$	$F_1(C_2)$	$F_2(C_2)$
T_1	0	0	X	X
T_2	0	X	X	1
T_3	1	0	0	0
T_4	0	X	X	0

Table 2.1: Example of a decision structure for a system consisting of two components with two behavioral modes each and four tests.

A 0 in the table means that the test will not be affected by a component in that specific behavioral mode, i.e. T_i will exactly equal zero. An X means the test will sometimes be affected. A one means the test will always be affected, i.e. T_i will be nonzero.

In a typical system, test results are regularly checked, e.g. every 20 ms, and if a test is invalidated the corresponding X:es and 1:s are to become inputs to the local algorithm, generating diagnoses. In the algorithm described in the following section, no difference is made between X:es and 1:s. To use the extra information of 1:s, a different algorithm needs to be chosen. For example, consider a system with an influence structure as Table 2.1, if a diagnosis has been stated including $F_1(C_1)$ even though T_3 is not invalidated, $F_1(C_1)$ can be removed since it cannot be broken unless T_3 is invalidated.

2.3 Local Algorithms

To create a global diagnoses, local diagnoses have to be created in each ECU. The input to the local diagnostic system is a set of test results, generated by the tests belonging to the specific agent. Other inputs could be conflicts or diagnoses read from the CAN bus to be merged with the own generated conflicts or diagnoses. In section 2.3.1 however, it is only shown how minimal diagnoses are calculated from a set of test results under MDH. The following section is a slightly edited excerpt from [Jen03]

2.3.1 Reiter's Algorithm

This algorithm's task is to, given a set of conflicts (or sub-diagnoses), compute the corresponding diagnoses. MDH is assumed to hold. These computations can be done in a batch process where the diagnoses are computed when all conflicts have been found, or incrementally where the set of minimal diagnoses are incrementally refined each time a new conflict is detected.

The diagnosis computation problem is most easily illustrated using a subset-superset lattice. Figure 2.1 shows such a lattice with five components, $M1$, $M2$,

$M3, A1$ and $A2$. Each node in the lattice represents a diagnosis candidate, $[M1, M2]$ means $AB(M1) \wedge AB(M2)$ and will be written as $\{M1, M2\}$. The edges in the figure represent subset/superset relationship between candidates. The set of minimal diagnoses is incrementally computed as follows. Whenever a new conflict is detected, any previous minimal diagnosis that does not explain the new conflict is replaced by one or more superset diagnoses, which are minimal, based on this new information. This is accomplished by replacing any invalidated minimal diagnosis by a set of new candidates, each of which contains the old minimal diagnosis and one assumption from the new conflict. Note that these new candidates are diagnoses by construction. However, the new diagnoses need not be minimal. Therefore, any of the new diagnoses which is a superset of any other minimal diagnosis, or is duplicated by another, is eliminated. The remaining diagnoses are minimal and are added to the set of minimal diagnoses. This procedure is then iterated for any conflict not processed. Note that the lattice in Figure 2.1 is only used to illustrate the procedure, the algorithm do not need to represent the whole lattice. This is fortunate since the lattice grows exponentially in size with number of components.

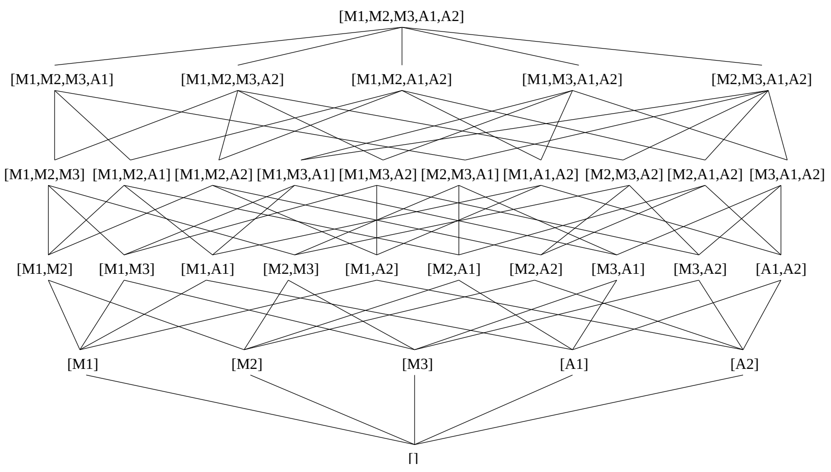

Figure 2.1: A subset/superset fault lattice with five components.

The algorithm can be summarized by the following steps:

1. Initialize the set of minimal diagnoses to hold only the empty set, i.e. $\{\{\}\}$.
2. Given a (new) conflict, find out if any minimal diagnosis is invalidated, i.e. has an empty intersection with the conflict.
3. Extend any invalidated diagnosis to a set of new diagnoses consisting

of the invalidated diagnosis and an element from the new conflict.

4. Remove any new diagnosis that are not minimal, i.e. are supersets of any other minimal diagnosis.

5. Iterate from 2 for all new conflicts.

In an ideal case where all conflicts are found and processed, the set of minimal diagnoses obtained from the algorithm equals the true set of minimal diagnoses. In reality, the set of detected conflicts is usually incomplete. This is due to the fact that when complicated structures with complicated components are considered, it is difficult to perform the local propagation in such a way that all conflicts are detected. The consequence of this incompleteness is that fewer diagnoses are invalidated than in the ideal case. It is important to note that no diagnosis will mistakenly be invalidated and eliminated which means that no erroneous diagnosis will be produced, only less specific diagnoses than in an ideal case.

2.3.2 Isolation with Generalized Fault Modes

Most AI approaches for fault isolation handle only the behavioral modes $\neg AB$ and AB. Since the components in a Scania heavy duty truck (or whatever the system is) generally can fail in more than one way, these approaches are inadequate. To isolate faults in components with general behavioral modes, a framework and an algorithm is needed. Such a framework and algorithm is presented in, among others, [Sun02]. The method presented in [Sun02] handles multiple faults and multiple fault modes.

Before the ideas behind an isolation process with general fault modes are presented a more general definition of a diagnosis is given.

Definition 2.7 (Diagnosis, general). A *diagnosis* for the system description SD and the observations OBS is a mode assignment D, for all components $c \in C$, such that

$$SD \bigcup OBS \bigcup D \tag{2.3}$$

is satisfiable.

◇

The above definition of a diagnosis does not restrict itself to only contain the $\neg AB$ and AB mode. In a similar way a conflict could be defined as:

Definition 2.8 (Conflict, general). A mode assignment π, for some subset of components, is a *conflict* if

$$SD \bigcup OBS \bigcup \pi \tag{2.4}$$

is not satisfiable.

◇

Assumption Based Diagnostics

The idea behind the method presented in [Sun02] is to have a number of sub-models, each with a corresponding assumption. The assumptions are logical expressions that state something about the behavioral modes of the components in the system that is being diagnosed. From the sub-models, test quantities can be derived to test whether the assumptions hold or not. If the test is in the rejection region, the assumption is rejected, i.e. the null hypothesis, H_0, is rejected and the sub-model is invalidated.

$$assM \rightarrow M \rightarrow T \in R^C \iff T \in R \rightarrow \neg\, M \rightarrow \neg\, assM$$

Since a submodel may produce reasonable values even if the assumption does not hold, no conclusions can be drawn if H_0 is not rejected.

$$T \in R^C \nrightarrow assM$$

The assumption that is rejected constitutes a conflict, $\neg\, assM$. To calculate the diagnoses, or candidates, the conflict is negated and evaluated. More details about the evaluation could be found in [Sun02].

Since some of the sub-models often are fault models, Lemma 2.1 is not fulfilled. Thus MDH does not necessarily holds. If the diagnosis statement should be *complete* a larger representation of the statement may be needed than if only the minimal diagnoses were considered under MDH. This is further exploited in [JdKR92].

2.3.3 Virtual Components

Reiter's algorithm described in section 2.3.1 is valid for components with only two operating modes, i.e. AB and $\neg AB$ mode. This algorithm could be extended to work with generalized fault modes by introducing **virtual components**. It is shown in [Jen03] that there is a significant gain in performance using virtual components compared to the method presented in section 2.3.2.

The first step is to map all fault modes of all components into virtual components. Table 2.2 shows an example of such a mapping.

Component behavioral mode		Virtual component
$F_1(C_1)$	$\rightarrow$	V_1
$F_1(C_2)$	$\rightarrow$	V_2
$F_2(C_2)$	$\rightarrow$	V_3
$UF(C_2)$	$\rightarrow$	$V_2 \wedge V_3$

Table 2.2: The mapping between component operating modes and virtual components.

This conversion could be done in advance so that no processing power is taken from the ECU. When the test results have been converted to a set of virtual components the algorithm described in 2.3.1 is utilized. Since it is difficult to interpret the diagnoses when it is represented using virtual components there must also be a conversion back to real components and behavioral mode assignments.

Example 2.2
A system consisting of two components with behavioral modes mapped to virtual components according to Table 2.2. The system receives the following test result, i.e. sub-diagnosis.

$$< F_1(C_1) >$$

To work with Reiter's algorithm the test result is converted to virtual components. The corresponding test result is

$$< F_1(C_1) >=< V_1 >$$

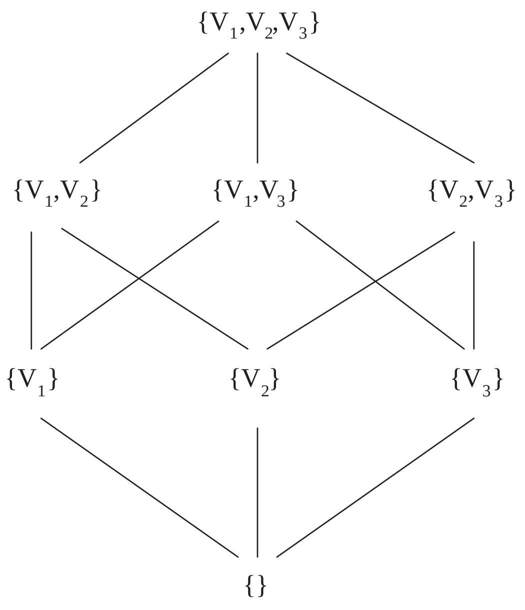

Figure 2.2: Lattice for a system with three virtual components.

Reiter's algorithm can now be used to process the test result, producing the diagnosis

$$\{V_1\}$$

Corresponding to the behavioral mode

$$\{F_1(C_1)\}$$

This is represented by line 1 in Figure 2.3. All nodes above the line are valid diagnoses since MDH holds, but $\{V_1\}$ is the minimal diagnoses. Now assume the following test result arrives

$$< F_1(C_2) >$$

This test result is converted to the corresponding virtual component.

$$< F_1(C_2) >=< V_2 >$$

Inserting this into Reiter's algorithm generates line 2 in Figure 2.3. The minimal diagnosis is now:

$$\{V_1, V_2\}$$

Corresponding to the behavioral mode diagnosis

$$\{F_1(C_1), F_1(C_2)\}$$

This is the correct diagnosis of the system. Note that a node containing two behavioral modes of the same component is translated to behavioral mode UF for that component.

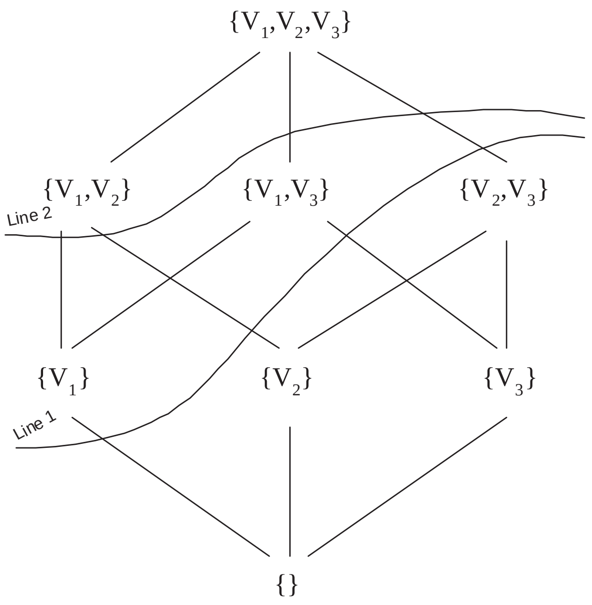

Figure 2.3: Lattice for three components with line 1 corresponding to test result $< F_1(C_1) >$ and line 2 corresponding to test result $< F_1(C_2) >$.

Chapter 3

Distributed Systems

This chapter is intended as a brief introduction to distributed systems. The network in a Scania truck, see Figure 4.1, consisting of many different ECUs, falls inside the definition of a distributed system. So to get a better understanding of the network from a distributed system point of view, the basic terminology is here presented and discussed. Most of the facts presented below are taken from [TvS02].

3.1 Properties of Distributed Systems

Within the field of distributed systems there are a few important goals that should be met when designing a system. These are transparency, openness and scalability, which are further explained below.

3.1.1 Transparency

A distributed system that is able to present itself to users and applications as if it were only a single computer system is said to be **transparent**. There exists different kinds of transparency, and the concept of transparency can be applied to several aspects of a distributed system, as shown in Table 3.1.

For the distributed system considered in this report, the failure transparency is the one of most interest since the whole purpose of the diagnostic system is to detect faults among the components being diagnosed, making it not failure transparent. On the other hand, if an ECU fails, the system should function as good as possible anyway, deleting the faulty ECU from the diagnostic system. Thus, it is important to be able to distinguish between failure transparency concerning the components and failure transparency concerning the ECUs.

Also, there is a trade-off between a high degree of transparency and the performance of a system. For example, if one of the ECUs are trying repeat-

Transparency	**Description**
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource might move to another location
Relocation	Hide that a resource might be moved while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource might be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Table 3.1: Different forms of transparency for distributed systems.

edly to transmit information to other ECUs, to hide an error in an ECU, but fails, it could have been more efficient to give up earlier.

3.1.2 Openness

An open distributed system offers services according to certain rules in syntax and semantics of those services. It is important to have a well defined interface with a specification of which names are available with which types of parameters, return values and so on. Proper specifications are complete and neutral. Complete means that everything that is necessary for connecting to the interface has indeed been specified. Neutral means that each object to be connected to the distributed system can be implemented in any way as long as it complies with rules for that specific interface.

If a system can function and communicate, inside the specification of the interface, even though parts have been supplied from different manufacturers it is said to have a high degree of **interoperability**. A second definition is **portability** which characterizes a system that runs applications on distributed system A, without modification, considering that they were developed for system B, assuming system B use the same interface as system A.

If a system is both interoperable and portable it is said to be flexible, meaning that it is easy to add new components or replace existing ones without affecting those components that stay in place.

It is preferable if the distributed system in a Scania truck is flexible making it easy to add, remove or change ECUs in future models.

3.1.3 Scalability

In a scalable system the size can be changed without making any big changes in hardware and software. Considering the fast development of technology, it is easy to understand that it is critical in the design of a distributed system to make it scalable. For example, it is highly reasonable to assume that the network of ECUs in today's Scania trucks will develop further, adding more and more processing units to the network, and therefore requiring it to be scalable.

A traditional centralized system, where the processing units transmit requests of communication with the central unit, is much less scalable than a distributed system where the different processing units share the load. The former creates an information bottleneck that prohibits further growth. Therefore, only distributed algorithms should be used. These generally have the following characteristics, which distinguish them from centralized algorithms:

1. No machine has complete information about the system state.
2. Machines make decisions based only on local information.
3. Failure of one machine does not ruin the algorithm.
4. There is no implicit assumption that a global clock exists.

When implementing a distributed diagnostic system, scalability becomes a central issue since storing diagnostic information, received from other ECUs, require memory and as more ECUs are added to the system, more memory needs to be allocated in each ECU. This scalability issue will be discussed further in chapter 6.

3.2 Hardware Concepts

There exist different models on how the hardware in a distributed system can be configured. The multiple *Processing Elements* (PEs) can either be connected via bus or switch. If it is bus-based, there is a single backbone with the different elements connected to it. In a switch-based system there are individual wires from machine to machine where the messages move along with an explicit switching decision made at each step to route the message along one of the outgoing wires.

How the memory is connected can also be classified into two groups. It can either be shared, which is usually denoted **multiprocessors** (Figure 3.1), or private, denoted **multicomputer** (Figure 3.2), for each PE.

A benefit of having a multiprocessor network is the smooth and efficient handling of memory. For example, the scenario of one PE having plenty of memory available while other PEs having none cannot arise. The downside of the multiprocessor system is all the traffic on the wires/bus when the PEs want

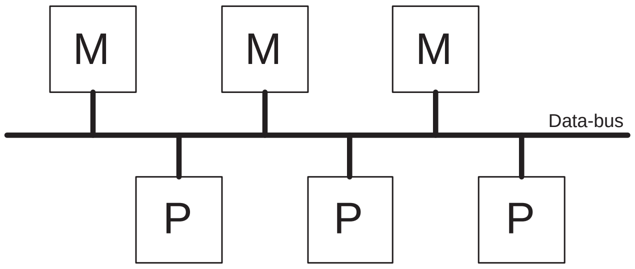

Figure 3.1: A bus-based multiprocessor system, P for processor and M for memory.

to collect information from the memory units. Further, a distinction can be made between multicomputer systems: **homogeneous** and **heterogeneous**. Homogeneous is, as the name reveals, a set of CPUs with the same kind of technology that usually have access to same amount of memory and therefore making them easy to interconnect. Heterogeneous, on the other hand, is a multicomputer system consisting of different, independent computers, which in turn are connected through different networks. Following from earlier definitions: a homogeneous network is not as flexible as a heterogeneous system where one can connect a machine that is different in technology but can still be part of the distributed system, if it uses the same interface, see above.

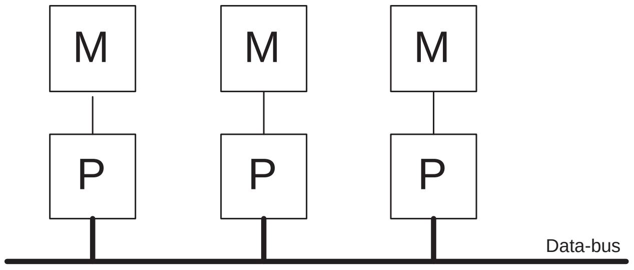

Figure 3.2: A bus-based multicomputer system, P for processor and M for memory.

The setup of hardware in today's Scania trucks is a typical bus-based heterogeneous multicomputer system, see Figure 4.1, with ECUs of different speed and memory size.

3.2.1 The CAN Bus

The network implemented in the distributed system in today's Scania trucks is a *Controller Area Network* (CAN). It was originally designed for the automotive industry but is today used in a wide field of applications. CAN enables

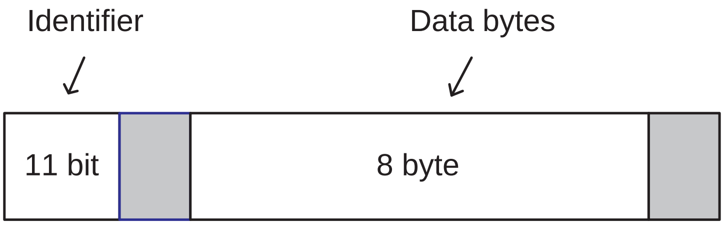

Figure 3.3: A CAN package. The shaded areas represent checksums and other control bits.

a huge reduction in wiring complexity compared to dedicated links for connection between the different ECUs.

One feature of CAN that suits distributed diagnosis particularly well is the option of multicast or peer-to-peer communication. Multicast means that information can be sent to a subset of receivers and peer-to-peer is communication one to one. The local diagnoses calculated by an ECU probably needs to be shared with one or many other ECUs, requiring peer-to-peer and multicast. When data is transmitted on the bus, no particular ECU is addressed. The message is sent with an identifier, leaving it up to the receivers to accept the message or not. This concept has become known in the networking world as the producer/consumer mechanism, whereby one node produces data on the bus for the other nodes to consume [MFB99]. Apart from data and the identifier, the message also contains various control bits and checksums, baked into one CAN package, see Figure 3.3,

The data transmitted is 8 byte, which is not always enough for diagnostic messages meaning that more than one package may need to be sent.

Chapter 4

Distributed Diagnostic Systems

In chapter 2, model based diagnosis was discussed and in chapter 3 distributed systems in general were discussed. In this chapter the two areas are linked together to build a theory on distributed diagnosis for embedded systems.

4.1 The Network Architecture

ECUs are typically connected via a CAN bus, see section 3.2.1. Figure 4.1 shows such a network used in current Scania heavy duty vehicles. It includes three separate CAN buses: red, yellow and green. The buses are connected by the *Coordinator* (COO). The COO acts like a router, making sure that no messages are exchanged between the buses unless it is necessary. There are between 20 and 30 ECUs in a typical Scania system, depending on the truck's type and outfit. Between 4 and 110 components are connected to each ECU. The ECUs' CPUs have typically a clocking speed of 8 to 64 MHz and a memory capacity of 4 to 150 kB [JBN05].

There are several reasons why the ECUs need to exchange information between each other over a network. Some of these are:

- A component can be used by multiple ECUs.
- A component does not necessarily have to be connected to the ECU that is controlling it.
- Diagnosis is performed on components by multiple ECUs.

Since multiple ECUs can use and perform diagnosis on the same component it is also important that they can inform each other whether the component is working or not. A method for sharing this type of information is presented in this thesis.

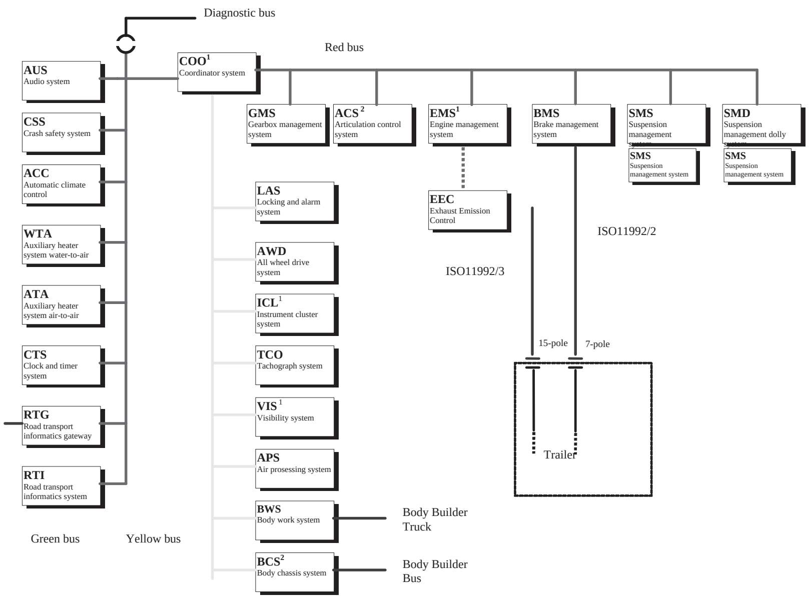
Diagnostic bus
Red bus
COO¹ Coordinator system
AUS Audio system
CSS Crash safety system
ACC Automatic climate control
WTA Auxiliary heater system water-to-air
ATA Auxiliary heater system air-to-air
CTS Clock and timer system
RTG Road transport informatics gateway
RTI Road transport informatics system
GMS Gearbox management system
ACS² Articulation control system
EMS¹ Engine management system
BMS Brake management system
SMS Suspension management system
SMD Suspension management dolly system
SMS Suspension management system
SMS Suspension management system
EEC Exhaust Emission Control
ISO11992/2
ISO11992/3
LAS Locking and alarm system
AWD All wheel drive system
ICL¹ Instrument cluster system
TCO Tachograph system
VIS¹ Visibility system
APS Air prosessing system
BWS Body work system
BCS² Body chassis system
15-pole
7-pole
Trailer
Body Builder Truck
Body Builder Bus
Green bus
Yellow bus

Figure 4.1: The network and ECU topology in a Scania heavy duty truck.

4.2 Current Diagnostic System

Each ECU performs on-line diagnosis. The current diagnostic system consists of tests which compares one or several components against a threshold value. The current Scania diagnostic system includes between 10 and 1000 diagnostic tests in each ECU. If a test result is outside the boundary set by the threshold, the test assumption is invalidated. Afterwards, when the tests are either validated or invalidated, the *Diagnostic Manager* (DIMA), calculates the *minimal diagnoses* from the generated sub-diagnoses. An isolation process follows were the minimal cardinality diagnoses, see Definition 2.4, are selected and every component that is represented in these diagnoses is assigned a DTC[1]. All components represented in the minimal cardinality diagnoses are presented to the technician at the workshop as **suspected** by the DTC. If a component is included in all minimal cardinality diagnoses, then it is presented as **confirmed** by the DTC. The process to derive DTCs is illustrated in figure 4.2. DTCs are only presented by those ECUs that owns the specific component (see Definition 4.1), i.e. an ECU cannot present a DTC belonging to another ECU.

[1]All components in the diagnoses are either in the AB or $\neg AB$ mode, i.e. virtual components are used for those with several behavioral modes.

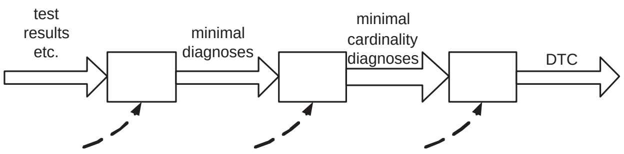

Figure 4.2: A simplified flowchart of the diagnosis procedure. The dashed arrows indicate where distributed diagnostic information might come in.

4.2.1 The Goal with the Distributed Diagnostic System

In this section the objective of this thesis, explained in section 1.2, is applied to the system of a Scania truck. Since the DTCs are the final result of the diagnostic system, the objective should concern DTCs. Therefore, the objective in section 1.2 applied to the diagnostic system in a Scania truck is:

> A DTC assigned to each component that does not contradict with the DTC assigned w.r.t. the global minimal cardinality diagnoses.

If the DTC is the same as the one generated from the global minimal cardinality diagnoses, the DTC is said to be globally consistent.

What this means in practice is that when all the necessary diagnostic information is processed and distributed, the resulting DTCs should be the same as those generated at the end of the flowchart in Figure 4.2 if the test results from all agents were put in at the beginning of the chart, i.e. the DTCs should be globally consistent.

Note that the global diagnoses does not necessarily set more components in a confirmed mode than the local diagnoses. It could just as well degrade components that are confirmed locally to be suspected globally. Consider Example 4.2 where agent A_1 should present a DTC for the component A as confirmed but the globally, and thus the correct, DTC for component A should only be suspected.

One question that now arises is if it is globally correct to set a component in the suspected mode, even though it globally should be either in the confirmed mode or perhaps not have a DTC at all? That depends on how one defines the term *globally correct* DTC. If it means that the result should be globally consistent, then it is not correct to suspect a component that should not be suspected, but if a globally correct DTC means that no contradictions exist with the global diagnoses, then it could be OK to set a component in the suspected mode if reasonable motives exists. This makes it harder to know which component or components that are the true faulty ones, but on the other hand it could decrease the work for the local diagnostic system to calculate the diagnoses.

4.3 Components, Signals and Objects

A diagnostic system involves a set of agents, $\mathcal{A}$, connected by the CAN bus. An agent is a piece of software in each ECU that handles the calculation and communication of diagnoses. An output signal in an agent is linked to input signals in one or several agents. Further, the diagnostic system consists of a set of objects, see Definition 4.5, which is a subset of the total number of components for the global system. The objects for a certain agent $A \in \mathcal{A}$ are diagnosed for abnormal behavior.

Each agent includes a number of tests. The objects Θ, which are analyzed for abnormal behavior by the tests, can have different origins and have different properties. It will be shown later that it becomes important to distinguish these types of different objects, hereafter referred to as signals and components. One could classify two different types of components and two types of signals to be analyzed by a certain agent. Components and signals will be diagnosed in the same way in the ECU. Here an explanation of each type of component and signal will be introduced.

A component is either private or common.

Definition 4.1 (Private Component). A private component is a component that is physically connected to an agent. It is clear which agent that owns and controls the private component. A private component is denoted $p \in P$, where P is all private components in the system.

◇

Definition 4.2 (Common Component). A common component, G, is a component that is physically connected to several agents or a component that is not connected directly to any agent and who's owner is uncertain e.g. pipes, links or other mechanical devices.

◇

The common component is a special type of component that currently cannot be found in the Scania diagnostic system. It will be assumed that whenever this type of component is added to the diagnostic system, it will be assigned an owner and treated as a **private component** by the owning agent. A common component is denoted $g \in G$, where G is all common components in the system.

Definition 4.3 (Input signal). An input signal, γ, is received from CAN. Several agents can read the same signal as long as it is distributed on the network. An input signal is denoted $\gamma \in \Gamma$, where Γ is all input signal in the system.

◇

This type of signal is similar to the output signal defined below. An input signal is read from CAN and diagnosed in the same way as components by the diagnostic system. The signal value could be dependent on one or more components, e.g. a sensor or an actuator, but it could also be an estimated or calculated value. It will be discussed later if it is necessary for the diagnostic system to know all information about the origin from the input.

Definition 4.4 (Output Component). This type of signals are the values distributed on the CAN bus. The signal can be derived from one or several physical components. It could also be estimated from some other form of data. An output signal is denoted $\sigma \in \Sigma$, where Σ are all output signals in the system.

◇

As for the input signal, the output signal value could be dependent on one or more components, e.g. a sensor or an actuator, but it can also be an estimated or calculated value.

Note that a signal can be of numerous types at the same time for the whole system, e.g. a sensor connected to two agents where one of the agents distributes its value on the CAN bus, the component would be a type as those defined in definitions 4.2, 4.3 and 4.4 at the same time from a system point of view.

When different types of components and signals have been explained, it is possible to define objects, which are the signals and components included in its local diagnostic system, for an agent.

Definition 4.5 (Objects). The set of objects for an agent, A, is

$$\Theta = P^A \cup \Gamma^A \cup G^A \cup \Sigma^A$$

where P^A is a set of private components, Γ^A is a set of input signals, G^A is a set of common components and Σ^A is a set of output signals. The output signals are special cases since they are based on the diagnoses of the private components.

◇

The objects are different for each agent. Example 4.1 explains components, signals and objects further.

Example 4.1
Consider Figure 4.3 where a system consisting of three agents is illustrated. Each agent have a set of tests and the objects for agent A_1 is $\Theta_1 = \{F, S_1, S_2\}$, the objects for agent A_2 is $\Theta_2 = \{A, B, E, G, H, S_2\}$ and the objects for agent A_3 is $\Theta_3 = \{C, D, S_1\}$. The classification of components and signals are as follows.

Agent 1 Component F is a private component; S_1 and S_2 are input signals.

Agent 2 Component A, B, E, G and H are private components, S_1 is an output signal and S_2 is an input signal.

Agent 3 Component C and D are private components, S_1 is an input signal and S_2 is an output signal.

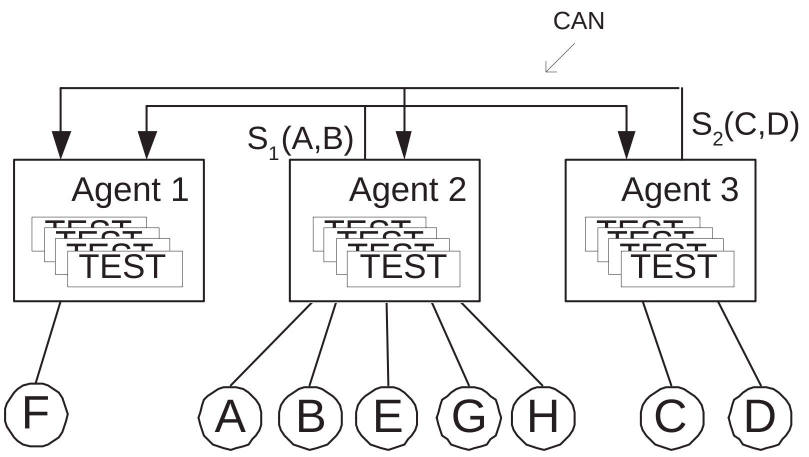

Figure 4.3: Agents with components A to G and signal S_1, depending on component A and B, and signal S_2 depending on component C and D.

4.4 Signals - Inputs and Outputs

Some reasoning about signals, i.e. inputs and outputs, and their characteristics will here be presented. The discussion will be concerning a few basic questions:

1. Is it necessary for a receiving agent to know about the origin of an input signal?
2. Should a transmitting agent treat the output signal as a special component in its own diagnostic system?
3. How is the cardinality of a diagnosis affected when signals are included in the diagnosis?

The transmitting agent is the agent distributing values on the CAN bus and the receiving agent is the one reading the value.

Let us start with the first question. Assume that an agent calculates an output based on the functionality of three private components. Using CAN, there is no way for the receiving agent to know which components the signal depends on, unless an **initialization** process is performed. In the initialization process each signal and which components it depends on would be declared, enabling the agents to transform signals to corresponding component representation. Is this necessary though?

The receiving agent cannot set DTCs on components owned by the transmitting agent, so it is enough to diagnose with a signal representation and then share the information of the diagnoses stated. When the agent, where the signal originated from, receives the diagnoses it recognizes the signal as one of its outputs and transforms it to a component representation since DTCs are not set on signals but on components. Therefore, without an initialization phase, the agents still have enough information to set globally consistent DTCs. For the receiving agent, the cardinality of the object would also always be one, because it cannot distinguish which of the three physical components that caused the problem. And by this the third question is also answered.

Regarding the second question, there is no reason for the output signal to be diagnosed as a signal in the transmitting agent instead of diagnosing the private components and from this determine which output signals that are diagnosed. In the case where one would like to share information about diagnoses that affect output signals, there is always a way to derive that kind of information regardless if the signal is part of the diagnostic system or not, since each ECU knows which components its output is dependent on. Also, if the output signals would be included as components in the diagnostic system, one would have to compensate for the cardinality in the diagnoses where the signal is present.

The conclusion of the reasoning above is that no initialization process is needed in order to exchange information regarding the origins of input signals and that the cardinality of the resulting diagnoses is not affected. It could also be concluded that the output signal should not be a part of the local diagnostic system.

4.5 Local and Global Diagnosis

Considering the network of ECUs in today's Scania trucks, shown in Figure 4.1, and how the components are linked to the different ECUs, see Figure 1.1, it is possible to define two different types of fault diagnosis for the system. First local diagnosis, where each agent state a set of diagnoses about its objects, without sharing any information with other agents. Since no diagnostic information is shared the local diagnoses can be incomplete. The second type is global diagnosis where all the test results of the system is considered when generating the diagnoses.

As mentioned before, the DTCs in the system should be set based on globally consistent diagnoses. Hence, the ECUs need to exchange diagnostic information to form the globally consistent diagnoses. In the process of sharing information the merge operator is used, the definition follows:

Definition 4.6 (Merge). Let $\mathbb{D}^1$ and $\mathbb{D}^2$ be two sets of diagnoses, then a merge of these diagnoses is the set of minimal sets

$$\mathbb{D}^1 \uplus \mathbb{D}^2 = min_s(D^1 \cup D^2 \mid D^1 \in \mathbb{D}^1, D^2 \in \mathbb{D}^2)$$

◇

From the definition of merge follows that the global diagnoses is a merge of the local diagnosis from each agent.

Theorem 4.1 (From local diagnoses to global diagnoses [Bit05a])**.** *For each $A \in \mathcal{A}$, let $\mathbb{D}^A$ be a set of local diagnoses consistent with the conflicts Π^A, then the minimal global diagnoses is*

$$\mathcal{D} = \biguplus_{A \in \mathcal{A}} \mathbb{D}^A$$

In short, if the local diagnoses for each agent is known then a merge of these generates the global diagnoses.

Example 4.2
Consider two agents holding the set of conflicts

$$\Pi^{A_1} = \{\{A, B\}, \{A, C\}\} \qquad \Pi^{A_2} = \{\{B, D\}\}$$

With the corresponding diagnoses

$$\mathbb{D}^{A_1} = \{\{A\}, \{B, C\}\} \qquad \mathbb{D}^{A_2} = \{\{B\}, \{D\}\}$$

To create the global diagnosis, the two local diagnoses are merged, resulting in the set

$$\mathbb{D}^{A_1} \uplus \mathbb{D}^{A_2} = \{\{A, B\}, \{A, D\}, \{B, C\}\}$$

Note, the non-minimal diagnosis $\{B, C, D\}$ is not included in the global diagnosis. Notice also that each diagnosis is consistent with every conflict, thus, every merged diagnosis is a global diagnosis.

4.5.1 Two Ways of Calculating the Global Diagnosis

One can distinguish between two different ways of calculating the global diagnoses. The conflicts generated from the different tests in each agent can either be transformed into local diagnoses and then merged to form the global diagnoses, Figure 4.4, or by first merging all the local conflicts and then generating the global diagnosis from the set of all conflicts, Figure 4.5. These different approaches are the basics of the first two methods described in the next chapter.

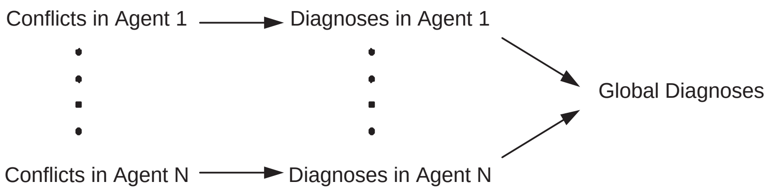

Figure 4.4: Generating global diagnoses from local conflicts to local diagnoses to global diagnosis.

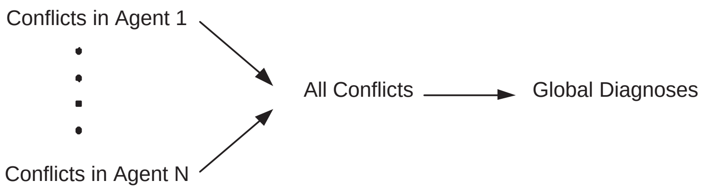

Figure 4.5: Generating global diagnosis from local conflicts to all conflicts to global diagnosis.

4.5.2 The Combinatorial Problem

A problem that arises in distributed diagnosis is the size of the global diagnoses that are generated by the merge of the local diagnoses. The number of global diagnoses grows exponentially with both the number and the size of the local diagnoses. This leads to a combinatorial explosion if many faults, generating many and large diagnoses, occur. A solution that seems reasonable is to only merge the diagnoses that are most probable i.e. to exclude the diagnoses in each agent that are least probable.

One way of calculating the probability of a specific diagnosis is to assign a probability to each fault mode. Normally, the no-fault mode is assigned the highest probability, i.e. it is more probable that a component is functioning correctly than incorrectly. The various fault modes have lower probability.

$$P(NF) >> P(F1), P(F2), \ldots, P(Fn)$$

The problem is to assign probabilities to the different fault modes. For example, in the case of a Scania truck certain probabilities of a fault when the truck is just produced will certainly change over time when the truck is used. Therefore, a simpler approach is desirable. The different fault modes can be assumed to have the same probability, enabling the use of minimal cardinality.

$$P(NF) >> P(F1) = P(F2) = \ldots = P(Fn)$$

The set of minimal cardinality diagnoses is usually smaller than the set of minimal diagnoses. Thus, the minimal cardinality diagnoses can be used to reduce the combinatorial explosion that occurs when several diagnoses are merged together.

Other approaches of probabilistic reasoning in fault isolation besides minimal cardinality reasoning exist. One could be found in [AP05] where the utilization of bayesian networks in fault isolation is explored.

Note, for components with more than two behavioral modes, minimal cardinality diagnosis only holds if the fault modes have an equal probability. Example 4.3 will highlight the implications of this.

Example 4.3

Consider a system with three components, all with two behavioral modes AB or $\neg AB$. The probability of AB is 0.01 for all three components. If all faults are assumed to occur independently the minimal cardinality diagnosis is the most probable. For example:

$$P(\{C_1\}) = 0.01$$
$$P(\{C_2, C_3\}) = 0.0001$$

If component C_1 has four fault modes, F_1, F_2, F_3, UF they are assumed to all have the same probability in order for minimal cardinality to be applicable. The probability of UF when all fault modes have probability 0.01 is (again, all faults occur independently):

$$\begin{aligned} P(UF) &= P(F_1, F_2) + P(F_1, F_3) + P(F_2, F_3) + P(F_1, F_2, F_3) \\ &= 0.0001 + 0.0001 + 0.0001 + 0.000001 = 0.000301 \end{aligned} \quad (4.1)$$

This probability is much lower than the probability of the other fault modes and therefore either the faults are dependent, for example

$$P(F_1, F_2) = P(F_1) \times P(F_2|F_1) \quad \text{where} \quad P(F_2|F_1) > P(F_2)$$

making the sum of probabilities (4.1) bigger or there are possibilities of faults not modeled, P_{unknown}, that need to be considered, i.e.

$$P(UF) = P(F_1, F_2) + P(F_1, F_3) + P(F_2, F_3) + P(F_1, F_2, F_3) + P_{\text{unknown}}$$

It could also be a combination of dependency between behavioral modes and unmodeled faults.

4.5.3 Merging Minimal Cardinality Diagnoses

Earlier it was shown how the global diagnoses could be generated from the merge of all local diagnoses, Theorem 4.1. Is this also true for minimal cardinality diagnoses? Unfortunately not. Sets of local minimal cardinality diagnoses cannot be merged together to form the global minimal cardinality diagnoses, i.e.

$$\mathcal{D}^{mc} \neq \biguplus_{A \in \mathcal{A}}^{\times} \mathbb{D}_A^{mc}$$

Here is an example to prove it. Note, in the following examples, to make it understandable, a complete component representation is assumed, meaning all ECUs know about all components of the system.

Example 4.4
Consider Example 4.2 with the minimal cardinality diagnoses $\mathbb{D}_{A_1}^{mc} = \{\{A\}\}$ and $\mathbb{D}_{A_2}^{mc} = \{\{B\}, \{D\}\}$. Then the merge results in

$$\mathbb{D}_{A_1}^{mc} \uplus^{\times} \mathbb{D}_{A_2}^{mc} = \{\{A, B\}, \{A, D\}\}$$

While

$$\mathcal{D}^{mc} = \{\{A, B\}, \{A, D\}, \{B, C\}\}$$

The global minimal cardinality diagnosis $\{B, C\}$ was not included in the merge of minimal cardinality local diagnoses.

The reason is that not all agents are independent of each other. Many agents run tests including some other agent's components, and thus the agent's local diagnoses might include signals that depends on some other agent's component. A solution, presented by [JBN05], is to first group the agents into modules, where each module of agents is independent of each other, as shown in the following example.

Example 4.5
If $\mathbb{D}_1 = \{\{A, B\}\}$, $\mathbb{D}_2 = \{\{B, C\}\}$, and $\mathbb{D}_3 = \{\{E\}\}$, then for the modules

$\bar{\mathcal{A}}_1 = \{A_1, A_2\}$ and $\bar{\mathcal{A}}_2 = \{A_3\}$, it follows that $\mathcal{D}_1^{mod} = \{\{A, B, C\}\}$ and $\mathcal{D}_2^{mod} = \{\{E\}\}$

A module of agents with diagnoses independent of each other can form a *Module Minimal Cardinality Diagnosis* (MMCD) denoted $\mathcal{D}_i^{mod,mc}$ for the i:th module. If all these MMCDs are merged, it can be proved ([Bit05a]) that

$$\mathcal{D}^{mc} = \biguplus_{\times} \mathcal{D}_i^{mod,mc}$$

Hence, grouping the agents into modules, merging the diagnosis inside the modules and finding the minimal cardinality diagnosis for each module to reduce the combinatorial problem, and last, merging the MMCDs, generates the minimal cardinality global diagnosis.

4.6 Centralized or Distributed Diagnosis

In general, there are three different ways to organize a diagnostic system working over a network. In a Scania truck the ECUs can either transmit all their data to a central unit, here called diagnostic agent, that performs tests and states the diagnoses. This setup is denoted centralized diagnostic system.

A different approach is to let the agents in the ECUs state their own local object diagnoses and then transmit their results to a centralized diagnostic agent who would merge the different local diagnoses. The advantage of this approach, denoted decentralized diagnosis, is the distribution of work creating the diagnosis in each agent instead of in a central unit. Still, a decentralized solution is in need of a central unit for the merge of the local diagnosis.

A preferable method would be to make the diagnostic system fully distributed with no need of any central unit. The agents would then have to state their own local diagnosis and then transmit diagnostic information between each other to generate a globally consistent diagnosis without the need of a central unit.

4.6.1 Centralized Diagnosis and Decentralized Diagnosis

An advantage of a centralized diagnostic system is the simplicity of it. No calculation needs to be done at local level and since the global diagnoses stated by the diagnostic agent is based on all the diagnostic information of the system, global consistency is always achieved. The communication on the CAN bus will be directed in only one way, from the ECUs to the central unit. A basic diagram of a centralized system is shown in Figure 4.6.

A disadvantage of centralized diagnosis is the scalability. A diagnostic system using one central diagnostic unit has a limit on how many ECUs that can be connected since it has a finite amount of processing power and memory. Thus, it would need changes in the hardware if the system expanded

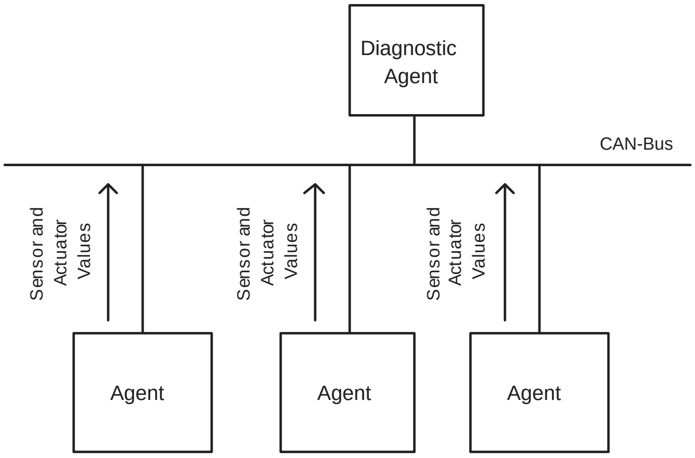
Diagnostic Agent
CAN-Bus
Sensor and Actuator Values
Sensor and Actuator Values
Sensor and Actuator Values
Agent
Agent
Agent

Figure 4.6: A centralized diagnostic system.

outside of the central unit's limits, e.g. faster processor to speed up the calculation of diagnoses from the increasing amount of information.

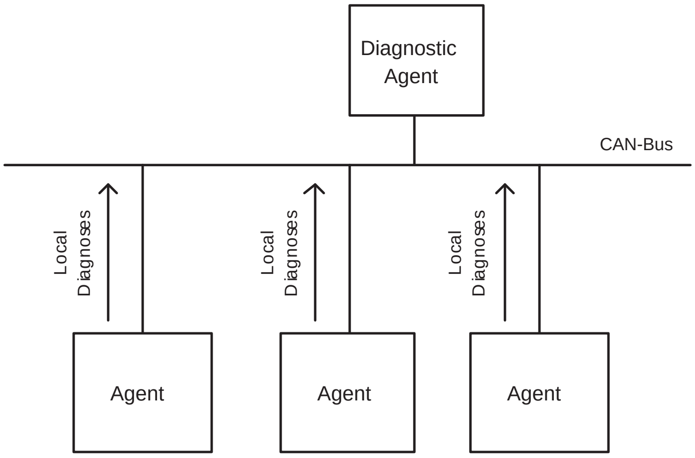

Figure 4.7: A decentralized diagnostic system.

A decentralized diagnostic system is in many ways similar to a centralized system. It is diagnoses that are transmitted from the local agents to the diagnostic agent, instead of sensor and actuator values, see Figure 4.7. The system cannot be considered scalable though, because it is still quite computationally intensive to merge the local diagnoses sent to the central unit. To increase the scalability, the merge could be processed in the agents with the central unit as coordinator agent instructing the agents how to merge their local diagnoses in between each other. Such a solution would require a more advanced algorithm, see [JBN05]. Another flaw of the centralized and decentralized method is the failure transparency from a distributed systems point of view. If the central unit fails, it is difficult, or impossible, for the other ECUs to hide that failure.

4.6.2 Distributed Diagnosis

In a distributed diagnostic system, see Figure 4.8, sharing diagnoses between ECUs without the use of a central unit is both scalable and failure transparent. For example, if one ECU stops working then the other agents will form the diagnosis for the rest of the system. The computations are shared between the agents, so for every ECU that is added not just the amount of diagnoses increases but also the computational power. The communication is distributed in the network of ECUs, so adding more ECUs adds traffic, but not in any specific part of the CAN bus.

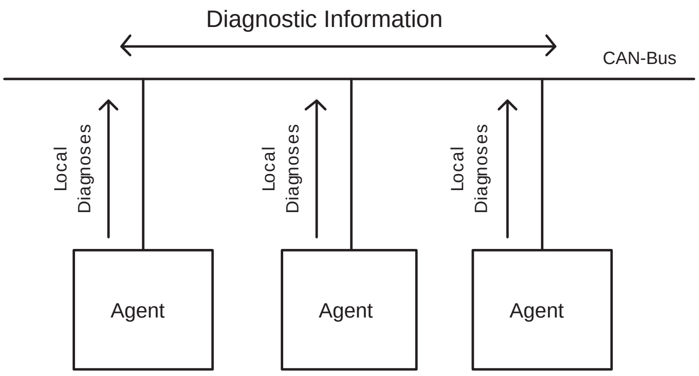

Figure 4.8: A distributed diagnostic system.

The drawback of this method is the complexity of the implementation. In centralized and decentralized diagnosis the agents transmit diagnostic information only to the diagnostic agent, but in this approach a more advanced method of communication is required because the agents exchange information between each other. Chapter 5 presents a few methods, suggesting how to implement distributed diagnosis.

4.7 Sharing Diagnostic Information

As discussed earlier, the different ECUs are dependent on each other. Agent A_1 needs to know if a component, that it controls, but connected to Agent A_2, is broken. It is possible that the diagnostic tests in A_1 does not respond to an error on a certain component that it is depending on, but the tests in A_2 does or that A_1 cannot isolate which component is broken on its own but it can with the help of the tests in A_2. This strongly motivates a diagnostic system that shares information. Every agent wants the best possible diagnoses it can have of both its own, its shared and its common components. Maybe the calculation of the best possible diagnoses are not feasible, taking in to account all the information that needs to be shared and the size of the local diagnoses generated. There is a trade-off between hardware usage and how good the generated diagnoses will be. Less information is transmitted to the price of worse diagnoses. Still though, the diagnoses need to be globally consistent to accomplish the goal, see section 4.2.1. The question is what to share and how to do it in order to generate feasible and good enough diagnoses that complies with the goal.

4.7.1 Sharing Conflicts

The diagnostic tests performed in each agent can generate a set of conflicts. These conflicts can be sent to other agents that are interested in the condition of the components included in these sets. The receiving agent can then insert this conflict into its algorithm to generate diagnoses, resulting in a more complete set than its local diagnoses including only diagnostic information from its own tests. Example 4.6 gives an example of two agents sharing conflicts; π^{tx} denotes transmitted conflicts, π^{rx} denotes received conflicts and $\overset{min}{\simeq}$ is the minimal diagnosis operator.

Example 4.6
Two agents in a system has found the following conflicts

$$\Pi^{A_1} = \{A, B, C\} \qquad \Pi^{A_2} = \{i, F\}$$

With the corresponding diagnoses:

$$\mathbb{D}^{A_1} = \{\{A\}, \{B\}, \{C\}\} \qquad \mathbb{D}^{A_2} = \{\{i\}, \{F\}\}$$

Component B make up the input signal i in agent A_2. Thus, A_2 transmits its conflict including signal i on CAN as $\pi^{tx} = \{i, F\}$. Agent A_1 receives this and translates signal i to its dependency, component B, $\pi^{rx} = \{B, F\}$. The received conflict results in extended diagnoses for A_1

$$\begin{aligned}\mathbb{D}^{A_1} &= \{\{A, B\}, \{A, F\}, \{B\}, \{B, F\}, \{B, C\}, \{C, F\}\} \\ &\overset{min}{\simeq} \{\{A, F\}, \{B\}, \{C, F\}\}\end{aligned}$$

Note, Agent A_1 has new and more complete, globally consistent diagnoses. Hence, the diagnoses have been improved by sharing conflicts.

The final diagnoses that are generated when sharing conflicts are the correct global diagnoses. The conflicts hold direct information about the result of the diagnostic tests performed in the agent and, thus, sharing conflicts is equivalent to adding the tests in the transmitting agent to the tests already present in the receiving agent. This is the same thing as looking upon the system as a centralized diagnostic system, see section 4.6, and therefore the result is globally consistent.

4.7.2 Sharing Diagnoses

A similar approach as used above for sharing conflicts can be used also when sharing diagnoses. Diagnostic information is in both cases sent and received between agents and, hence, more complete local object diagnoses can be formed. The local diagnoses must be available in the different ECUs. In the receiving agent a merge would be performed between the received diagnoses with the one already stated locally. It was in Theorem 4.1 shown that the result will be globally consistent. Example 4.7 gives an example of two agents sharing diagnoses; $\mathbb{D}^{tx}$ denotes a set of transmitted diagnoses, $\mathbb{D}^{rx}$ denotes a set of received diagnoses and $\overset{min}{\simeq}$ is the minimal diagnosis operator.

Example 4.7
Consider the local diagnoses in Example 4.6

$$\mathbb{D}^{A_1} = \{\{A\},\{B\},\{C\}\} \qquad \mathbb{D}^{A_2} = \{\{i\},\{F\}\}$$

A sharing of diagnoses from A_2 results in $\mathbb{D}^{tx} = \{\{i\},\{F\}\}$ which will be received by A_1 and translated to $\mathbb{D}^{rx} = \{\{B\},\{F\}\}$ and merged with its own diagnoses to form

$$\begin{aligned}\mathbb{D}^{A_1} &= \{\{A,B\},\{A,F\},\{B\},\{B,F\},\{B,C\},\{C,F\}\}\\ &\overset{min}{\simeq} \{\{A,F\},\{B\},\{C,F\}\}\end{aligned}$$

This, naturally, is the same result as in Example 4.6.

Does all the diagnostic information in the agents have to be shared or could it be enough to transmit only certain diagnoses or conflicts and in that way reduce the CAN traffic and calculation time for the ECUs? Maybe not all information is of interest for the receiving agent. In the section below, this matter will be discussed more thoroughly.

4.7.3 The Information to Share

A receiving agent, A, is only interested in diagnoses or conflicts available on the CAN bus that does not have an empty intersection with its own objects, i.e. $\Theta^A \cap D^{rx} \neq \emptyset$, since the other diagnoses does not affect the receiving agent's functionality. Thus, it may be unnecessary for an agent to share conflicts or diagnoses consisting of only private components.

The method gives different results when sharing conflicts or sharing diagnoses. Conflicts are, as discussed above, directly generated from tests and sharing them is like adding, in the receiving agent, the tests that generated the shared conflicts, making the diagnoses stated in the receiving agent globally consistent. Tests in the transmitting agent including only private components, not included in any output signal, do not add any information in the receiving agent. When sharing diagnoses, on the other hand, consistency becomes a problem if not all diagnoses are shared. This difference between conflicts and diagnoses can be seen in Examples 4.8 and 4.9.

Example 4.8
Consider the conflicts in the following agents:

$$\Pi^{A_1} = \{A, i\} \qquad \Pi^{A_2} = \{\}$$

i is an input signal. Agent A_1 shares the conflict $\pi^{tx} = \{\{A, i\}\}$ and agent A_2 receives $\pi^{tx} = \{\{A, C\}\}$, since the input signal, i, is connected to an output signal in A_2 which depends only on component C. This gives the diagnoses

$$\mathbb{D}^{A_2} = \{\{A\}, \{C\}\}$$

The DTCs are set as suspected, thus globally consistent.

Example 4.9
Same scenario as in previous example, but this time sharing diagnosis

$$\mathbb{D}^{A_1} = \{\{A\}, \{i\}\} \qquad \mathbb{D}^{A_2} = \{\{\}\}$$

Agent A_1 shares only the input diagnosis since A is a private component not included in any output signal. $\mathbb{D}^{tx} = \{i\}$ and agent A_2 receives $\mathbb{D}^{rx} = \{C\}$, which gives

$$\mathbb{D}^{A_1} = \{\{A\}, \{i\}\} \qquad \mathbb{D}^{A_2} = \{\{C\}\}$$

Since $\{\{C\}\}$ is the only diagnosis present in A_2, the DTC confirmed will be set, which is not globally consistent.

A significant problem with the method above is that depending on the cardinality of the diagnoses including only private components, i.e. those not shared, is that in some scenarios could the receiving agent confirm a component, which it owns, broken that is not, see Example 4.9. Thus, making it globally inconsistent and not complying with the goal.

The inconsistency is generated when the cardinality of the not shared diagnoses present in A_1 are disregarded. Information about cardinality needs to be present since it is upon information about cardinality that the DTC is set.

4.7.4 Focusing on Probable Diagnosis

Instead of selecting which diagnostic information to share based on type of components or signals it could be based upon probability. The following example uses such an approach.

Example 4.10
Agent A_1 has stated the following local diagnoses

$$\mathbb{D}^{A_1} = \{\{i_1, i_2, A\}, \{B\}, \{C, i_1\}, \{A, C, D\}\}$$

Using cardinality diagnoses, see Definition 2.4, assuming that it is much higher probability that a component X is in $\neg AB(X)$ mode than in $AB(X)$, one can easily conclude that the most probable diagnosis is $\{B\}$. Knowing that, is it then necessary to share the other diagnoses including inputs and/or outputs? That, again, depends how one chooses to trade off hardware usage to diagnostic quality.

As seen in Example 4.4, merging diagnoses based on probability, i.e. cardinality, can generate a statement that is inconsistent with the global diagnoses. To meet the goals of this project, see page 27, all the diagnoses with cardinality lower or equal to the minimal cardinality of the merged product need to be shared. If the cardinality is higher than this number it will not be represented in the minimal cardinality diagnoses, because a diagnosis with a specific cardinality cannot by merging generate diagnoses with lower cardinality. Thus these diagnoses do not need to be shared. All the other ones, i.e. the diagnoses with cardinality lower or equal to the cardinality of the minimal cardinality diagnoses of the merged product need to be shared to ensure global consistency.

Unfortunately, knowing the minimal cardinality of the merged product is not a simple task. It could be done by first performing a pre-merge of the diagnoses to be merged, and then calculate the minimal cardinality of the result. This approach would take away the whole purpose, though, since a pre-merge, in fact, demands the same workload as a normal merge unless only some diagnoses are pre-merged, but then, which ones should be merged. A different approach could be to sum all minimal cardinalities and use this as an upper limit on the global minimal cardinality.

Module Diagnosis

If it is decided that all different component types should be shared, then all agents in a module would calculate the same diagnosis, the module diagnosis. Instead of calculating this in every agent it can be done in a more efficient way, presented in [JBN05], where the authors present an algorithm dealing with this. The basic idea is to first divide the agents into independent modules and then merge the local diagnoses in these agents in an order where the complexity is minimized. The module diagnoses build up from agent to agent and the last one to have its diagnosis merged will also store the module diagnosis. If the global minimal cardinality diagnoses are requested, then a merge can be performed between the module minimal cardinality diagnoses, see page 36.

4.7.5 Problems with Component Representation

The diagnoses or conflicts that an agent receives often include components belonging to another agent and therefore unknown to the receiving agent. How is the agent supposed handle these components? Either, every agent knows about every component in the whole system (at least all components in its module) or the unknown components included in the received diagnoses can be added dynamically. Still, an agent cannot change the diagnostic status i.e. none, suspected or confirmed on other agent's components and extra components mean extra memory and processor usage. Therefore, the best solution would be if agents only stored information about components that are its own.

Chapter 5

Proposed Methods for Distributed Diagnosis

On basis of the discussions in the previous chapter, a more formal model was created. Based on this model, methods for sharing diagnostic information were constructed and are here presented. Algorithms are presented for both the transmitting agent who transmits the diagnostic information and the receiving agent who receives it.

5.1 Model for Distributed Diagnosis

The objects that could be assigned with a DTC is always components, never signals. Signals can however be part of a diagnosis.

Components

There are a few different types of components. There are a set of **private components**, $P^A \subseteq \mathcal{C}$, with the following property:

$$\forall i \neq j : \quad P^{A_i} \cap P^{A_j} = \emptyset \tag{5.1}$$

There are a set of **common components**, $G \subseteq \mathcal{C}$, with the following property:

$$\forall A : \quad G \cap P^A = \emptyset \tag{5.2}$$

Signals

In the set of output signals Σ, which are all signals on the CAN bus, an output signal is denoted $\sigma \in \Sigma$. The set of input signals is Γ. An input signal is denoted $\gamma \in \Gamma$. The set of signals are disjoint from the set of components and an output signal's dependency is a subset of private components and input signals. An output signal is in other words created from the values of private components and input signals. The creation results in a new object, $\sigma \in \Sigma$.

$$\Sigma \cap \mathcal{C} = \emptyset \tag{5.3}$$

$$dep(\sigma) \subseteq P^A \cup \Gamma^A \tag{5.4}$$

An output signal, σ, from an agent can be connected to several input signals. An input signal can however never be connected to several output signals. Further, the function $con(x)$ is the *connection function* extracting which input an output is connected to and vice versa.

$$con(\sigma) \subseteq \Gamma \tag{5.5}$$

$$con(\gamma) \in \Sigma \tag{5.6}$$

$$\forall i \neq j : \quad con(\sigma_i) \cap con(\sigma_j) = \emptyset \tag{5.7}$$

The set of input signals is disjoint from the set of components and the dependency of an input is the dependency from its origin.

$$\Gamma \cap \mathcal{C} = \emptyset \tag{5.8}$$

$$dep(\gamma) = dep(con(\gamma)) \tag{5.9}$$

Limitations and Assumptions

The methods presented below are all developed under some assumptions, or limitations. They are all correct under these circumstances but the intention is to remove the need for these assumptions in the future methods.

The assumptions are:

$$\forall \sigma \in \Sigma^A : \quad dep(\sigma) \subseteq P^A \tag{5.10}$$

$$\forall\, \sigma_i, \sigma_j \in \Sigma^A, i \neq j : \quad dep(\sigma_i) \cap dep(\sigma_j) = \emptyset \tag{5.11}$$

An output signal from an agent A can only depend on the agent's private components, compare with equation (5.4), and two or more output signals can never depend on the same private components. These limitations will be discussed further in section 5.3.

If one takes a closer look at equation (5.4) one notes that a common component, $g \in G$, cannot be a part of any signal. This assumption is due to logical and physical reasons and has no need to be removed in future methods.

5.2 Algorithms for Distributed Diagnosis

Based on the model in section 5.1, a number of different methods for distributing diagnostic information were developed and some of them are presented here.

The first method is based on the sharing of conflicts and the second method is based on the sharing of diagnoses. The methods are much alike and both demand that the agents can handle component information belonging to other agents. The third method is a more advanced extension of the second method where the problem with component representation in other agents is solved. The diagnostic information transmitted over CAN is also reduced. This method is the main contributing method for distributed diagnosis presented in this thesis.

As said above, not all ideas that came up during the project are presented here. Some of the ideas were to have approximate methods, i.e. to set DTCs that are correct but perhaps not consistent with DTCs based on a set of globally consistent diagnoses. The conclusions were however that there is no need for approximate approaches. Another approach was to have a similar method to the one presented in [JBN05] but the conclusion was that the method was too complicated and not really suitable to be implemented in a Scania network system.

When calculating diagnoses, the need for a function similar to the dependency function arose. This function of a signal $s \in \Sigma$, called the *dependent diagnoses function*, $dpd(s)$, returns a set of sets. Each set represent a diagnosis consisting of one or several components from the dependency of the signal s.

$$\begin{aligned} dpd_i(s) &\in dpd(s) \\ \bigcup_i dpd_i(s) &= dep(s) \end{aligned} \tag{5.12}$$

Example 5.1 will clarify the use of the dependent diagnoses function.

Example 5.1
Consider Figure 5.1 where S_1 and S_2 represent two output signals in the system. Assume that Agent A_1 has two conflicts saying that none of its inputs can be in the $\neg AB$ mode. Agent A_1s set of diagnoses is therefore:

$$D^{A_1} = \{\{S_1, S_2\}\}$$

where

$$\{S_1, S_2\} \subseteq \Gamma^{A_1}$$

Assume that the diagnosis is transmitted to agent A_2 and A_3. Agent A_2 receives the diagnosis and detects:

$$S_1 \cap \Sigma^{A_2} \neq \emptyset$$

$$S_2 \cap \Gamma^{A_2} \neq \emptyset$$

The diagnoses to be merged with the original diagnoses in A_2 is thus:

$$\mathbb{D}^{A_2}_{merge} = \{\{dpd(S_1)\}, \{S_2\}\} = \{\{A\}, \{B\}, \{S_2\}\}$$

Agent A_3 receives the same diagnosis and detects:

$$S_1 \cap \Gamma^{A_3} \neq \emptyset$$

$$S_2 \cap \Sigma^{A_3} \neq \emptyset$$

The diagnoses to be merged with the original diagnoses in A_3 are thus:

$$\mathbb{D}^{A_3}_{merge} = \{\{S_1\}, \{dpd(S_2)\}\} = \{\{S_1\}, \{C\}, \{D\}\}$$

5.2.1 Method 1: Sharing Conflicts

As previously seen under section 4.7.1, globally consistent diagnoses can be generated by sharing conflicts and from these globally consistent DTCs can be assigned.

Goal: By sharing conflicts, generate a set of diagnoses in each agent that globally consistent DTCs can be assigned based upon.

Algorithm for the Transmitting Agent

Algorithm 1 Method 1 - Transmitting agent

```
Require: Minimal local conflicts $\Pi^A$ in each agent A
Ensure: Conflicts to transmit, $\Pi^{tx}$, including all information needed for the
    receiving agent to set globally consistent DTCs
 1: $\Pi^{tx} := \emptyset$
 2: for all $\pi \in \Pi^A$ do
 3:     for all $\sigma \in \Sigma^A$ do
 4:         if $dep(\sigma) \cap \pi \neq \emptyset$ then
 5:             $\pi := \sigma \cup (\pi \setminus dep(\sigma))$
 6:         end if
 7:     end for
 8:     $\Pi^{tx} := \Pi^{tx} \cup \pi$
 9: end for
10: Transmit $\Pi^{tx}$ on the CAN bus.
```

Line 1 Initiate the conflicts to be transmitted as an empty set.

Line 2-7 Substitute the components that the output signals are dependent on, for output components, representing the output signals. Add, by iteration, the conflicts to be transmitted to the set Π^{tx}.
Comment: The components in the transmitting agent that make up an output signal is represented in the receiving agent as an input component. Thus, if any of these components are diagnosed in the transmitting agent, it is the same thing as the input signal for the receiving agent is diagnosed.

Line 8 Add, by iteration, the conflicts to be transmitted to the set Π^{tx}.

Line 9 Transmit Π^{tx} on the CAN bus.

Algorithm for the Receiving Agent

Algorithm 2 Method 1 - Receiving agent

```
Require: $\Pi^{tx}$
Ensure: Globally consistent DTCs on its own components
 1: $\Pi^{rx} := \emptyset$
 2: for all $\pi \in \Pi^{tx}$ do
 3:     $outputdep := \emptyset$
 4:     for all $c \in \pi$ do
 5:         if $c \cap P^A = \emptyset$ then
 6:             set flag for $c$ that no DTC should be set
 7:         end if
 8:     end for
 9:     for all $s \in \pi$ do
10:         if $s \in \Sigma^A$ then
11:             $outputdep := outputdep \cup\!\!\!\!\times\ dpd(s))$
12:             $\pi := \pi \setminus s$
13:         end if
14:     end for
15:     $\Pi^{rx} := \Pi^{rx} \cup (outputdep \cup \pi)$
16: end for
17: call the algorithm for generating diagnoses with $\Pi^{rx}$ as input
```

Line 1 Initiate the conflicts to be transmitted as an empty set.

Line 4-8 Mark which components that should not be assigned a DTC.
Comment: An agent can only set DTCs for its own components.

Line 9-13 For all received conflicts: merge all output component dependencies iteratively and add with the conflict minus the received input component.

Comment: A received input component that is included in the receiving agent's set of output components can depend on many components. A conflict including this type of component generates new conflicts in the receiving agent diagnoses need to be merged to form the correct diagnoses for the receiving agent.

Line 15 Add, iteratively, the conflicts received to the set Π^{rx}.

Line 17 Insert the received conflicts in the local algorithm for generating diagnoses.

Result: A set of diagnoses for each agent, consistent with the global minimal cardinality diagnoses, $\mathcal{D}^{mc}$. Based upon this set, globally consistent DTCs can be assigned. It has previously in this thesis been shown how sharing of all conflicts, on component level, generates the global diagnoses, therefore the DTCs assigned will be globally consistent. A component, included in the transmitted conflicts, that is private to the transmitting agent needs to be marked by the receiving agent, since no DTC should be set for this component by the receiving agent.

5.2.2 Method 2: Sharing Diagnoses

Similarly to sharing conflicts it was in the previous chapter discussed how exchanging diagnoses could generate globally consistent diagnoses and from this globally consistent DTCs can be assigned.

Goal: To generate a set of diagnoses that globally consistent DTCs can be assigned based upon.

Algorithm for the Transmitting Agent

Algorithm 3 Method 2 - Transmitting agent

```
Require: Minimal local diagnoses $\mathbb{D}^A$
Ensure: Diagnoses to transmit, $\mathbb{D}^{tx}$, including all information needed for the
    receiving to set globally consistent DTCs
 1: $\mathbb{D}^{tx} := \emptyset$
 2: for all $D \in \mathbb{D}^{tx}$ do
 3:     for all $\sigma \in \Sigma^A$ do
 4:         if $dep(\sigma) \cap D \neq \emptyset$ then
 5:             $D = \sigma \cup (D \setminus dep(\sigma))$
 6:         end if
 7:     end for
 8:     $\mathbb{D}^{tx} := \mathbb{D}^{tx} \cup D$
 9: end for
10: Transmit $\mathbb{D}^{tx}$ on the CAN bus.
```

Line 1 Initiate the diagnoses to be transmitted as an empty set.

Line 2-7 Substitute the components that the output signals are dependent on for output components, representing the output signals.
Comment: The components in the transmitting agent that make up an output signal is represented in the receiving agent as an input component. Thus, if any of these components are diagnosed in the transmitting agent, it is the same thing as the input signal for the receiving agent is diagnosed.

Line 8 Add, iteratively, the conflicts to be transmitted to the set Π^{tx}.

Line 10 Transmit $\mathbb{D}^{tx}$ on the CAN bus.

Algorithm for the Receiving Agent

Algorithm 4 Method 2 - Receiving agent

```
Require: $\mathbb{D}^{tx}$
Ensure: Globally consistent DTCs on its own components
 1: $\mathbb{D}^{rx} := \emptyset$
 2: for all $D \in \mathbb{D}^{rx}$ do
 3:   $outputdep := \emptyset$
 4:   for all $c \in D$ do
 5:     if $c \cap P^A = \emptyset$ then
 6:       set flag for $c$ that no DTC should be set
 7:     end if
 8:   end for
 9:   for all $s \in D$ do
10:     if $s \in \Sigma^A$ then
11:       $outputdep := outputdep \uplus dpd(s))$
12:       $D := D \setminus s$
13:     end if
14:   end for
15:   $\mathbb{D}^{rx} := \mathbb{D}^{rx} \cup (outputdep \cup D)$
16: end for
17: $\mathbb{D}^A := \mathbb{D}^A \uplus \mathbb{D}^{rx}$
```

Line 1 Initiate the diagnoses to be received as an empty set.

Line 4-8 Mark which components that should not be assigned a DTC.
Comment: An agent can only set DTCs for its own components.

Line 9-13 For all received diagnoses: merge all output component dependencies iteratively and add with the diagnoses minus the received input

component.
Comment: A received input component that is included in the receiving agent's set of output components can depend on many components. A diagnosis including this type of component generate new diagnoses in the receiving agent diagnoses need to be merged to form the correct diagnoses for the receiving agent.

Line 15 Add, iteratively, the diagnoses received to the set D^{rx}.

Line 17 Merge the local diagnoses with the received one to form the new, enhanced local diagnoses.

Result: A set of diagnoses for each agent, consistent with the global minimal cardinality diagnoses, $\mathcal{D}^{mc}$. Based upon this set, globally consistent DTCs can be assigned. It has previously in this thesis been shown how sharing of all diagnoses generate the global diagnoses, therefore the DTCs assigned will be globally consistent. A component, included in the transmitted diagnoses, that is private to the transmitting agent needs to be marked by the receiving agent, since no DTC should be set for this component by the receiving agent.

5.2.3 Method 3: Sharing Diagnoses Extended

Not all components need to be shared, as in Method 2, it is enough with selected information about cardinality for the diagnoses in the transmitting agent. The receiving agent can with the help of this information draw correct conclusions of which components that are confirmed or suspected. For a mathematical proof of the method, see Appendix A. An example of the method is shown in Example 5.2.

Goal: To generate a set of diagnoses for each agent that globally consistent DTCs can be assigned based upon.

Algorithm for the Transmitting Agent

Algorithm 5 Method 3 - Transmitting agent

```
Require: Minimal local diagnoses $\mathbb{D}^A$
Ensure: Diagnoses to transmit, $\mathbb{D}^{tx}$, including all information needed for the
    receiving agent to set globally consistent DTCs
 1: $\mathbb{D}^{tx} = \emptyset$
 2: for all $D \in \mathbb{D}^A$ do
 3:     for all $\sigma \in \Sigma^A$ do
 4:         if $dep(\sigma) \cap D \neq \emptyset$ then
 5:             $D = \sigma \cup D$
 6:         end if
 7:     end for
 8:     $X = |D \cap P^A| - |D \cap \Sigma^A|$
 9:     $D = D \setminus (P^A \cap D)$
10:     $\mathbb{D}^{tx} = \mathbb{D}^{tx} \cup \{D, X\}$
11: end for
12: $Y = \min(|D_1|, \ldots, |D_m|)$ for all $D_i \in (\mathbb{D}^A \setminus \mathbb{D}^{tx})$
13: $\mathbb{D}^{tx} = \{\mathbb{D}^{tx}, Y\}$
```

Line 1 Initiate the diagnoses to be transmitted as an empty set.

Line 2-7 Substitute the components that the output signals are dependent on for output components, representing the output signals.
Comment: The components in the transmitting agent that make up an output signal is represented in the receiving agent as an input component. Thus, if any of these components are diagnosed in the transmitting agent, it is the same thing as the input signal for the receiving agent is diagnosed.

Line 8,9 Set a cardinality variable X_i which equals the cardinality for the diagnosis pruned of all signals minus the cardinality of the outputs. Delete all private components from the diagnosis.
Comment: In the merge in the receiving agent the resulting cardinality will be depending on the cardinality of the transmitted diagnoses. Since the private components will not be sent, information about the cardinality needs to be present.

Line 11 Store the minimal cardinality of the diagnoses consisting of only private components as Y.

Comment: In the merge of diagnoses consisting of only private components the cardinality of the product will be the sum of the cardinalities of the diagnoses to be merged. Since the decision of which components that are suspected or confirmed broken is based on minimal cardinality, the minimal cardinality needs to be sent too.

Line 12 Transmit the diagnoses including inputs or outputs pruned on all private components but with the corresponding Xs and Y.
Comment: This package could, for example, look like
$\mathbb{D}^{tx} = \{\{\gamma, X_1 = 3\}, \{\gamma, \sigma, X_2 = 4\}, Y = 2\}$

Algorithm for the Receiving Agent

Algorithm 6 Method 3 - Receiving agent

```
Require: $\mathbb{D}^{tx}$
Ensure: Globally consistent DTCs on its own components
 1: $\mathbb{D}^{rx} = \emptyset$
 2: for all $D \in \mathbb{D}^{rx}$ do
 3:   $outputdep = \emptyset$
 4:   for all $s \in D$ do
 5:     if $s \cap (\Gamma^A \cup \Sigma^A) = \emptyset$ then
 6:       $X = X + 1$
 7:       $D = D \setminus s$
 8:     else if $s \cap \Sigma^A \neq \emptyset$ then
 9:       $outputdep = outputdep \uplus dpd(s)$
10:       $D = D \setminus s$
11:     end if
12:   end for
13:   $\mathbb{D}^{rx} = \mathbb{D}^{rx} \cup (outputdep \uplus D)$
14: end for
15: $\mathbb{D}^A = \mathbb{D}^A \uplus \mathbb{D}^{rx} = \{\{D^A \cup D^{rx}, X\} : D^A \in \mathbb{D}^A, \{D^{rx}, X\} \in \mathbb{D}^{rx}\} \cup$
    $\{\{D^A, Y\} : D^A \in \mathbb{D}^A\}$
```

Line 1,3 Initiate the diagnoses to be received and the output dependency diagnoses as empty sets.

Line 4-13 For all received signals in the diagnosis: check if the signal is an output or input in the receiving agent. If it is neither, add one to X for that diagnosis and delete the signal from the diagnosis. If it is an output, then merge all output component dependencies iteratively and add with the diagnoses minus the received input component. If it is an input, nothing needs to be done.
Comment: An agent does not diagnose components or signals that it

does not know. If a signal that it does not know is included in the diagnoses it should be deleted but also the cardinality needs to be compensated for. A signal that is included in the receiving agent's set of output components can depend on many components. A diagnosis including this type of component generates new diagnoses in the receiving agent and these need to be merged to form the correct diagnoses for the receiving agent.

Line 18 Merge $\mathbb{D}^A$ and $\mathbb{D}^{rx}$ to generate the diagnoses from where the globally consistent DTCs can be set.
Comment: In the merge X_i and Y need to be compensated for. The X_is are added to the cardinality of the merges between D^A and D_i^{rx} and united with D^A where Y is added to the cardinality of D^A

Result: Set of diagnoses for each agent, that globally consistent DTCs can be assigned based upon. It has earlier in this thesis been shown that a merge of all diagnoses generates the global diagnosis. The DTCs are to be set based on the global minimal cardinality diagnoses. In this method not all diagnoses are merged, but since cardinality information about the diagnoses that could affect the generated minimal cardinality diagnoses is shared, consistent DTCs can be set anyway.

Example 5.2
Consider Figure 5.1 again. Here signals are denoted with a unique symbol S_i which is equivalent with σ_i for the transmitting agent and γ_i for the receiving agent. Assume that each agent has its corresponding set of diagnoses according to:

$$\mathbb{D}_1 = \{\{F, S_2\}, \{S_1, S_2\}\}$$
$$\mathbb{D}_2 = \{\{A, E\}, \{S_2, E\}, \{S_2, B\}, \{S_2, A, G\}, \{H\}\}$$
$$\mathbb{D}_3 = \{\{\}\}$$

A_1 and A_2 have both diagnosed signals (outputs or inputs) and therefore these agents transmit their diagnoses on the CAN bus.

$\mathbb{D}_1^{tx}$: $\{F, S_2\}$ consists of one private component and one signal which translates into $\{\{S_2\}, 1\}$. $\{S_1, S_2\}$ consists of two signals and translates into $\{\{S_1, S_2\}, 0\}$. There is no diagnoses consisting of only private components, therefore Y=0. This results in:

$$\mathbb{D}_1^{tx} = \{\{\{S_2\}, 1\}, \{\{S_1, S_2\}, 0\}, 0\}$$

$\mathbb{D}_2^{tx}$: $\{A, E\}$ consists of one output component in signal S_1 and one private component, which translates into $\{\{S_1\}, 1\}$. $\{S_2, E\}$ consists of one signal and one private component which translates into $\{\{S_2\}, 1\}$. $\{S_2, B\}$

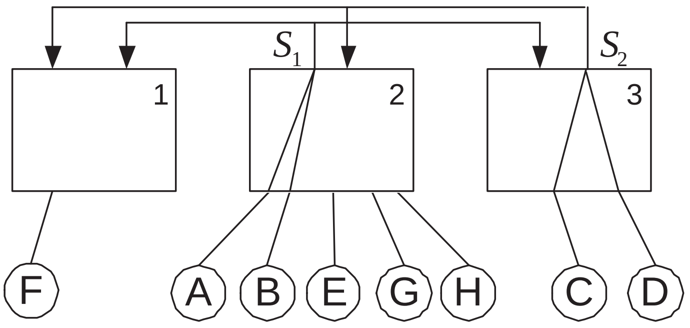

Figure 5.1: Three agents. Signals are denoted S_i.

consists of one signal and one output component in signal S_1, which translates into $\{\{S_1, S_2\}, 0\}$. $\{S_2, A, G\}$ consists of one signal, one output component and one private component which translates into $\{\{S_1, S_2\}, 1\}$. There is only one diagnosis consisting of only private components and its cardinality is 1, therefore $Y = 1$. This results in:

$$\mathbb{D}_2^{tx} = \{\{\{S_1\}, 1\}, \{\{S_2\}, 1\}, \{\{S_1, S_2\}, 0\}, \{\{S_1, S_2\}, 1\}, 1\}$$

A_1 receives diagnoses from A_2. A_1 has no output so $\mathbb{D}_1^{rx,2} = \mathbb{D}_2^{tx}$

$$\mathbb{D}_1^{rx,2} = \{\{\{S_1\}, 1\}, \{\{S_2\}, 1\}, \{\{S_1, S_2\}, 0\}, \{\{S_1, S_2\}, 1\}, 1\}$$
$$\mathbb{D}_1 \uplus \mathbb{D}_1^{rx,2} = \{\{S_2, F\}, 1\}, \{\{S_1, S_2\}, 0\}$$

A_1 has no own components among its minimal cardinality diagnoses, thus no DTCs will be set.

A_2 receives diagnoses from A_1. S_1 is an output signal from A_2 so the signal will be transformed into its dependency components.

$$\mathbb{D}_2^{rx,1} = \{\{\{S_2\}, 1\}, \{\{A, S_2\}, 0\}, \{\{B, S_2\}, 0\}, 0\}$$
$$\mathbb{D}_2 \uplus \mathbb{D}_2^{rx,1} = \{\{S_2, A, E\}, 0\}, \{\{S_2, B\}, 0\}, \{\{S_2, E\}, 1\}, \{\{S_2, A, G\}, 0\}, \{\{S_2, H\}, 1\}$$

B is among the components in the minimal cardinality diagnoses, thus B will be confirmed broken.

A_3 receives diagnoses from A_1 and A_2. S_2 is an output signal from A_3

so the signal will be transformed into its dependency components.

$$\begin{aligned}
\mathbb{D}_3^{rx,1} &= \{\{\{C\},1\},\{\{D\},1\},\{\{S_1,C\},0\},\{\{S_1,D\},0\} \\
\mathbb{D}_3^{rx,2} &= \{\{\{S_1\},1\},\{\{C\},1\},\{\{D\},1\},\{\{S_1,C\},0\},\{\{S_1,D\},0\}, \\
&\qquad \{\{S_1,C\},1\},\{\{S_1,D\},1\},1\} \\
\mathbb{D}_3 \uplus \mathbb{D}_3^{rx,1} \uplus \mathbb{D}_3^{rx,2} &= \{\{\{C\},2\},\{\{D\},2\},\{\{C,D\},2\},\{\{S_1,C\},0\}, \\
&\qquad \{\{S_1,D\},0\}\}
\end{aligned}$$

C and D are represented in one minimal cardinality diagnoses each, thus both will be suspected.

The global merge of the local diagnoses gives in rising order of cardinality:

$$\mathcal{D} = \mathbb{D}_1 \uplus \mathbb{D}_2 = \{\{B,C\},\{B,D\},\{F,C,H\},\{F,D,H\}\ldots\}$$

B is confirmed broken and C and D are suspected which consistent with the result above in each agent.

5.3 Discussion Concerning the Limitations and Assumptions

In section 5.1 a few limitations and assumptions were introduced. Two of these will here be discussed more thoroughly, namely equation (5.10) and equation (5.11). They are both quite strong assumptions and before implementing the methods in a real system it would be preferable if they could be erased. The reason for their being is, however, a problem not so easy to solve. We have in all models assumed that no initializing phase is needed, so that when an ECU receives a signal it does not know the component dependency of it, just the value. A component could be included in the dependency of two different signals that an agent diagnose and cause problems when determining the cardinality. Example 5.3 will clarify the problem.

Example 5.3
Consider the system of Figure 5.2. The output signals from agent 1, S_1 and S_2, are both dependent on component B. Agent 2 states the following diagnoses:

$$\mathbb{D}^{A_2} = \{\{S_1,S_2\},\{C\}\}$$

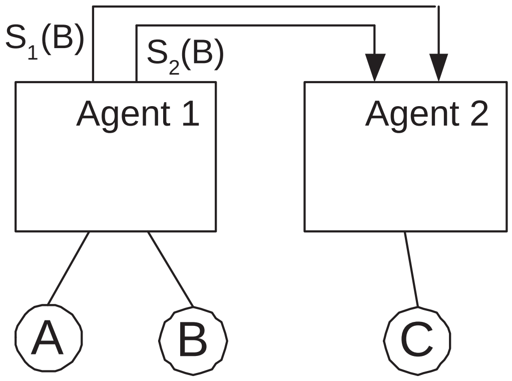

Figure 5.2: An example setup where one component is represented in two signals.

Agent 2 would then set the DTC confirmed on component C which is not the correct diagnosis since $\{S_1, S_2\}$ really has cardinality equal to one. Component C should be suspected, not confirmed. Note, the given example is simplified to highlight the problem with signals. In reality it would be unlikely for two signals to depend on only one, and the same, component.

This problem can be found not only in the setup of Example 5.3, i.e. when a component is included in two output signals. Also when the input from one ECU is included in the output of another. Both the input signal and the output signal could be inputs to a third ECU and the same component would then be included in two different causing, again, a problem with cardinality. It is in both cases the same cause of the problem: a component included in two different signal's dependencies. The receiving agent can not detect this and therefore the cardinality may be calculated to equal an incorrect value.

To solve the problem, the ECUs need to have information about the signals and their dependencies. One way could be to either have an initialization phase where, at start-up, all ECUs exchange information about each others signals and which components they depend on. When diagnosing a signal, an ECU could then determine which components it is dependent on and from this information conclude the correct cardinality of its diagnoses.

A second solution would be to have a pre-map of all the signals and their component dependencies before connecting them to the system. The result would be the same as above but no initialization would be needed since the ECUs at start-up already would have the correct information about the signals and their dependencies. This would make the system less scalable and flexible since when an ECU is added to the system, all the other ones need to be reprogrammed.

Chapter 6

Implementation in an Embedded System

To evaluate the ideas behind the methods presented in chapter 5 and also to examine the possibilities of the desired characteristics of an implementation of a distributed diagnostic system, defined in section 1.2, a test arrangement consisting of two ECUs was constructed. Some of the key issues that were dealt with in the implementation are presented in the following chapter.

6.1 Hardware Setup

The test arrangement consists of two Scania engine controllers connected via a 250 kbit/s CAN bus. Faults can be hardware simulated and the resulting fault codes can be read with the help of standard Scania tools. Apart from the two ECUs, the setup consists of a power supply and a specially designed control box. The outputs of the ECUs are connected to optical indicators, which is of no importance to the diagnostic system, but used for debugging. The inputs of the ECUs are connected to switches on the control box and these represent components that can be either broken or not. The programming was performed on PCs and transferred to the ECUs via an interface. On this system a framework for distributed diagnosis was successfully implemented and evaluated.

6.2 Software Description

The method implemented and examined in this thesis was based on the second method in chapter 5. The difference between Method 2 and the one implemented is that the implementation has a pre-mapped table of all signal dependencies. Thus no replacement of the dependency of output signals for output signals in the transmitting agent is needed. Since the transmitted set of diagnoses does not include any signals, there is no need for $outputdep$ in the receiving agent and the merge could be done directly.

Various test and component setups were evaluated and in each case the resulting DTCs were correct, i.e. globally consistent. Due to lack of time none of the methods in chapter 5 were fully implemented. The implementation was however enough in order to analyze the desirable characteristics that are presented in the objective of section 1.2.

6.3 Processes in Embedded Systems

First, a short introduction in a typical real-time embedded system is given. An embedded system (in this case the ECU) needs to perform a couple of tasks. The number of tasks that needs to be performed varies in each ECU depending on what it controls. Each task needs to be performed in different frequencies depending on how important they are and how much computational power they require. One can therefore make a couple of processes that are cyclic executed and all tasks within that process are executed. A typical system can consist of for example two cyclic processes, e.g. a 10 ms process (100 Hz) and a 50 ms process (20 Hz). The tasks that not require a specific frequency are located in the background[1] process.

Each process needs to be finished in less time than the process time and a priority rating system is needed for the processes, e.g. the background process has the lowest priority and can be interrupted by the 50 ms process. The 50 ms process can in turn be interrupted by the 10 ms loop. All this is illustrated in Figure 6.1.

There are also some processes that are frequently recurrent but not triggered by internal interrupts. These processes, triggered by external interrupts, are often kept at a minimum in processing time, almost negligible in comparison to let say a 10 ms process. External interrupts can for example be incoming CAN messages (where the interrupt routine only handles storage of the data, not any processing) or CAD[2] dependent interrupts (engine ECU).

A worst case scenario in computational time are often easy to calculate in frequently recurrent tasks, such as fuel injection calculation, but almost impossible in dynamic processes, such as fault isolation. The isolation procedure is thus most suitable placed in the background process. Critical isolation tasks, e.g. electrical faults on breaks etc., can however be placed in a faster loop.

[1]The background process can actually also be a cyclic loop but with variable cycle time.
[2]Crank Angle Degree

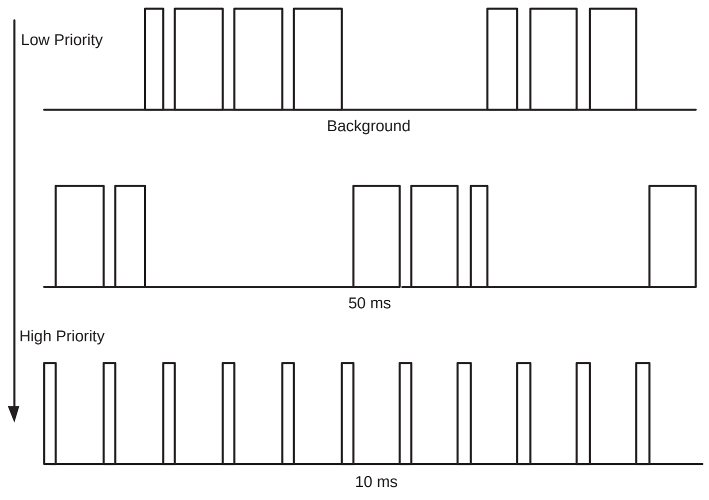

Figure 6.1: The priority order between the 10ms, 50ms and the background process.

6.4 Data Transferring on a CAN Bus

As said in section 3.2.1, each CAN message is sent with an identifier which describes the type of message and where it came from. One message can carry up to eight bytes. When two or more messages are sent on the same time the CAN controller checks the priority of the messages by checking the identifier of the message. A zero in the identifier has higher priority than a one so the controller holds the message if it detects that another message with higher priority is trying to be sent. Each message sent on the CAN bus must therefore be unique.

When a message is received, an interrupt is triggered and the data is copied to a buffer. The buffer is evaluated typically every 10 ms. In the evaluation the data is unpacked and scaled so that the signals sent on the network get the right units.

In order to have a scalable system (see section 3.1.3) all messages that might come into use for the system has to be predefined. So when the interrupt for incoming message is triggered, or the evaluation function is executed, the system has to know what to do with the data belonging to a message with a certain identifier. This causes problems when a fully scalable system is designed. All thinkable identifiers have to be implemented in advance, i.e. the number of possible ECUs with distributed diagnosis has to be decided in advance. Another issue that causes problems because of limitations in CAN is how data larger then eight bytes should be sent. This could however easily be solved by introducing a simple protocol for handling larger diagnostic data.

6.4.1 Protocol Design

The necessary diagnostic information does mostly not fit into one CAN message. One solution could be to divide the large data amount into smaller fragments or packets, i.e. CAN messages, and add a header with all the necessary information. The design possibilities are almost infinite. A suggestion is to divide the diagnoses into sub-diagnoses and transmit one sub-diagnosis per packet.

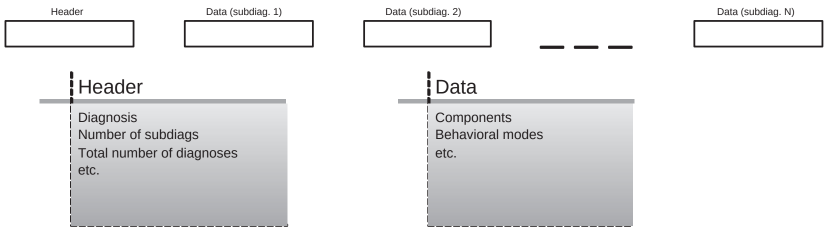

Figure 6.2: A diagnosis statement sent with a header and N data packets.

How often a CAN message can be sent is often limited by the ECU and that is the biggest bottleneck when it comes to transfer a diagnosis statement over the bus. The number of messages should thus be minimized since the correct DTCs could not be calculated until all diagnostic information is received. The preallocated memory (see section 6.5) used for storage of the received diagnosis statement is then mostly occupied of the receiving algorithms.

6.4.2 Transparency

A basic idea throughout the implementation was to make the system transparent from the ECUs point of view, meaning that an ECU treat the other ECUs as a system, not as individual units. An agent does not know from where it receives its messages or to who it transmits its messages, the receiving unit decides what to receive. A different setup, with less transparency, where the communication is addressed could be a different approach. This would require each agent to know which other agents that would be interested in the stated set of diagnoses and select where to transmit its diagnostic information. A transparent system is simplifies the communication and enables a higher degree of scalability and thus it seems as the preferable approach.

6.5 Memory Structure

Since the length of the stated diagnoses is dependent on how many components an ECU is diagnosing and the design of the tests, it is impossible to determine how much data memory is needed to store the diagnoses. It would be preferable to use dynamic memory allocation and store the diagnoses in a so called **heap**. New memory is allocated whenever it is needed and returned when it is not needed anymore. There are some problems however with dynamic memory allocation as, for example, memory fragmentation, memory leakage etc. See [KR89]. On an ECU as simple as the present one in a Scania truck, dynamic memory is very difficult to implement. Thus, instead a static memory allocation is used where a worst-case size of memory needed for generated diagnoses is pre-allocated. Scania's own memory handlers could be used. Pre-allocation is, however, potentially very wasteful because of all the memory that could be left unused by the isolation.

6.5.1 Memory Conflicts

When several processes use the same memory, area conflicts can occur. The conflicts can for example occur when the diagnosis isolation process reads the received diagnosis statements from the other agents and suddenly a new statement is sent over the network. An interrupt is then triggered and starts a function that wants to write in the memory area where the isolation was reading. Some kind of memory handling is needed if not mixed diagnosis data is to be processed by the system causing strange behaviors of the isolation.

A simple solution would be to use a semaphore system where each process that wants to access the memory first checks if it is available and if so, set the semaphore indicating that other processes have to wait. When the accessing process is finished it releases the lock to the memory. It is important that the system does not lose data coming from external interrupts if a memory lock is set. This is solved as a special case in the interrupt routine. For simple and fast reading procedures it could be solved by just holding the interrupt until the reading procedure is done. For larger and slower processes a rolling buffer or several buffers could be used.

6.6 Time Handling

As discussed earlier, the processes in the CPU are run in either different time intervals or in the background. The diagnostic procedure is best suited to be executed in the background process, but could also be calculated in a timed loop. Below, both methods are discussed in detail.

6.6.1 Diagnosis Executed in a Fixed Timed Loop

If the diagnosis executed in a cyclic loop, new diagnoses will be ready at the end of every loop sequence, unless, the fixed time is too short for the isolation to be finished. If, for example, it takes 8 ms for the isolation process to be executed and it is placed in the 10 ms loop, then there will hardly be time for anything else to be executed in the CPU. The isolation process should therefore, in this case, be placed in a slower time interval.

The obvious problem with diagnosis though, is the difficulty of knowing how long it takes to execute. It is a dynamic process and takes different amount of time depending on the design of the tests and their results. Naturally, large diagnoses take longer to process than smaller ones. Thus, if placed in a fixed time sequence a worst-case time length needs to be calculated and based on that a suitable time loop can be chosen. A worst-case, however, can be very long and the difference between the average time and worst-case time is often quite large resulting in a slow diagnostic system.

6.6.2 Diagnosis Executed in the Background Process

As opposed to performing the diagnostics in a fixed time interval it can be handled in the background process. This suit well with the fact that the diagnoses take different amount of time to generate, because in the background the generation of new diagnoses process for as long as necessary and present the result whenever ready. Here, no worst-case scenario is needed and the calculations of new diagnoses start as soon as the old one is finished. The diagnostic system should be faster in comparison to a fixed time loop in most cases but there is no guarantee when the background process is executed.

6.6.3 Synchronization

In either case, fixed timed loop or background process, synchronization of processes is a big issue. The biggest problems occur in the receiving agent. It cannot know if, or when, a diagnosis statement is coming or how many. It could be good to have a standard message telling that everything is OK (also suitable for self-healing etc.). The receiving agent can then know from how many other agents it should expect a diagnosis statement.

Two (perhaps more) choices exist when it comes to synchronization. The receiving agent could either wait until all diagnoses are collected from the other agents, or it could just go on with the isolation and DTC generation without considering the other diagnoses. In the latter alternative, the receiving agent could generate a DTC that not is based on all information, i.e. the agent would know it was doing something that is wrong, and yet do it! One could therefore think that the smarter alternative would be to wait for all other agents. Problems can however occur even in this case.

Assume that an agent is in the process of receiving diagnostic information from another agent and a third agent starts transmitting its diagnoses. By the time the first transmitting agent is finished sending its diagnoses, the receiving agent will be waiting for the third agent to finish the transmitting of its diagnoses. Assume further that the isolation in the first transmitting agent is pretty fast (remember that the biggest bottleneck was the data transmitting on CAN) and starts transmitting again, the same or new, diagnoses. The receiving agent then has to wait for the first one again and so forth. This would completely lock up the isolation process of the receiving agent. It could be solved by keeping track of from which agents it has received diagnoses, but then it would have done the same fault as when it just continued without considering the newest information.

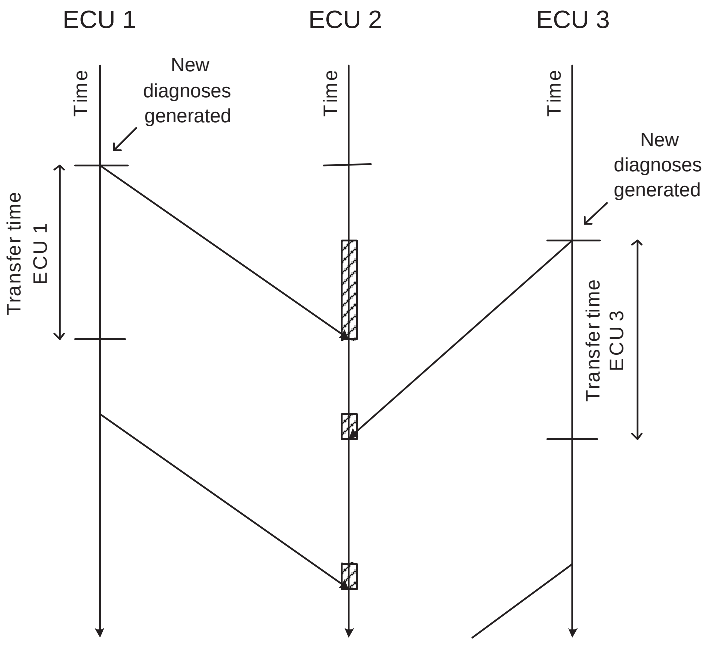

Figure 6.3: Synchronization problem with several ECUs.

6.7 Using Reiter’s Algorithm in Distributed Diagnosis

When using Reiter’s algorithm to generate a diagnosis statement, conflicts or sub-diagnoses are inputs and the minimal diagnoses are the outputs. If the algorithm is used in a reversed manner where the outputted diagnoses are fed as an input to the algorithm, the minimal conflicts or sub-diagnoses are generated. This trick can be useful when dealing with distributed diagnosis.

In Method 2 and 3, see previous chapter, local diagnoses are generated in each ECU, transmitted on CAN and received as diagnoses, requiring a merge between the received diagnoses and the local ones to generate the global diagnoses. Reiter’s algorithm is well suited for performing such a merge. This is done by running the algorithm twice. The first time to generate the minimal sub-diagnoses of the received diagnoses, and the second time to merge the just calculated sub-diagnoses with the local set of diagnoses.

The received diagnoses can be fed into an empty Reiter's algorithm to change their representation from a set diagnoses to a set of sub-diagnoses. Empty means from step one in the algorithm, see page 11. Using the algorithm this way has the reverse effect, which is transforming diagnoses into sub-diagnoses as opposed to the original purpose of the algorithm, see Example 6.1.

Example 6.1
Consider an agent with the following conflicts (sub-diagnoses works in the same way since the corresponding set of sub-diagnoses includes the same components):

$$\pi_1^1 = \{C_1, C_3, C_4\} \qquad \pi_2^1 = \{C_1, C_2, C_3, C_5\} \qquad \pi_3^1 = \{C_1, C_2\}$$

Not all conflicts are minimal, thus redundant information exists. From the conflicts, Reiter's algorithm produces the following set of diagnoses:

$$\mathbb{D} = \{\{C_1\}, \{C_2, C_3\}, \{C_2, C_4\}\}$$

The resulting diagnoses are all minimal. If the diagnosis statement is fed into Reiter's algorithm again it gives back the conflicts:

$$\pi_1^2 = \{C_1, C_2\} \qquad \pi_2^2 = \{C_1, C_3, C_4\}$$

which both are minimal conflicts.

After the received diagnoses have been transformed into sub-diagnoses they are merged with the existing diagnoses of the receiving agent. The merge is performed by running Reiter's algorithm from step 2, with the existing diagnoses as base and the sub-diagnoses as input. A correct merge between the received and local diagnoses was thereby performed without the need of an explicit merge algorithm. The transformations between the transmitting agent's diagnoses and a global diagnosis statement in the received agent are shown in Figure 6.4.

A legitimate question to ask is why the sub-diagnoses generated in the transmitting agent are not sent as sub-diagnoses directly, (similar to Method 1) so that less changes back and forth between representations would be needed. Well, it is a trade-off between processing more data in the receiving agent (an additional execution of Reiter's algorithm) and sending more information on CAN. The diagnoses generated by Reiter's algorithm in the transmitting agent are minimal and consist of no redundant information, while the sub-diagnoses generated from the tests is represented by much more data. The CAN communication is the biggest bottleneck for distributed diagnosis. It needs to be minimized and, therefore, it was chosen to transmit diagnoses.

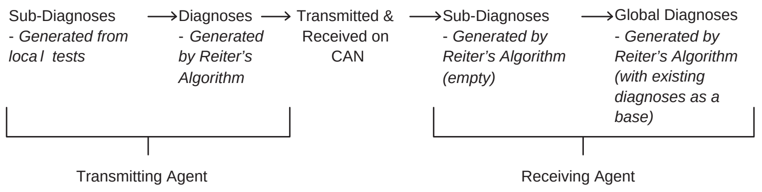
Sub-Diagnoses
- Generated from local tests
Diagnoses
- Generated by Reiter's Algorithm
Transmitted & Received on CAN
Sub-Diagnoses
- Generated by Reiter's Algorithm (empty)
Global Diagnoses
- Generated by Reiter's Algorithm (with existing diagnoses as a base)
Transmitting Agent
Receiving Agent

Figure 6.4: A merge between two local diagnoses using Reiter's algorithm.

6.8 Performance of the Implementation

Each of the desired characteristics, described in section 1.2, of an implementation of a distributed diagnostic system is discussed below. The discussion is based on the implementation that was done in this project.

Time and Memory Consumption

The synchronization problem was solved by simply holding the isolation process until all data was received from the CAN bus. Since the transferring of data is by far the biggest time consumer of a distributed diagnosis system, the isolation process became significantly slower than the original local isolation procedure. The transmitting process was placed in a 50 ms loop, resulting in one diagnostic message every 50 ms from each agent. As discussed in section 6.6.3, this increase in time consumption could be reduced by just considering the last received information, instead of holding the process while waiting for a new message. This would imply that inconsistent diagnoses are calculated under some time.

CPU time is spent on handling the diagnostic information from the other agents. The extra calculation power that is needed, in a distributed system, is dependent on the complexity of the local and the received diagnoses. Reiter's algorithm have to be run two extra times per received set of diagnoses according to section 6.7. This results in a total of three times for a system of two agents, five times for a system of three agents etc., which is a considerable increase.

Since a pre-map of the signal dependencies was utilized in the implementation, the amount of allocated memory in the receiving agent has to be enough to process a worst case set of diagnoses received from the transmitting agent. Considering this implementation, the allocated memory in each ECU is twice as big compared to the original local diagnostic system. If Method 2 or Method 3 were fully implemented the extra amount of memory needed would be significantly reduced.

Bus Analysis

As previously mentioned, a protocol is needed for sending larger amounts of data than eight bytes. In the project implementation the protocol suggested in section 6.4.1 was used. If nothing is diagnosed a message for indicating that no faults exist, or that the system is healed, is sent every 50 ms.

A CAN message every 50 ms on a 250 kbit/s bus was measured to correspond to approximately 1 % bus load per agent. The bus load increases linearly with the frequency of the transmitted messages, e.g. a message sent every 10 ms would approximately correspond to a 5 % bus load.

Scalability

Since this implementation includes a pre-mapping of signal dependencies it is not very scalable. For every new agent that is added, a new map has to be constructed and new memory has to be allocated and so forth. The mapping problem disappears if a full implementation of Method 2 or Method 3 is made. Although, for every kind of implementation, all CAN message identifiers has to be defined in advance. This is discussed more deeply in section 6.4.

Chapter 7

Conclusions

A few methods for distributed diagnosis are proposed. All the methods generate diagnoses in each agent from which globally consistent DTCs can be assigned. A method based on Method 2 in chapter 5 was implemented. The resulting distributed diagnostic system assigns globally consistent DTCs and thereby complies with the goal.

The main conclusions drawn from designing a distributed diagnostic system are the following:

- The information necessary to share is the diagnostic information of the components that are shared over the network, plus cardinality information of the remaining private components, in order to calculate the globally consistent diagnoses.

- It is sufficient to share the behavioral mode of the signal, not the components it originates from, under some assumptions and limitations. Without the assumptions and limitations a component could be represented in several signals and could thus cause inconsistency in cardinality. To prevent this, an initialization process or pre-map of signals would be needed.

- There are some problems, mainly because of signal representation on the CAN bus, one has to deal with if a fully scalable and flexible system is desired. In today's design, all messages need to be predefined in each ECU that take use of a particular message. Further, if the dependency components are to be represented in the signals that is diagnosed, these would have to be predefined if no initialization process is implemented.

- When implementing a distributed diagnostic system, problems with memory handling and process synchronization arises. This is further complicated due to that the isolation process is a non deterministic job and requires non deterministic amount of memory.

Chapter 8

Future Work

The algorithms could be developed further so that the assumptions in section 5.1 are removed. That a component cannot be part of several output signals, or that an output signal cannot depend on input signals, are quite strong restrictions and not desirable in an implementation. New problems arise when trying to solve this, which also needs a solution, e.g. how multiple representations of components should be handled. The future algorithms should have a proof stating that they produce consistent globally correct DTCs without the limitations of this report.

The algorithms could be more optimized to better suite a fast and effective implementation. A better and optimized protocol for transferring diagnosis messages could also be designed.

References

[AP05] B. Wahlberg A. Pernestål, M. Nyberg. A bayesian approach to fault isolation - structure estimation and inference. 2005.

[Bit05a] Jonas Biteus. Distributed diagnosis and simulation based residual generators. Technical report, Dept. of Electrical Engineering, 2005. LiU-TEK-LIC-2005:31, Thesis No. 1176.

[Bit05b] Jonas Biteus. Personal correspondance, Department of Electrical Engineering, Linköpings Universitet, November 2005.

[JBN05] M. Jensen J. Biteus and M. Nyberg. Distributed diagnosis for embedded systems in automotive vehicles. *IFAC World Congress*, 2005.

[JdKR92] Alan K. Mackworth Johan de Kleer and Raymond Reiter. Characterizing diagnoses and systems. 1992.

[Jen03] M. Jensen. Distributed fault diagnosis for networked embedded systems. Master's thesis MMK 2003:63 MDA 231, KTH Machine Design, The Royal Institute of Technology, KTH Machine Design, SE-100 44 Stockholm, Sweden, December 2003.

[JKZ02] X. Koutsoukos J. Kurien and F. Zhao. Diagnosis of large active systems. *Proceedings of the 13th International Workshop on Principles of Diagnosis*, May 2002.

[KR89] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Number ISBN 91-970296-45. Prentice Hall International, Hemel Hempstead, England, 1989.

[MFB99] K. Ratcliff M. Farsi and M. Barbosa. An overview of controller area network. *Computing and Control Engineering Journal*, June 1999.

[NF05] Mattias Nyberg and Erik Frisk. *Model Based Diagnosis of Technical Processes*. 2005.

[NKM02] John P. Hayes Nagarajan Kandasamy and Brian T. Murray. Time-constrained failure diagnosis in distributed embedded systems. 2002.

[NRW03a] A. ten Teije N. Roos and C. Witteveen. Multi-agent diagnosis with semantically distributed knowledge. 2003.

[NRW03b] A. ten Teije N. Roos and C. Witteveen. Multi-agent diagnosis with spatially distributed knowledge. 2003.

[NRW04] A. ten Teije N. Roos and C. Witteveen. Reaching diagnostic agreement in multi-agent diagnosis. 2004.

[PBZ98] P. Pogliano P. Baroni, G. Lamperti and M. Zanella. Diagnosis of large active systems. *Elsevier Science*, July 1998.

[Pro02] Gregory Provan. A model-based diagnosis framework for distributed systems. 2002.

[Sun02] Dan Sune. Isolation of multiple-faults with generalized fault-modes. Master's thesis, Linköpings Universitet, SE-581 83 Linköping, 2002.

[TvS02] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems*. Prentice Hall, international edition, 2002.

Notation

Symbols used in the report.

Operators

$\cup$	Union
$\cap$	Intersection
$\in$	Belongs to
$\subseteq$	Subset
$\subset$	Strict subset
$\wedge$	Logical *and*
$\vee$	Logical *or*
$\neg$	Logical *not*
$\setminus$	Domain difference
$\cup\!\!\!\!\!\times$	Merge

Functions

$con(.)$	Connection
$dep(.)$	Dependency
$dpd(.)$	Dependent Diagnoses

Diagnoses

$\mathcal{D}$	Set of global diagnoses
$\mathbb{D}$	Set of local diagnoses
D	Diagnosis

Abbreviations

CAD	Crank Angle Degree
CAN	Controller Area Network
COO	Coordinator
DIMA	Diagnostic Manager
DTC	Diagnostic Trouble Code
ECU	Electronic Control Unit
MDH	Minimal Diagnosis Hypothesis
OBD	On Board Diagnosis
PE	Processing Elements

Appendix A

Proof of Method 3

[Bit05b]

A union between two local diagnoses, $D^{A_1} \cup D^{A_2}$, should result in the same cardinality as a union between a local diagnosis and a received diagnosis, $D^{A_1} \cup D^{rx}$, when X is used to represent the private components in the received diagnoses, i.e. $|D^{A_1} \cup D^{A_2}| = |D^{A_1} \cup D^{rx}| + X$, where the left-hand side cardinality have full component representation.

Two local diagnoses, in a system consisting of N agents:

The different component types and signals are here assumed to be diagnosed

$$
\begin{aligned}
&D^{A_1} = P_1 \cup G_1 \cup \Gamma_1, \qquad D^{A_1} \in \mathbb{D}^{A_1} \\
&D^{A_2} = P_2 \cup G_2 \cup \Gamma_2, \qquad D^{A_2} \in \mathbb{D}^{A_2} \\
&|D^{A_1} \cup D^{A_2}| = |P_1 \cup P_2 \cup G_1 \cup G_2 \cup dpd(\Gamma_1 \cup \Gamma_2)| = |G_1 \cup G_2| + \\
&\qquad\qquad\qquad\qquad\qquad\qquad |P_1 \cup P_2 \cup dpd(\Gamma_1 \cup \Gamma_2)|
\end{aligned}
$$

because

$$|G_1 \cup G_2| \cap |P_1 \cup P_2 \cup dpd(\Gamma_1 \cup \Gamma_2)| = \emptyset$$

Split the set Γ_1 into two sets:

$$
\Gamma_1 = \bar{\Gamma}_1 \cup \widetilde{\Gamma}_1, \text{where } \begin{cases} \widetilde{\Gamma}_1 = \{\gamma \in \Gamma_1 \mid con(\gamma) \in \Sigma_2\} \\ \bar{\Gamma}_1 = \{\gamma \in \Gamma_1 \mid con(\gamma) \in \Sigma_{3\ldots N}\} \end{cases}
$$

$$
\Gamma_1 = \bar{\Gamma}_1 \cup P_2^{OUT}, \text{where } P_2^{OUT} \in \mathop{\bigcup\!\!\!\!\!\times}\, dep(\sigma), \sigma \in con(\widetilde{\Gamma}_1)
$$

$$P_2^{OUT} \subseteq P_2$$

Split the set Γ_2 into two sets:

$$\Gamma_2 = \bar{\Gamma}_2 \cup P_1^{OUT}, \text{ where } \begin{cases} \bar{\Gamma}_2 = \{\gamma \in \Gamma_2 \mid con(\gamma) \in \Sigma_{3\ldots N}\} \\ P_1^{OUT} \in \biguplus dep(\sigma), \text{for all } \sigma \in con(\Gamma_2) \cap \Sigma_1 \end{cases}$$

$$P_1^{OUT} \subseteq P_1$$

One diagnosis $D \in D^{A_1} \cup D^{A_2}$:

$$|D| = |G_1 \cup G_2| + |P_1 \cup P_1^{OUT}| + |P_2 \cup P_2^{OUT}| + |\bar{\Gamma}_1 \cup \bar{\Gamma}_2|$$
Because $|dpd(\bar{\Gamma}_1 \cup \bar{\Gamma}_2)| = |\bar{\Gamma}_1 \cup \bar{\Gamma}_2|$

X represents private components as in Method 3:

$$D^{rx} = G_2 \cup \Gamma_2 \cup \Sigma_2 = G_2 \cup \bar{\Gamma}_2 \cup \widetilde{\Gamma}_2 \cup \Sigma_2$$
$$|D^{A_1} \cup D^{rx}| = |P_1 \cup G_1 \cup G_2 \cup \Gamma_1 \cup \bar{\Gamma}_2 \cup \widetilde{\Gamma}_2 \cup \Sigma_2| =$$
$$= |G_1 \cup G_2| + |P_1 \cup P_1^{OUT}| + |\bar{\Gamma}_1 \cup \bar{\Gamma}_2| + |\widetilde{\Gamma}_1 \cup \Sigma_2|$$

Let $X = |P_2 \cup P_2^{OUT}| - |\widetilde{\Gamma}_1 \cup \Sigma_2|$

$$X = |P_2| + |P_2^{OUT}| - |P_2 \cap P_2^{OUT}| - |\widetilde{\Gamma}_1| - |\Sigma_2| + |\widetilde{\Gamma}_1 \cap \Sigma_2|$$
$$|P_2^{OUT}| - |\widetilde{\Gamma}_1| = 0 \quad \text{and} \quad |P_2 \cap P_2^{OUT}| - |\widetilde{\Gamma}_1 \cap \Sigma_2| = 0 \qquad \Rightarrow$$
$$X = |P_2| - |\Sigma_2|$$

This implies

$$|D^{A_1} \cup D^{A_2}| = |D^{A_1} \cup D^{rx}| + X$$

because

$$\begin{aligned} |D^{A_1} \cup D^{rx}| + X = {} & |G_1 \cup G_2| + |P_1 \cup P_1^{OUT}| + |\bar{\Gamma}_1 \cup \bar{\Gamma}_2| \\ & + |P_2| + |P_2^{OUT}| - |P_2 \cap P_2^{OUT}| - |\widetilde{\Gamma}_1| - |\Sigma_2| + |\widetilde{\Gamma}_1 \cap \Sigma_2| = \\ & |G_1 \cup G_2| + |P_1 \cup P_1^{OUT}| + |P_2 \cup P_2^{OUT}| + |\bar{\Gamma}_1 \cup \bar{\Gamma}_2| \end{aligned}$$

Q.E.D.