A Decentralized Service Based Architecture for Design and Modeling of Fault Tolerant Control Systems

Mattias Nyberg, Carl Svärd
Department of Electrical Engineering, University of Linköping, SE-58183 Linkoping, Sweden
mattias.nyberg@scania.com, carl.svard@liu.se

ABSTRACT

The paper presents a hierarchical architecture for fault tolerant control of mechatronic systems. In the architecture, both the diagnosis and the reconfiguration are completely decentralized according to the structure of the control system. In each module of the control system, diagnosis and reconfiguration are included and use only locally available signals. Therefore, no extra dependencies are added to the control system software, something which is important to facilitate efficient engineering of large scale system. All this is achieved by using a purely service oriented view of the system including both hardware and software. The service view with no cyclic dependencies is further used to obtain Bayesian networks for modeling the system.

1 INTRODUCTION

The main contribution of the present work is an architecture and design method for fault-handling in mechatronic control systems. We have primarily been inspired by control systems in automotive vehicles. In these systems, fault tolerance mostly means that each fault should lead to a minimal degradation of the performance, and typically, fault tolerance is achieved by control-system reconfiguration. A second contribution is that we show how a fault tolerant control system, designed in accordance with the proposed architecture, can be modeled using Bayesian networks.

One standard solution to Fault Tolerant Control (FTC), as often suggested in the literature, e.g. see (Zhang and Jiang, 2008; Blanke *et al.*, 2006; Bonivento *et al.*, 2003), is to have one centralized diagnoser diagnosing the whole system, and then a centralized reconfiguration based on the diagnosis result. However, for large-scale mechatronic control systems, possibly with functionality distributed over several Electronic Control Units (ECU's), another solution might be needed. We list three reasons supporting this claim. The first is that to obtain good FTC-performance, e.g. a fast and guaranteed response to detected faults, the FTC-algorithms need to be executed locally in each ECU, since network communication would slow down and make the response unreliable. A second reason is the complexity associated with the number of fault combinations and number reconfiguration possibilities that would need to be considered in a centralized solution. A third reason is that in order to facilitate efficient distributed engineering of large scale systems, it is important to minimize the dependencies between subsystems; a centralized FTC-solution would easily lead to that all subsystems connected to the reconfiguration become, via the diagnoser and the reconfiguration, dependent on all subsystems providing inputs to the diagnoser. For these reasons, instead of the centralized solution, the FTC problem is in our proposed architecture solved using a decentralized approach. In this architecture, the FTC mechanisms are distributed according to the architecture of the control system, which is assumably already structured to cope with system size and complexity. Important also is that the proposed FTC architecture adds no extra dependencies in the software (SW) compared to the control system architecture.

In the control system, we view both software and hardware subsystems as service providers and the relation between subsystems is based solely on how failures of these services propagates in the system. In this framework, we have abstracted away all objects and information not related to failures, such as signals. Thereby, a minimalistic pure failure-oriented view of the system is obtained. This view is used as the foundation both for the FTC architecture and for modeling the system. Note that since inability to perform a service is the same as a failure (Laprie, 1991), we can speak about a service-oriented view as well as a failure-oriented view.

The service (failure) oriented view of the system facilitates a design of the fault handling mechanism with noncyclic dependencies within the system. This is important for modeling since it makes it possible to use Bayesian networks; a clear benefit since Bayesian networks have shown to be powerful for modeling and making inference in systems containing uncertainties. One example is automotive engine control systems, whose On-Board Diagnosis (OBD) systems are required by regulations to handle subtle faults like biases, which usually have uncertain effects on the system. By having a model in the form of a Bayesian network, inference tasks like diagnosis becomes easy. Other examples of useful inference tasks are analysis of FTC-performance and analysis of consequences of different faults, for example FMEA or FTA, see e.g.(Storey, 1996).

Architecture of fault tolerant control systems have

been considered also in earlier literature, e.g. see (Blanke *et al.*, 2006; Grunske and Kaiser, 2005; Staroswiecki, 2008; Gehin and Staroswiecki, 2008). The key differences compared to these earlier works are that: 1) the diagnosis task in the present paper is completely decentralized among the components of the system, 2) we use the service view, where software components are viewed as service providers with service dependencies, as the basis for the FTC-architecture, 3) we discuss explicitly how the different SW-components of the systems communicate with each other about failure, 4) we use Bayesian networks as the tool for modeling, and we include in the model also the diagnosers.

2 BASIC CONCEPT

We consider a system of ECU's connected in a communication network. The control system software in each ECU is assumed to be designed, in a first iteration, without considering or including diagnosis or FTC functionality. We partition the control system software into separate parts, called *components* or *modules*. Each module communicates with other modules, within the same ECU or, by using the communication network, also with modules in other ECU's.

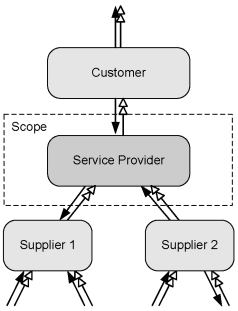

Figure 1: A Service Provider, its suppliers and customer. Service (failure) dependencies are represented by (white) double headed arrows and signal flows by single headed arrows.

2.1 The Service View

In the control system, the purpose of each module is to provide one or several services. Therefore we view each module as a *service provider*. The service is provided to a *customer*, which itself is a module and service provider. There may also be several customers, and the top-level customer of a system is often a human user. To provide its service, a service provider may use a number of *suppliers*, which in turn are service providers, see Figure 1. Note that the service of a module is not, as in (Staroswiecki, 2008; Gehin and Staroswiecki, 2008), defined as the computation of the output signal of the module, or the output signal itself. Instead the service is the *purpose* or the *objective* of the module.

In our service view of the system, the service dependencies, represented in Figure 1 by double headed arrows, do in general not have the same direction as the signal flows within the system. For example consider a service provider which is master controller of a cascade control system. One of its suppliers is the slave controller which receives the reference signal from the master. Thus the signal flow in this example is directed from the service provider to the supplier, but the service dependency always has the direction from supplier to service provider. To find out the direction of service dependencies, one can think of how faults propagate in the system. In the example, a fault in the slave controller propagates and affects the service provided by the master controller. The idea to allow fault propagation not to follow signal flow directions has been used also in (Deb *et al.*, 1995; Misra *et al.*, 1994) but stands in contrast to models used in (Mohamed and Zulkernine, 2008; Grunske and Kaiser, 2005), in which fault propagation is constrained to signal flow directions.

How to determine the granulation of the partition of the control software into modules, is a matter of consideration. Using small modules, it is easier to add diagnosis and FTC-functionality. A small module is also easier to model. However, in a small module, with few signals available, good diagnosis and FTC may be difficult to include because of lack of redundancy. Furthermore, the complexity of the whole system and its model may be increased if the modules are small but many, because the number of service providers becomes large and also the number of dependencies between them.

For sake of clarity, we will from now on assume that each service provider only provides one single service. The extension to several services is trivial.

One main advantage with the service view is that the service dependencies form a graph with no directed cycles. This means that we later on in Section 6 can use these dependencies as the basis for a Bayesian network modeling the system. An argumentation to why directed cycles are avoided is as follows. For a service to be delivered, a set of tasks need to be performed. We assume that each service provider performs at least some of these tasks, and not just delegate all tasks further to a supplier. This means that the service delivered by a supplier is always a proper subset, in terms of tasks, compared to the service delivered by the service provider itself. A directed cycle in the service dependency graph would therefore lead to the contradiction that the set of tasks performed by some service provider would be a proper subset of itself.

2.2 Service Status

The service of a service provider may be available or not available. If available, the service has a service quality which is mostly nominal. It may also happen that it becomes disturbed in some way. We collect these *service statuses* into three classes which we denote *nominal* (*NOM*), *disturbed* (*DIST*), and *unavailable* (*UNA*). The service status of a service provider m will be denoted $\mathcal{S}_m$. Even though the quality of a service may change, we do assume that the objective of a service does not change. However, this assumption can be relaxed by considering different operating modes as in (Gehin and Staroswiecki, 2008).

The status *NOM* is always applicable, but the statuses *UNA* and *DIST* may or may not be needed depending on the specific case. If needed for a specific service provider, the service status *DIST* can also be extended to several levels of disturbed, i.e. $DIST_1$, $DIST_2$, $DIST_3$, etc.

As stated in Section 1, it is important to minimize the dependencies between modules. Therefore, in accordance with ideas of modularization, software hierarchies, abstraction, encapsulation etc., e.g. see (Storey, 1996), each service provider is, for its design and computations, only allowed to use information within its *scope*, see Figure 1. The scope contains knowledge about its suppliers and often also some part of the hardware or electronics.

2.3 Service Status Estimation

The service provider has, in addition to providing the service itself, also the task to monitor and estimate the status of its own service and to communicate this estimated status to its customers. The reason is that if the customer becomes aware that a service delivered has a degraded status, the customer can adapt to the situation for example by reconfiguring itself. Also, when the customer is to estimate the status of its own service, it is crucial to take into account a possibly degraded status of one of its supplier's services. For a customer to be able to utilize a communicated service status from the service provider, all statuses must be clearly defined in advance. This definition must be available to, and agreed with, the engineers designing the SW-modules acting as customers of the service provider.

When a service provider estimates the status of its service, a variety of information sources throughout the system could be useful, but according to the principles of scope discussed above, only information within its scope is allowed to be used. It is evident that a service status estimate obtained by using information within the scope only may differ and be less accurate compared to what would be the result using all information in the system. This is the prize we pay to keep the number of dependencies low.

The estimate produced by a service provider m of its service status $\mathcal{S}_m$, using information within its scope, is denoted $\widehat{\mathcal{S}}_{m|m}$. Clearly, an estimated service status is in general not equal to the true service status, i.e. $\widehat{\mathcal{S}}_{m|m} \neq \mathcal{S}_m$. Except for the fact that we use only the limited information within the scope, another reason is that this is a general estimation problem under the influence of noise and model uncertainties.

Since the estimate $\widehat{\mathcal{S}}_{m|m}$ might not be possible to obtain with high accuracy, it is sometimes enough to use a more restricted domain for $\widehat{\mathcal{S}}_{m|m}$ compared to $\mathcal{S}_m$. For instance, $\mathcal{S}_m \in \{NOM, DIST, UNA\}$ while $\widehat{\mathcal{S}}_{m|m} \in \{NOM, UNA\}$. We use the convention that the estimated service status communicated from a service provider to a customer should always be guaranteed from the perspective of the service provider itself. That is, the estimated service status is set to *NOM* as long as there are no signs of anything else, but as soon as there are signs of that the service status is not *NOM*, the estimated service status should be changed to *DIST* or *UNA*. The customer should then always assume that the true service status is not better than the communicated one, but it may be worse, e.g. if the estimated service status is *DIST*, the customer should assume that the service status is not *NOM*.

We will apply the view of service providers also to hardware (physical) components. Everything said above is valid except that hardware components do not have the ability to estimate and communicate the status of its services. This is principally the same as if they are always communicating the status *NOM*.

3 ILLUSTRATIVE EXAMPLE

To illustrate the principles and concepts in previous and subsequent sections we introduce a simple, yet relevant and realistic, mechatronic system inspired by automotive applications.

3.1 System Description

We consider a simplified air-control system whose main task is to control the air-flow past a throttle in order to maintain a given stoichiometric ratio under the influence of an external fuel reference command. A schematic view of the system is shown in Figure 2, in which signal flows are represented by single headed arrows and service dependencies by (white) double headed arrows. Note that signal flows form directed cycles but service dependencies do not. The physical part of the system, lower part of Figure 2, consists of a tank, containing high-pressure air, connected through a pipe to ambient air. A throttle valve and a fuel injector is mounted in the pipe. The throttle valve is maneuvered via a proportional valve and there are sensors measuring the position of the throttle valve and the pressure in the air-tank. The sensors, the proportional valve, and the injector are connected to an ECU, upper part of Figure 2, in which a set of controllers and estimators are implemented as software modules.

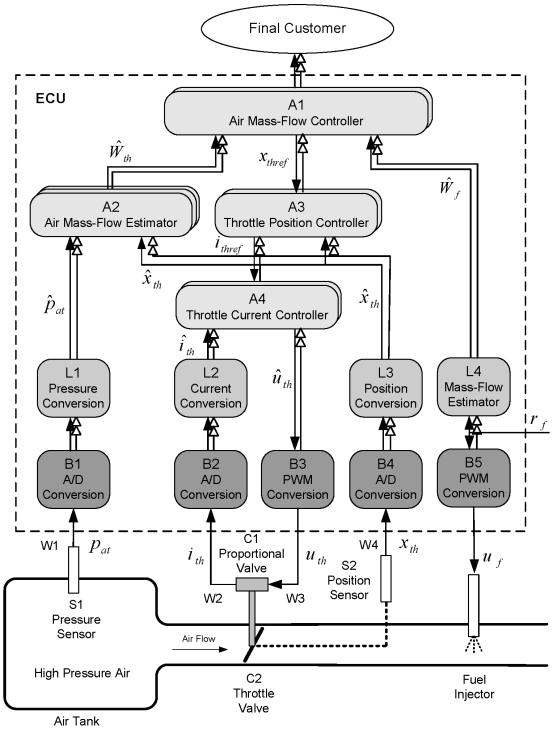

Figure 2: The air-control system. Double headed arrows indicate service dependencies, single headed arrows signal flows.

Note that the main reason to consider this relatively small-sized system is merely to clarify the proposed approach and exemplify introduced concepts, not to exemplify how the approach handles large-scale systems. This is left for future work.

3.2 Service Providers

Following the ideas introduced in Section 2, we consider all components in the system, hardware components as well as software modules, as service providers. A list of all service providers, their provided service, suppliers and customers, is shown in Table 1.

For example, consider module $A3$, whose service is to control the position of the throttle valve. Module $A3$ is master controller in a cascade control subsystem consisting of modules $A3$ and $A4$. The service provided by the slave controller $A4$ is to control the current through the proportional valve. If $A4$ fails to provide its service,

	Service	Suppliers(s)	Customer
A1	Control air mass-flow past throttle in order to maintain stoichiometric ratio	A2, A3, L4	Final
A2	Deliver air mass-flow past throttle	L1, L3	A1
A3	Control throttle valve position	A4, L3, C2	A1
A4	Control current through proportional valve	L2, B3, C1, W3	A3
L1	Deliver pressure measured in air tank	B1, S1, W1	A2
L2	Deliver current through proportional valve	B2, W2	A4
L3	Deliver position of throttle valve	B4, S2, W4	A2
L4	Deliver fuel mass-flow in injector	B5	A1
B1	Deliver digital number corresponding to voltage at input port	-	L1
B2	Deliver digital number corresponding to voltage at input port	-	L2
B3	Deliver power at output port corresponding to duty cycle	-	A4
B4	Deliver digital number corresponding to voltage input port	-	L3
B5	Deliver power corresponding to duty cycle	-	L4
W1	Transport measure voltage from sensor to input port, supply sensor with voltage and ground	-	L1, S1
W2	Transport current from proportional valve to grounded input port	-	L2
W3	Transport power from output port to proportional valve	-	A4
W4	Transport voltage from sensor to input port, supply sensor with voltage and ground	-	L3, S2
S1	Deliver voltage corresponding to pressure	W1	L1
S2	Deliver voltage corresponding to position	W4	L3
C1	Deliver force corresponding to current	-	A4
C2	Deliver position corresponding to force	-	A3

Table 1: Service Providers in the air-control System

then also $A3$ will fail to provide its service. This means that $A4$ is a supplier of $A3$ although the signal flow, i.e. the current reference signal, goes from $A3$ to $A4$, c.f. the single and double headed arrows in Figure 2. To utilize closed-loop control, $A3$ uses feedback of the throttle valve position. The position is delivered by $L3$, which is also a supplier of $A3$. To control the position of the valve, $A3$ is also dependent of that the valve itself works properly. Hence the physical hardware component $C2$, corresponding to the throttle valve, is also a supplier of $A3$.

The scope associated with $A3$ consists of knowledge about modules $A4$, $L3$, and the physical component $C2$, i.e. the throttle valve, since $A3$ needs to know what kind of device it should control.

To illustrate how a failure and degraded service statuses propagates through the system, suppose $A3$'s supplier $L3$ fails to provide its service, i.e. deliver the throttle valve position, and sets $\widehat{\mathcal{S}}_{L3|L3} = \mathit{UNA}$. Since $A3$ is dependent on the throttle valve position for closed-loop control, it reconfigures itself and instead applies an open-loop control strategy in which the throttle valve position is not needed. $A3$ can then still provide its service, but since the performance of the open-loop controller is slightly worse than the closed-loop controller, it sets $\widehat{\mathcal{S}}_{A3|A3} = \mathit{DIST}$. Since the meaning of service statuses is agreed between suppliers and service providers, module $A1$ knows that the throttle position control performance is degraded when $\widehat{\mathcal{S}}_{A3|A3} = \mathit{DIST}$.

4 GENERAL PRINCIPLES OF A SERVICE PROVIDER

As stated above, a service provider has the task to provide its services, and also the task to estimate the status of these services. To achieve fault tolerance, the service provider should also try to keep up the service quality, even though faults make the service status of suppliers *DIST* or *UNA*. The standard solution for this is reconfiguration, as exemplified above. In this paper we represent reconfiguration by using a concept of *variants*. That is, a service provider m may exist in several variants denoted m:1, m:2 etc. and each variant typically uses different, possibly overlapping, subsets of suppliers. Thus their sensitivity to supplier services becoming *DIST* or *UNA* differ, and by selecting the most appropriate variant to be run, fault tolerance is obtained. When using variants, the service status of the service provider equals that of the selected variant $m{:}i$, i.e. $\mathcal{S}_m = \mathcal{S}_{m:i}$. If there is no reconfiguration, we will view this as a case of only one possible variant.

In a service provider with the capability of reconfiguration, there must exist a *selector*, which is a mechanism for choosing the variant to be executed in a given situation. We will not give any general principles for this, even though we later on, in Section 5, discuss an example design. In a real-time system, the change of variant may cause transients to occur. This needs to be handled but is considered out of scope of the present paper.

Example 1 (Variants in Service Provider $A2$) Recall the air-control system described in Section 3 and consider service provider $A2$, whose service is to provide the air mass-flow past the throttle. For this it uses the suppliers $L1$ and $L3$. To increase fault-tolerance, there are three variants of the service provider, denoted $A2$:1, $A2$:2, and $A2$:3. The variants are illustrated in Figure 2 as multiple shaded layers behind the block representing $A2$.

Variant $A2$:1 is the main variant and uses signals from both suppliers $L1$ and $L3$ to estimate the air mass-flow. Variant $A2$:2 uses only the signal from supplier $L1$ for the estimation, and in a similar way variant $A2$:3 uses only the signal from supplier $L3$. With this design, $A2$ can deliver its service, possibly with reduced quality, even if one of the suppliers $L1$ and $L3$ fails to deliver its service.

4.1 Service Status Estimation

To estimate the service status of the service provider, and also to be able to select the most appropriate variant, the service status estimates of the variants are needed, i.e. $\widehat{\mathcal{S}}_{m:i|m}$. For the estimation of $\widehat{\mathcal{S}}_{m:i|m}$, the variant $m{:}i$ uses the service statuses communicated from its suppliers. Further, it typically uses the signals to and from its suppliers and possibly also other signals within the service provider. The important features of these signals are extracted using *diagnostic tests*, and thus, the signals themselves are not directly used for the status estimation. Diagnostic tests are obtained for example by using standard techniques from the field of FDI (Fault Detection and Isolation) (Blanke *et al.*, 2006),e.g. analytical redundancy and residuals.

The service status estimation can be seen as a mapping from each combination of estimated supplier service statuses and test results to the service status of the variant. In the simplest cases this mapping is hard coded in the software using constructs like if- or switch/case-statements. In more involved cases, the mapping can be

represented and processed using a lookup-table. Also possible is to use a model based approach in which the service status is inferred using a model together with observed test results and estimated service statuses communicated from the suppliers. Whatever the chosen solution is, the mapping must be possible to represent efficiently in a model, to facilitate efficient inference and analysis of the system (see also Section 6).

Some general principles for estimating the service status will now be given. We start with the simple case where there are no diagnostic tests involved.

No Diagnostic Tests Involved
For the service status estimation, each variant has a number of possible service status values. It may correspond to the full domain of the possible values of the estimated service status of the service provider, but it can also be a subset of the full domain. For each service status value to be valid, the estimated service statuses of the suppliers used must fulfill a condition.

For an example, consider a service provider variant $A{:}1$ with suppliers B and C. To represent the service status estimation in $A{:}1$ we can use a table as the one exemplified below.

	$\widehat{\mathcal{S}}_{A:1\mid A}$		
	NOM	*DIST*	*UNA*
$\widehat{\mathcal{S}}_{B\mid B}$	*NOM*	*NOM* $\vee$ *DIST*	-
$\widehat{\mathcal{S}}_{C\mid C}$	*NOM*	-	-

(1)

The interpretation of the table is that the service status of $A{:}1$ is estimated to be *NOM* if both supplier B and supplier C communicate that their service status is *NOM*. Else, if Supplier B communicates service status *NOM* or *DIST*, then the service status is estimated to be *DIST*. Else, the service status is estimated to be *UNA*.

Using Diagnostic Tests
To improve the service status estimation we will now discuss how information from also diagnostic tests can be used. Without loss of generality, we assume that diagnostic tests utilized for service status estimation are contained within the service provider. Thus the signals used by a test must be known within the scope. Otherwise we get a dependency, via the diagnostic test, to the outside of the scope of the service provider.

For an example of how a diagnostic test may be utilized, consider a variant that implements a PI controller. Then we can use diagnostic tests checking the control error and integrator size. A further example, if a variant implements an observer, we can use diagnostic tests based on residuals checking the validity of the input signal against a model of the system. The use of diagnostic tests means that even though all suppliers estimate their services to be *NOM*, additional information from diagnostic tests may imply that the service status of the variant becomes *DIST*.

Each diagnostic test uses a subset of the suppliers. This means that if any supplier in the subset communicates the service status *UNA*, the diagnostic test can not be run.

Exactly how the results of the diagnostic tests are used by the service provider to assist the service status estimation will not be further discussed here. Partly due to space limitation but mainly because we believe that there is not one single best way since different circumstances demand different solutions. However, an example will be presented later in Section 5.

Example 2 (Service Status Estimation in $L1$)
Recall again the air-control system presented in Section 3. The service of service provider $L1$ is to deliver the measured pressure in the air tank. The customer of $L1$ is $A2$ and its suppliers are $B1$, $S1$, and $W1$. Due to the nature of this service and the limited amount of signals available for use in $L1$, its only possible communicated service statuses are *NOM* and *UNA*, i.e. $\widehat{\mathcal{S}}_{L1|L1} \in \{\textit{NOM}, \textit{UNA}\}$

To aid the estimation of its service status, $L1$ contains one diagnostic test T_{L1} which simply checks if the pressure is in-range, with respect to physical limitations of the tank, as well as the specification of the sensor $S1$. That is, T_{L1} equals 0 (no alarm) if $J^1_{L1} \geq \hat{p}_{at} \geq J^2_{L1}$ and 1 (alarm) otherwise. The service status estimation logic for $L1$ is thus very simple; if $T_{L1} = 0$ and $\widehat{\mathcal{S}}_{B1|B1} = \textit{NOM}$, then $\widehat{\mathcal{S}}_{L1|L1} = \textit{NOM}$. Otherwise, $\widehat{\mathcal{S}}_{L1|L1} = \textit{UNA}$.

4.2 Summary

We summarize the elements of the service provider together with their requirements:

- A service provider may exist in several variants, and there must be a selector, i.e. a mechanism for selecting the variant.
- Each variant uses a subset of suppliers and a subset of diagnostic tests. The subsets may be overlapping.
- Each diagnostic test in the service provider uses a subset of suppliers, and the subsets may be overlapping.
- Each variant has a mechanism for service status estimation.
- The service status estimation uses as input: estimated service statuses communicated from the suppliers, and diagnostic test results.
- The service status estimation must be possible to represent efficiently, e.g. in a lookup table.

5 EXAMPLE DESIGN OF SERVICE PROVIDERS

We consider the principles in Section 2 and 4 to be sound and general in the sense that they can be applied to every service provider implemented in software. Below follows a design discussion on a more detailed level. However, we consider this to be an example design rather than a general design since, depending on circumstances such as practical considerations, other solutions might be preferred. For instance, in many small service providers, much more simple solutions are natural, e.g. see Example 2.

The strategy used consists of a two-part solution; variant service status estimation and variant selection. In the first part, the service status of the different variants are estimated. This is done by executing the diagnostic tests and then, based on the test results, using a *diagnoser* to estimate the service statuses of the suppliers. This estimate of the service status of supplier n is denoted by $\widehat{\mathcal{S}}_{n|\text{tests}}$. Each estimate $\widehat{\mathcal{S}}_{n|\text{tests}}$ is then combined with the one communicated from the supplier, i.e. $\widehat{\mathcal{S}}_{n|n}$ to obtain $\widehat{\mathcal{S}}_{n|m} = \min(\widehat{\mathcal{S}}_{n|\text{tests}}, \widehat{\mathcal{S}}_{n|n})$ where we have used the min-operator to represent a selection of the worst status. Next, for each variant $m{:}i$, use the estimated statuses $\widehat{\mathcal{S}}_{n|m}$ of the suppliers together with the principle illustrated in the table (1) to obtain estimated service status of each variant $\widehat{\mathcal{S}}_{m:i|m}$.

In the second part, the variant with best estimated service status $\widehat{\mathcal{S}}_{m:i|m}$ is chosen as the one to be executed.

If there is no unique variant with best estimated service status, a fixed linear preference order of all variants, determines which one to select.

The estimated service status of the whole service provider becomes equal to the status of the selected variant, i.e. let $\widehat{\mathcal{S}}_{m|m} = \widehat{\mathcal{S}}_{m:i|m}$. In case all variants have their estimated status $\widehat{\mathcal{S}}_{m:i|m}$ equal to *UNA*, no variant can be executed and the estimated service status of the service provider becomes *UNA*, i.e. $\widehat{\mathcal{S}}_{m|m} =$ *UNA*.

These principles are now illustrated in an example.

Example 3 (Service Status Estimation in $A2$)
We return to the air-control system and the air mass-flow estimation module $A2$. In the following, we illustrate how the principle outlined in Section 5 is used to estimate the service statuses of the different variants $A2{:}1$, $A2{:}2$ and $A2{:}3$ described in Example 1.

In order to estimate the service statuses of suppliers $L1$ and $L3$, a diagnostic test T_{A2} is utilized. This test is based on a residual r_{A2} formed as the difference between a pressure estimate, obtained from a physical model of the tank, and the pressure $\hat{p}_{at}$, delivered by $L1$, see Figure 2. The diagnostic test is obtained by comparing the residual r_{A2} with a threshold J_{A2}, and hence T_{A2} equals 0 if $|r_{A2}| \leq J_{A2}$ and 1 otherwise.

Since T_{A2} uses signals from both $L1$ and $L3$, the failure signature matrix (FSM) for the test is

	$\mathcal{S}_{L1}$ = *DIST*	$\mathcal{S}_{L3}$ = *DIST*
T_{A2}	X	X

where $\mathcal{S}_{L1}$ and $\mathcal{S}_{L3}$ denotes the true service status of $L1$ and $L1$ respectively. Thus, if T_{A2} alarms we may conclude that either $L1$ or $L3$, or both, has status *DIST*. We assume that the diagnoser available for use is conservative, hence if T_{A2} alarms it concludes that both $L1$ and $L3$ have status *DIST*.

Given the outcome of T_{A2}, we attain the service status estimates $\widehat{\mathcal{S}}_{L1|T_{A2}}$ and $\widehat{\mathcal{S}}_{L3|T_{A2}}$. The final estimates are calculated as $\widehat{\mathcal{S}}_{L1|A2} = \min\left(\widehat{\mathcal{S}}_{L1|T_{A2}}, \widehat{\mathcal{S}}_{L1|L1}\right)$ and $\widehat{\mathcal{S}}_{L1|A2} = \min\left(\widehat{\mathcal{S}}_{L3|T_{A2}}, \widehat{\mathcal{S}}_{L3|L3}\right)$, where $\widehat{\mathcal{S}}_{L1|L1}$ and $\widehat{\mathcal{S}}_{L3|L3}$ are the communicated service statuses from the suppliers. According to Section 4.1, the test T_{A2} is not run if $\widehat{\mathcal{S}}_{L1|L1} =$ *UNA* or $\widehat{\mathcal{S}}_{L3|L3} =$ *UNA*. In those cases, the estimated service statuses are based solely on the communicated statuses, i.e. $\widehat{\mathcal{S}}_{L1|A2} = \widehat{\mathcal{S}}_{L1|L1}$ and $\widehat{\mathcal{S}}_{L3|A2} = \widehat{\mathcal{S}}_{L3|L3}$.

Given the estimated supplier service statuses, we can estimate the status of each of the variants $A2{:}1$, $A2{:}2$, and $A2{:}3$. The conditions, in terms of estimated supplier service statuses, for a variant's service status are in general a result of an engineering process. For example, the air mass-flow estimation algorithm used in $A2{:}1$ is designed in a way so that an adequate estimate can be calculated even if $\widehat{\mathcal{S}}_{L1|A2} =$ *DIST* or $\widehat{\mathcal{S}}_{L3|A2} =$ *DIST* so that in this case $\widehat{\mathcal{S}}_{A2:1|A2} =$ *DIST*. The supplier status conditions for all variants are given in the following table with an assumed preference order where a status to the left is preferred over a status to the right.

	$\widehat{\mathcal{S}}_{A2:1\|A2}$			$\widehat{\mathcal{S}}_{A2:2\|A2}$		$\widehat{\mathcal{S}}_{A2:3\|A2}$	
	NOM	*DIST*	*UNA*	*DIST*	*UNA*	*DIST*	*UNA*
$\widehat{\mathcal{S}}_{L1\|A2}$	*NOM*	*NOM* $\vee$ *DIST*	-	*NOM*	-	-	-
$\widehat{\mathcal{S}}_{L3\|A2}$	*NOM*	*NOM* $\vee$ *DIST*	-	-	-	*NOM*	-

The table should be interpreted in this way: if, for example, $\widehat{\mathcal{S}}_{L1|A2} =$ *DIST* and $\widehat{\mathcal{S}}_{L3|A2} =$ *NOM* then $\widehat{\mathcal{S}}_{A2:1|A2} =$ *DIST*.

Suppose now there is a fault affecting the electrical wire $W1$ connected to pressure sensor $S1$, e.g. short-cut to ground. This leads to that a voltage of 0 V is delivered to the ECU input port. In spite of the fault affecting $W1$, $B1$ can nevertheless deliver its service, which is to give the voltage at the input port, and therefore $\widehat{\mathcal{S}}_{B1|B1} =$ *NOM*.

In $L1$, the monitoring test T_{L1}, see Example 2, alarms since 0 V does not correspond to a pressure in-range and therefore $\widehat{\mathcal{S}}_{L1|L1} =$ *UNA*. If we assume that there are no other faults in the system, we have that $\widehat{\mathcal{S}}_{L3|L3} =$ *NOM*. The test T_{A2} is not executed since $\widehat{\mathcal{S}}_{L1|L1} =$ *UNA* and thus $\widehat{\mathcal{S}}_{L1|A2} = \widehat{\mathcal{S}}_{L1|L1} =$ *UNA* and $\widehat{\mathcal{S}}_{L3|A2} = \widehat{\mathcal{S}}_{L3|L3} =$ *NOM*.

In accordance with the table above we then have that $\widehat{\mathcal{S}}_{A2:1|A2} =$ *UNA*, $\widehat{\mathcal{S}}_{A2:2|A2} =$ *UNA*, and $\widehat{\mathcal{S}}_{A2:3|A2} =$ *DIST*. The variant to be executed in $A2$ becomes $A2{:}3$ and $\widehat{\mathcal{S}}_{A2|A2} = \widehat{\mathcal{S}}_{A2:3|A2} =$ *DIST*.

6 DIAGNOSTIC MODELING

A diagnostic model is a model that includes all relevant faults and the symptoms they cause in the system. We need such a model because of at least two reasons. The first is the use for analysis, and the second is for model based diagnosis and troubleshooting at the workshop. The questions asked during analysis are to obtain measures of how fault tolerant the system is, or how easy it is to localize faults. For example we can investigate probability of false alarm, probability of missed detection, probabilities of failures of different degrees of severity. The questions asked during diagnosis and troubleshooting are for example: given a set of diagnostic test results, what is the most probable faulty component, or what is the most probable cause for a specific service to become unavailable? As said in the introduction, we use a model in the form of a Bayesian network since it allows for probability based inference.

A system with service providers, software modules, and hardware components, arranged such that no directed cycles appear, can be interpreted as a causal network. Then by adding probabilities, we obtain a Bayesian network. In addition to service providers, we add nodes also for diagnostic tests, estimated service statuses, and selectors. We will now describe the details of these principles, and we will use a model of service provider $A2$, $A3$, $L1$, $B1$, $S1$, and $W1$ as an illustrating example, see Table 1 and Figure 3.

6.1 Structure and Variables of the Bayesian Network

We consider four kinds of nodes: true service statuses of both the service provider $\mathcal{S}_m$ and its variants $\mathcal{S}_{m:i}$, estimated service statuses of both the service provider $\widehat{\mathcal{S}}_{m|m}$ and its variants $\widehat{\mathcal{S}}_{m:i|m}$, diagnostic test results T_j, and selectors V_m. Note that service providers are both hardware components and software modules. The methodology for determining nodes, structure and variables is now described step by step:

1. The structure is first constructed by considering the service providers and their service dependencies. At this stage hardware components have to be included in the model. We create one service status

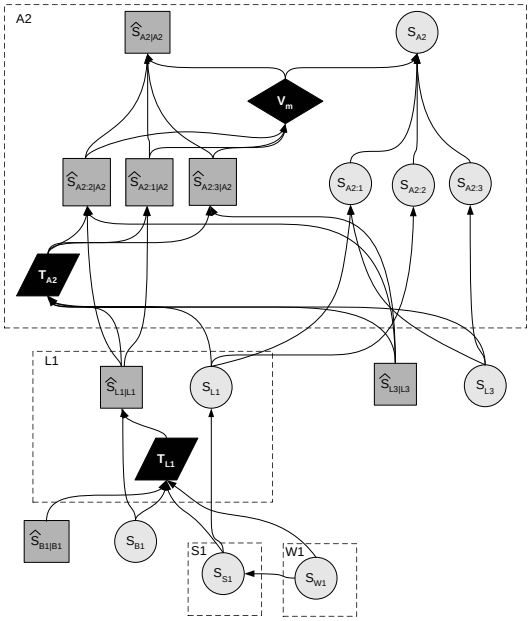

Figure 3: Model of Service Providers $A2$, $L1$, $S1$, and $W1$.

node $\mathcal{S}_m$ for each service provider and additionally, for service providers having several variants, one service status node $\mathcal{S}_{m:i}$ for each variant. The causal links are the service dependencies. This is illustrated by the light-gray circular nodes in Figure 3. As seen, there are also links from each variant service-status $\mathcal{S}_{m:i}$ to the true service status $\mathcal{S}_m$ of the service provider. In this case, with variants, the node $\mathcal{S}_m$ only represents a copying of the true service status from the selected variant.

2. In parallel to the structure built so far, we build for software modules a structure of nodes representing the estimated service statuses of service providers and variants. This is illustrated in Figure 3 by the dark-gray square nodes.
3. We add nodes for test results T_j. These are illustrated in Figure 3 by black rhombus. The test result nodes have incoming links from the true service statuses of suppliers since these are the nodes determining if the test will alarm or not. Also incoming links from estimated service statuses communicated from the suppliers to tests are needed since, as described in Section 4.1, the execution of a test is disabled if suppliers communicate *UNA*. The tests in a service provider are used to estimate the service status of the different variants, and therefore, the outgoing links from the tests go to the estimated service statuses of the variants.
4. We add nodes for selectors V_m. The selector node represents the selection of variant, which we here view, in accordance with Section 5, as a mapping from estimated service statuses of the variants to variants. As seen in Figure 3, the selector node has outgoing links to control which variant that is to be used both for the estimated service status $\widehat{\mathcal{S}}_{m|m}$ and the true service status $\mathcal{S}_m$.
5. The variable domains need to be determined. The domains of the estimated service status nodes $\widehat{\mathcal{S}}_{m|m}$ and $\widehat{\mathcal{S}}_{m:i|m}$ are given from the software as described in Section 4.1. The test results T_j have binary domains $\{0, 1\}$ and the domain of each selector V_m is the set of possible variants of the service provider. The domains of true service statuses $\mathcal{S}_m$ and $\mathcal{S}_{m:i}$ are obtained by analyzing which fault modes each hardware component can be in and how these fault modes propagate up-wards in the system. Thus, for hardware components, the domain typically correspond to the different fault modes. For software modules, the domains include the status *NOM* and *UNA*, but the status *DIST* might in some cases need to be extended to obtain a higher resolution. That is, one value of the variable domain of $\widehat{\mathcal{S}}_m$ may correspond to several possible values in the domain of $\mathcal{S}_m$. The reason is that we may know, and want to model, that a certain test responds differently depending on in which way, or how much, a service of a supplier is disturbed.

6.2 Conditional Probability Distributions

The CPD's of the estimated service status nodes $\widehat{\mathcal{S}}_{m|m}$ and $\widehat{\mathcal{S}}_{m:i|m}$ and the the selectors V_m are deterministic and given by the corresponding algorithms in the software. The representation of each CPD can be a lookup table, or the algorithm itself if it is efficient enough.

The CPD's of the tests T_j and the service statuses $\mathcal{S}_m$ and its variants $\mathcal{S}_{m:i}$ are stochastic, and they are typically determined by expert knowledge and/or by analyzing the system. Here it is useful to use the principle of noisy-or to reduce the number of parameters and simplify the determination of the CPD, e.g. see (Jensen and Graven-Nielsen, 2007).

Example 4 (Inference)
Using the principles introduced above, a Bayesian network of the whole air-control system was built using (GeNIe, 2010). To illustrate the use of the model, we have here included four experiments. The results of these experiments are shown in Table 2. The first two experiments illustrate an analysis of how the system responds to faults, and the last two experiments illustrate troubleshooting scenarios.

In the first experiment, the single fault open circuit (OC) of the wire $W1$ has been assumed by setting the evidence $\mathcal{S}_{W1} = OC$ and all other true service statuses of physical components to the evidence *NOM*. Then the effect on all software modules is computed using the Bayesian network and the result, concerning all affected parts of the network, is shown in the left part of Tables 2a and 2b. The probability for *NOM* is left out from the table but is 1 minus probabilities for *DIST* and *UNA*. We can for example see that the true service status $\mathcal{S}_{L1}$ will be *UNA*, the true service status $\mathcal{S}_{A2}$ will be *DIST*, and $\mathcal{S}_{A1}$ will be *DIST* with probability 0.9. It is also evident that the software modules $L1$ and $A2$ will provide estimates $\widehat{\mathcal{S}}_{L1|L1}$ and $\widehat{\mathcal{S}}_{A2|A2}$ in total agreement with the true statuses. However, the software module $A1$ may be a bit conservative since $P(\widehat{\mathcal{S}}_{A1|A1} = \textit{DIST}) = 0.95$ compared to $P(\mathcal{S}_{A1} = \textit{DIST}) = 0.9$.

The second experiment is similar but the fault assumed is a single fault bias (B) in sensor $S1$. In this case, due to the nature of bias faults, we can see in the right part of Table 2a that there is no guaranteed response of any diagnostic test. The diagnostic tests are used to compute estimated statuses and as shown in the table, also these become uncertain.

	Open Circuit $W1$	**Bias Fault** $S1$
Test	*ALARM*	*ALARM*
T_{L1}	1.0*	0.05
T_{A2}	1.0*	0.90*
T_{A1}	0.05	0.10

(a) Diagnostic test probabilities

	Open Circuit $W1$		**Bias Fault** $S1$	
Status	*DIST*	*UNA*	*DIST*	*UNA*
$\widehat{\mathcal{S}}_{L1\|L1}$	-	1.0	-	0.05
$\widehat{\mathcal{S}}_{A2\|A2}$	1.0	0.0	0.90	0.0
$\widehat{\mathcal{S}}_{A1\|A1}$	0.95	0.05	0.78	0.13
$\mathcal{S}_{L1}$	0.0	1.0	0.9	0.05
$\mathcal{S}_{A2}$	1.0	0.0	0.82	0.09
$\mathcal{S}_{A1}$	0.90	0.05	0.74	0.13

(b) Service status probabilities

	Open Circuit $W1$	**Bias Fault** $S1$
Comp.	*faulty*	*faulty*
$W1$	0.98	0.0
$W2$	0.0	0.0
$W3$	0.0	0.0
$W4$	0.0	0.0
$S1$	0.02	0.8
$S2$	0.0	0.2
$C1$	0.0	0.0
$C2$	0.0	0.0

(c) Fault probabilities

Table 2: Probabilities computed using the Bayesian network.

As a conclusion so far, we can also use the numbers in the table to obtain a measure of fault tolerance against the two faults considered. As measure we use the probability that the system can continue its operation, which corresponds to $P(\mathcal{S}_{A1} \neq \mathit{UNA} | \mathcal{S}_{W1} = \mathit{OC})$ and $P(\mathcal{S}_{A1} \neq \mathit{UNA} | \mathcal{S}_{S1} = B)$ respectively. From the table we can conclude that these numbers are 0.95 and 0.87 respectively.

In the third experiment, we consider a troubleshooting scenario and assume that the single fault open circuit of the wire $W1$ is present and that this has resulted in that the tests T_{L1} and T_{A2} have responded. This is in agreement with the conclusions of the first experiment and it is highlighted in Table 2a by a star next to these two test results. For troubleshooting, the only thing given is the test results from all tests and these are therefore given as evidence to the Bayesian network. We then use the Bayesian network to compute the probabilities of different faults. This is shown in the left part of Table 2c and it is evident that the Bayesian inference succeeds in perfectly localizing the fault.

In the fourth experiment, we assume that the single fault bias in the sensor $S1$ is present and that this has resulted in that only the test T_{A2} has responded. As seen in the right part of Table 2c, the inferred probability of a fault in $S1$ is 0.8 but also a fault in $S2$ gets some probability mass. That is, the fault localization is not perfect, but a mechanic trusting these diagnosis results would certainly first try to repair sensor $S1$ which in fact would give a successful repair.

7 CONCLUSIONS

The paper has presented a hierarchical architecture for fault tolerant control of large-scale mechatronic systems. In the architecture, both the diagnosis and the reconfiguration are completely decentralized according to the structure of the control system. In each module of the control system, diagnosis and reconfiguration are included and use only locally available signals. Therefore, no extra dependencies are added to the SW. This is of key importance since a requirement of efficient engineering of large scale system, is to reduce and control the amount of dependencies in the system. All this has been possible to achieve by using a purely service oriented view of the system including both hardware and software. The service view with no cyclic dependencies is further used as the basis for obtaining Bayesian networks for modeling the system.

ACKNOWLEDGMENTS

This work was supported by Scania CV AB, Södertälje, Sweden. The authors also want to thank the MSc student Erik Landström for his work with modeling examples.

REFERENCES

(Blanke *et al.*, 2006) M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki. *Diagnosis and Fault-Tolerant Control*. Springer Berlin Heidelberg, 2 edition, 2006.

(Bonivento *et al.*, 2003) Claudio Bonivento, Andrea Paoli, and Lorenzo Marconi. Fault-tolerant control of the ship propulsion system benchmark. *Control Engineering Practice*, 11(5):483 – 492, 2003. Automatic Control in Aerospace.

(Deb *et al.*, 1995) S. Deb, K.R. Pattipati, V. Raghavan, M. Shakeri, and R. Shrestha. Multi-signal flow graphs: a novel approach for system testability analysis and fault diagnosis. *Aerospace and Electronic Systems Magazine, IEEE*, 10(5):14–25, 1995.

(Gehin and Staroswiecki, 2008) A.L. Gehin and M. Staroswiecki. Reconfiguration analysis using generic component models. *IEEE Trans. on Systems, Man, and Cybernetics. Part A: Systems and Humans*, 38(3):575–583, 2008.

(GeNIe, 2010) GeNIe. Decision Systems Laboratory, University of Pittsburgh, URL: http://genie.sis.pitt.edu/, 2010.

(Grunske and Kaiser, 2005) L. Grunske and B. Kaiser. Automatic generation of analyzable failure propagation models from component-level failure annotations. International Conference on Quality Software, 2005.

(Jensen and Graven-Nielsen, 2007) F. Jensen and T. Graven-Nielsen. *Bayesian Networks and Decision Graphs*. Springer, 2 edition, 2007.

(Laprie, 1991) J.C Laprie, editor. *Dependability: Basic Concepts and Terminology*. Springer, 1991.

(Misra *et al.*, 1994) A. Misra, J. Sztipanovits, and J. Carnes. Robust diagnostics: Structural redundancy approach. SPIE's Symposium on Intelligent Systems, 1994.

(Mohamed and Zulkernine, 2008) A. Mohamed and M. Zulkernine. On failure propagation in component-based software systems. International Conference on Quality Software, pages 402–411, 2008.

(Staroswiecki, 2008) M. Staroswiecki. On fault handling in control systems. *International Journal of Control, Automation, and Systems*, 6(3):296–305, 2008.

(Storey, 1996) Neil Storey. *Safety-Critical Computer Systems*. Addison Wesley, 1996.

(Zhang and Jiang, 2008) Y. M. Zhang and J. Jiang. Bibliographical review on reconfigurable fault-tolerant control systems. *IFAC Annual Reviews in Control*, 32(2):229–252, 2008.