A Service Based Approach to Decentralized Diagnosis and Fault Tolerant Control

Mattias Nyberg and Carl Svärd

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. 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.**

I. Introduction

This paper is motivated from work with fault tolerance of mechatronic systems in automotive vehicles. In these applications, fault tolerance is important in the sense that each fault should lead to a minimal degradation of the performance, and typically, the means for fault tolerance is control-system reconfiguration.

One standard solution to Fault Tolerant Control (FTC), as often suggested in the literature, e.g. see [1], [2], [3], [4], 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 systems with complex functionality distributed over several Electronic Control Units (ECU:s), such as automotive control systems, another solution might be needed. For these systems, we believe that instead of the centralized approach, the diagnosis and FTC problems need to be solved using a decentralized and modular architecture dividing the large system into smaller subsystems. Another architectural issue, relevant in both the centralized and decentralized approach, is how the diagnoser should interact with the control system. However, as noted in [1] and there posed as an open question, the literature is sparse on such investigations.

The present paper introduces a decentralized architecture for diagnosis and FTC. The role of diagnosis in the context of FTC is discussed, specifically how the diagnoser should interact with the reconfiguration. With the suggested approach, and due to the decentralization, it turns out that the diagnosis task and the interaction sometimes becomes so simple so it is not even necessary or relevant to speak about diagnosis.

We suggest that the proposed architecture is suitable for large-scale mechatronic systems. This since both diagnosis and FTC are completely decentralized and follows 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.

We view both software and hardware components as service providers. We organize these components into a hierarchy where the relations between components is based solely on how service failures propagate in the system. In this framework, we have abstracted away all objects and information not related to failures, such as signals. In this way, a minimalistic purely 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 [5], 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 with noncyclic dependencies within the FTC 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 very powerful for modeling and making inference in systems containing uncertainties. One example is automotive mechatronic systems, whose diagnosis system is 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 become easy. Other examples of useful inference tasks are analysis of FTC-performance and analysis of consequences of different faults.

Architecture of fault tolerant control systems have been considered also in earlier literature, e.g. see [2], [6], [7], [8]. 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 network as the tool for modeling, and we include in the model also the diagnosers.

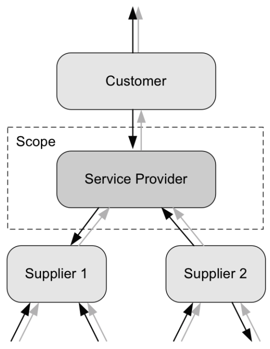

Fig. 1. A Service Provider, its suppliers and customer. Service dependencies are represented by gray arrows and signal flows by black arrows.

II. Concept Basics

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

This work was supported by Scania CV AB

Mattias Nyberg and Carl Svärd are with Department of Electrical Engineering, University of Linköping, SE-58183 Linkoping, Sweden mattias.nyberg@scania.com

considering or including diagnosis or FTC functionality. We partition the control system software into separate parts, called *components* or *modules*. Each such module communicates with other modules, within the same ECU or, by using the communication network, with modules in other ECU:s.

A. 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 [7], [8], 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.

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 system and the 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.

Note that the service dependencies, represented in Figure 1 by gray arrows, do in general not have the same direction as the signal flows within the system. It can be realized that the graph constructed from these service dependencies will contain no cycles, something which is of key importance when, later on in Sec. VI, the system is modeled using Bayesian networks.

To exemplify the directional difference between service dependency and signal flow, 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 approach here stands in contrast to earlier literature, e.g. see [9], [6], in which fault propagation is constrained to signal flow directions.

B. Service Status

The service provided by 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 $\mathcal{S}_m$. Except that the status of a service may change, we assume that the service, provided by a service provider, is persistent, i.e. the service is always provided and does not change in character. However, this assumption can be relaxed by considering different operating modes as in [8].

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 be extended to several levels of disturbed, i.e. *DIST_1*, *DIST_2*, *DIST_3* etc.

To facilitate efficient distributed engineering of large scale systems, it is important to minimize the dependencies between modules. Therefore, in accordance with ideas of modularization, abstraction, encapsulation etc., each service provider, is for its design and computations, only allowed to use information from its *scope*, e.g. see Figure 1. The scope contains knowledge about its suppliers and often also some part of the hardware or electronics.

C. 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 of 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 the 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 in advance be clearly defined. This definition must be available to, and agreed with, the engineers designing the SW-modules acting as customers of the service provider.

Note that in the proposed architecture, we assume that there is no module dedicated to diagnosis only. Instead all diagnosis for FTC is decentralized and performed within the control system modules.

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[1].

Since the estimate $\widehat{\mathcal{S}}_{m|m}$ might not be possible to obtain with high precision, it is sometimes enough to use a lower resolution of $\widehat{\mathcal{S}}_{m|m}$ compared to $\mathcal{S}_m$. This means that the domains of $\widehat{\mathcal{S}}_{m|m}$ and $\mathcal{S}_m$ may differ. For instance, $\mathcal{S}_m \in \{\mathit{NOM}, \mathit{DIST}, \mathit{UNA}\}$ while $\widehat{\mathcal{S}}_{m|m} \in \{\mathit{NOM}, \mathit{UNA}\}$.

We use the convention that the estimated service status communicated from a service provider to a customer should

[1]The second reason is in fact a special case of the first since if the scope was infinite we could use the true physical values and become immune to e.g. measurement noise.

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*.

III. 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. We consider a simplified air-management system whose main task is to control the air-flow through 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 black arrows and service dependencies by gray arrows. Note that signal flows form directed cycles but service dependencies do not.

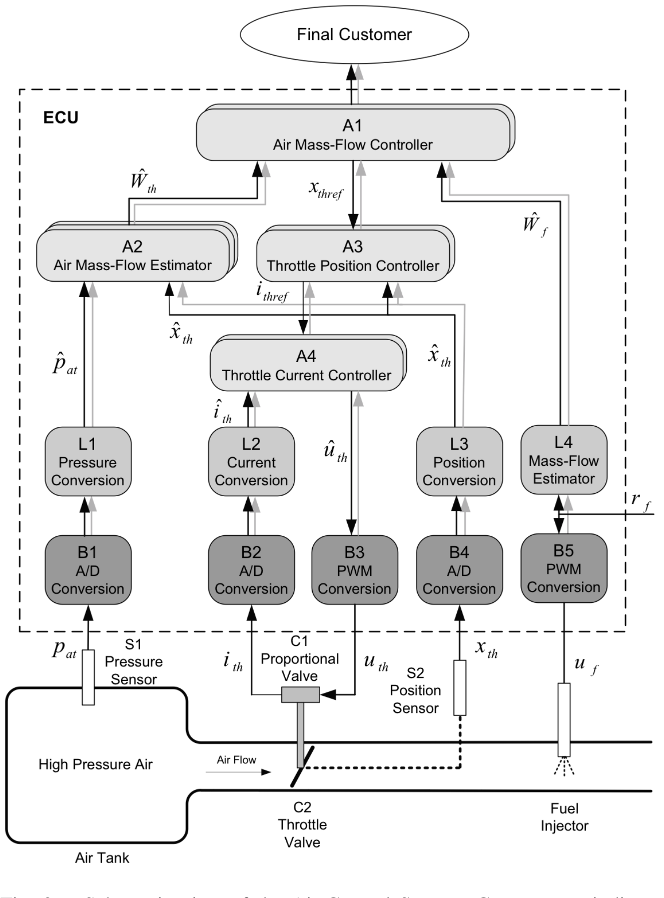

Fig. 2. Schematic view of the Air-Control System. Gray arrows indicate service dependencies and black arrows signal flows.

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 through 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.

Following the ideas introduced in Section II, we consider all components in the system, hardware components as well as software modules, as service providers.

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, 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 black and gray 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 $A3$. The scope associated with $A3$ consists of modules $A4$ and $L3$, but also the physical component $C2$, i.e. the throttle valve, since $A3$ need to know what kind of device it should control.

IV. 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 suppliers *DIST* or *UNA*. The standard solution for this is reconfiguration. In this paper we represent reconfiguration in accordance with [8] 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 currently active, 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.

Example 1 (Variants in Service Provider $A2$): Recall the air-control system described in Section III 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, $A2$ will be able to deliver its service, possibly with reduced quality, even if one of the suppliers $L1$ and $L3$ fails to deliver its service.

A. *Estimation of Service Status*

To estimate a service status of the service provider, the service status estimate of the selected variant is needed, i.e. $\widehat{\mathcal{S}}_{m:i|m}$. For the estimation of $\widehat{\mathcal{S}}_{m:i|m}$, the selected variant 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) [2], [10], e.g. analytical redundancy and residuals.

The service status estimation can be seen as a mapping from each combination of 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 VI). This can for example be achieved if the mapping is represented as a lookup table. Another requirement is that real-time systems require predictable and limited computation times.

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.

1) No Diagnostic Tests Involved: A variant has a *default service status* meaning the best possible service status of the variant. The most common case is, of course, that the default service status is *NOM*. However, under the assumption that supplier service statuses are *DIST* or *UNA*, it may not be possible to design a variant delivering the service with status *NOM*. It might then be useful to have a variant designed to deliver a service with status *DIST* but with less requirements on the supplier service statuses. This is a typical situation when reconfiguration is considered.

Based on the idea of default service status and also that, naturally, a variant can not work if any of its suppliers is not available, we obtain the following general principles:

a) If all suppliers communicate service status *NOM*, then the estimated service status of the variant becomes equal to the default service status.
b) If any supplier communicates service status *UNA*, then the estimated service status of the variant becomes *UNA*.

If any supplier communicates the service status *DIST*, then the estimated service status of the variant may be *NOM*, *DIST*, or *UNA*. The choice is based on a systematic analysis or engineering decisions. In other words, except for items (a) and (b) above, no other general principles for service status estimation can be stated.

To represent the service status estimation we can use a table as the one exemplified below.

	Estimated Service Status		
Comm. Serv. Stat. from:	*NOM*	*DIST*	*UNA*
Supplier 1	*NOM*	*NOM* ∨ *DIST*	-
Supplier 2	*NOM*	-	-

(1)

The interpretation of this table is that the service status is estimated to be *NOM* if both Supplier 1 and Supplier 2 communicate that their service status is *NOM*. Else, if Supplier 1 communicates service status *NOM* or *DIST*, then the service status is estimated to be *DIST*. Else, the service status is estimated to be *UNA*.

2) Using Diagnostic Tests: If a service provider knew the true status of the services from all suppliers, the service status estimation would be easy. However, since the suppliers only communicate the service status estimated by themselves, there is a good potential to utilize also diagnostic tests when estimating the service status of the variant.

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 estimates 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 V.

Example 2 (Service Status Estimation in $L1$): Recall again the air-control system presented in Section III. The service of service provider $L1$ is to deliver the pressure in the air tank. The customer of $L1$ is $A2$ and its supplier is $B1$. 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. $\widehat{\mathcal{S}}_{L1|L1} \in \{NOM, UNA\}$

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. 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 $\widehat{\mathcal{S}}_{B1|B1} = NOM$, then $\widehat{\mathcal{S}}_{L1|L1} = NOM$. Otherwise, $\widehat{\mathcal{S}}_{L1|L1} = UNA$.

Note that a diagnoser is hardly needed in this case. This clearly illustrates how the role of diagnosis may become marginalized as a result of our approach, as discussed in Section I.

B. *Selection of 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. The selector can be viewed as a mapping from the diagnostic test results and estimated service statuses communicated from the suppliers to variants. We will not give any general principles for this, but instead discuss an example design given in Section V. 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.

C. 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: 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.

From above it is clear that there are many design choices to be made. For example, how many variants should there be of a service provider, how to determine the logic of the service status estimation? This should be made such that the desired level of FTC performance is achieved. How to asses the FTC performance is discussed in Sec. VI.

V. Example Design of Service Providers

We consider the principles in Section II and IV to be sound and general in the sense that they can be applied to every service provider implemented in software. When discussing the design of service providers on a more detailed level we choose to present this as an example rather than a general design. The reason is that in a specific case, the most beneficial solution is not necessarily the one presented here. For instance, in many small service providers, much more simple solutions may be preferred, e.g. see Example 2.

We will present an algorithm for the top-level execution of a service provider, but first we need to fix the strategy of selecting the variant to be executed. The strategy used here is 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, use 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 the minimization selects the worst status. Next, the status estimation of the different variants is performed according to the principles discussed in Section IV-A.1.

In the second part, the variant with best service status 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.

Now we present the algorithm. To simplify the presentation we have assumed only one level of *DIST*.

1) For each diagnostic test, run the test if the communicated service status from all its suppliers are not *UNA*.
2) With the diagnostic test results as input, run the diagnoser to obtain $\widehat{\mathcal{S}}_{n|\text{tests}}$ for each supplier.
3) For each supplier, let $\widehat{\mathcal{S}}_{n|m}$=min($\widehat{\mathcal{S}}_{n|\text{tests}}$, $\widehat{\mathcal{S}}_{n|n}$).
4) 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 the variant $\widehat{\mathcal{S}}_{m:i|m}$.
5) If any variant has status *NOM*, run the most preferred one having status *NOM*.
   Else, if any variant has status *DIST*, run the most preferred one having status *DIST*.
6) If any variant $m{:}i$ has been run, set the estimated service status of the service provider equal to the status of that variant, i.e. let $\widehat{\mathcal{S}}_{m|m}$=$\widehat{\mathcal{S}}_{m:i|m}$.
   Else, set the estimated service status of the service provider equal to *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 V 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 generator in which the pressure in the air tank p_{at} is calculated by using a model and the position of the throttle valve $\hat{x}_{th}$, as delivered by $L3$. A residual is then formed as the difference between the calculated pressure and the pressure $\hat{p}_{at}$, delivered by $L1$. The diagnostic test is obtained by comparing the residual r_{A2} with a threshold J_{A2}. 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*. To be safe, the latter is assumed.

Given the outcome of T_{A2}, we attain the service status estimates $\widehat{\mathcal{S}}_{L1|T_{A2}}$ and $\widehat{\mathcal{S}}_{L3|T_{A2}}$. Following the method described in Section V, 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 V, 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 status of a variant to be valid is 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.

	$\widehat{\mathcal{S}}_{A2:1\|A2}$			$\widehat{\mathcal{S}}_{A2:2\|A2}$		$\widehat{\mathcal{S}}_{A2:3\|A2}$	
	NOM	*DIST*	*UNA*	*DIST*	*UNA*	*DIST*	*UNA*
$\widehat{\mathcal{S}}_{L1\|A2}$	*NOM*	*NOM* ∨ *DIST*	-	*NOM*	-	-	-
$\widehat{\mathcal{S}}_{L3\|A2}$	*NOM*	*NOM* ∨ *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 an electrical fault affecting one of the wires connected to pressure sensor $S1$, e.g. shortcut to ground. This implies a voltage of 0 V to the ECU input port. In spite of the fault affecting the wire, $B1$ 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} 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*. Following the principle in Section V, the monitoring 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*, cf. Example 3.

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*. Using the reconfiguration principle described in Section V, we conclude that the variant to be executed in $A2$ is $A2$:3 and that $\widehat{\mathcal{S}}_{A2|A2}$ = $\widehat{\mathcal{S}}_{A2:3|A2}$ = *DIST*.

VI. Diagnostic Modeling with Bayesian Network

A *diagnostic model* includes all relevant faults and their symptoms 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?

Based on the principles presented in Section II and IV, we will now shortly describe how a diagnostic model of the system can be built. The idea is to, as far as possible, use standard principles for modeling with Bayesian networks, e.g. see [11].

A system with service providers, software modules and hardware components, arranged such that no directed cycles appear can be interpreted as a casual network. Then by adding probabilities to the causal network, we obtain a Bayesian network which allows for probability based inference, something that is necessary to answer the questions posed above. In addition to service providers, we add nodes also for diagnostic tests, estimated service statuses, and selectors. If a service provider has variants, each variant is represented by its own node. The selector node represents the selection of variant, which according to Section IV-B, is a mapping from communicated service statuses of the suppliers and test results to variants. The service provider becomes a node representing a copying of the service status from the selected variants. Thus, the service provider node has incoming links from each variant and from the selector.

The CPD:s (Conditional Probability Distributions) of the estimated service statuses and selectors are deterministic and given by the actual implementation. CPD:s for diagnostic tests and service providers are given by expert knowledge.

By utilizing these principles, we have modeled the whole system illustrated in Figure 2. A part of the model, representing service provider $A2$, is shown in Figure 3. In this model, the mapping representing the selection of variant, has been refined to follow the algorithm given in Sec. V.

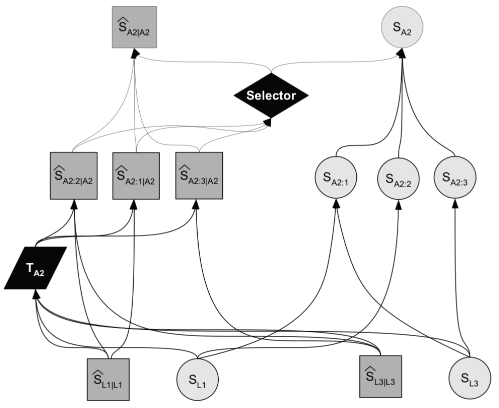

Fig. 3. Model of Service Provider A2

VII. 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. 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.

References

[1] Y. M. Zhang and J. Jiang, "Bibliographical review on reconfigurable fault-tolerant control systems," *IFAC Annual Reviews in Control*, vol. 32, no. 2, pp. 229–252, 2008.

[2] M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki, *Diagnosis and Fault-Tolerant Control*, 2nd ed. Springer Berlin Heidelberg, 2006.

[3] R. Isermann, *Fault-Diagnosis Systems*. Springer Berlin, 2006.

[4] C. Bonivento, A. Paoli, and L. Marconi, "Fault-tolerant control of the ship propulsion system benchmark," *Control Engineering Practice*, vol. 11, no. 5, pp. 483 – 492, 2003, automatic Control in Aerospace.

[5] J. Laprie, Ed., *Dependability: Basic Concepts and Terminology*. Springer, 1991.

[6] L. Grunske and B. Kaiser, "Automatic generation of analyzable failure propagation models from component-level failure annotations," ser. International Conference on Quality Software, 2005.

[7] M. Staroswiecki, "On fault handling in control systems," *International Journal of Control, Automation, and Systems*, vol. 6, no. 3, pp. 296–305, 2008.

[8] A. Gehin and M. Staroswiecki, "Reconfiguration analysis using generic component models," *IEEE Trans. on Systems, Man, and Cybernetics. Part A: Systems and Humans*, vol. 38, no. 3, pp. 575–583, 2008.

[9] A. Mohamed and M. Zulkernine, "On failure propagation in component-based software systems," ser. International Conference on Quality Software, 2008, pp. 402–411.

[10] J. Gertler, *Fault Detection and Diagnosis in Engineering Systems*. Marcel Dekker, Inc., Basel, 1998.

[11] F. Jensen and T. Graven-Nielsen, *Bayesian Networks and Decision Graphs*, 2nd ed. Springer, 2007.