Institutionen för systemteknik
Department of Electrical Engineering
Examensarbete

Evaluation of a diagnostic tool for use during system development and operations

Examensarbete utfört i Fordonssystem
av

Daniel Andersson
Patrik Sköld

LITH-ISY-EX--07/3916--SE
Linköping 2007

Evaluation of a diagnostic tool for use during system development and operations

Master Thesis

Department of Electrical Engineering
Linköping University

Daniel Andersson

Patrik Sköld

LITH-ISY-EX--07/3916--SE

Supervisor: **Torbjörn Fransson**
Andreas Bergström
Mattias Krysander

Examiner: **Erik Frisk**

Linköping, 16 April 2007

Presentationsdatum
21-03-2007

Publiceringsdatum (elektronisk version)

Institution och avdelning
Institutionen för systemteknik

Department of Electrical Engineering

Linköpings universitet

Språk

___ Svenska
_X_ Annat (ange nedan)

Engelska
Antal sidor
91

Typ av publikation

____Licentiatavhandling
_X_Examensarbete
____C-uppsats
____D-uppsats
____Rapport
____Annat (ange nedan)

ISBN

ISRN **LiTH-ISY-EX-3916**

Serietitel

Serienummer/ISSN

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

Titel Evaluation of a diagnostic tool for use during system development and operations
Title

Författare Daniel Andersson & Patrik Sköld
Author

Sammanfattning
Abstract

Rodon is a diagnostic tool developed by Sörman. SAAB's interest in Rodon regards the possibility to use the tool for development and operations of aircraft systems. The main goal of this thesis was to evaluate the capacity of Rodon and determine how SAAB can use the diagnostic tool during development and operations.

The tool uses model based diagnosis with artificial intelligence for fault isolation which is a powerful approach. If Rodon is introduced at SAAB, then detailed models of systems will be necessary to create, including the nominal behavior of the system and different faulty behaviors. In order to achieve high quality fault isolation, it is necessary to have complete and consistent models. To be able to use all applications that Rodon feature for a modeled system, preferable characteristics are that the model should be static, have discrete control signals, and have well defined system behavioral modes.

During development of a system Rodon can be used to improve and easy the work for failure analysis, guidance of sensor placements, evaluation of tests, generation of decision structures, and fault isolation. Since design of tests during development is a desirable application that Rodon does not have, two different methods are presented that utilizes Rodon to generate all possible limit checking tests.

In conclusion, Rodon can be very useful in several different aspects if introduced, but benefits gained by using Rodon will have to be compared to the labor cost of creating good models.

Nyckelord
Keywords
Model based diagnosis, artificial intelligence, diagnostic modelling tool, Rodon, failure analysis, generation of decision structures, guidance of sensor placements, fault isolation.

Abstract

Rodon is a diagnostic tool developed by Sörman Information and Media AB. Saab's interest in Rodon regards the possibility to use the tool for development and operations of aircraft systems. The main goal of this thesis was to evaluate the capacity of Rodon and determine how Saab can use the diagnostic tool during development and operations.

The tool uses model based diagnosis with artificial intelligence for fault isolation which is a powerful approach. If Rodon is introduced at Saab, then detailed models of systems will be necessary to create, including the nominal behavior of the system and different faulty behaviors. In order to achieve high quality fault isolation, it is necessary to have complete and consistent models. To be able to use all applications that Rodon feature for a modeled system, preferable characteristics are that the model should be static, have discrete control signals, and have well defined system behavioral modes.

During development of a system Rodon can be used to improve and ease the work of failure analysis, guidance of sensor placements, evaluation of tests, generation of decision structures, and fault isolation. Since design of tests during development is a desirable application that Rodon does not have, two different methods are presented that utilizes Rodon to generate all possible limit checking tests.

In conclusion, Rodon can be very useful in several different aspects if introduced, but benefits gained by using Rodon will have to be compared to the labor cost of creating good models.

Acknowledgment

We would like thank our university supervisor Mattias Krysander and examiner Erik Frisk for many interesting discussions and guidance through troubled waters. We would like to thank our supervisors Andreas Bergström and Torbjörn Fransson for all guidance at Saab. Big thanks also go out to all others that have helped us during this journey: Birgitta Lantto, all employees at Saab TDCM, Johan Planander, the people at Sörman and Oscar Särnholm. Finally our families and friends deserves a big thank for putting up with all diagnostic yapping during this period.

Daniel Andersson and Patrik Sköld, Linköping, 2007

Contents

Chapter 1

Introduction

Rodon is a diagnostic tool developed by Sörman Information and Media AB. Saab's interest in Rodon regards the possibility to use the tool for development and operations of aircraft systems. An aircraft, such as Saab's JAS 39 Gripen, is a complex system divided into many subsystems. It is of great importance to detect if any of these subsystems malfunction. For this purpose functional monitoring is used to alarm when important subsystems do not behave as expected during flight. If an alarm has been set, the diagnosis problem during maintenance is to find the component causing the faulty behavior through fault isolation.

1.1 Purpose

The thesis project will evaluate the capacity of Rodon and the main goal is to determine how Saab can use Rodon during development and operations.

1.2 Method

The chosen work method was to model various systems in Rodon and analyze them by using different features, and by this way gaining experience and material for a general analysis of the tool. Therefore, the work started by trying to find suitable systems to model by studying various subsystems in JAS 39 Gripen. One electrical system and one hydraulic system were chosen, since it was of desire to cover different system characteristics and behaviors. An ongoing analysis of what Saab was interested in and discussions with the supervisors finally led to the questions that needed answers. New models were developed and analyzed with the purpose to answer questions that arose during the analysis process.

1.3 Thesis Outline

The diagnosis theory necessary for this thesis is presented in Chapter 2. A vision of how a diagnostic modeling tool desirably could be used during the lifecycle of an airplane is given in Chapter 3. Chapter 4 consists of a short introduction to Rodon followed by Chapter 5 that describes how it is to work with Rodon, which benefits the tool has, problems that can occur, and limitations. In Chapter 6, a method is given with the purpose to decide which tests that should be implemented to achieve fault isolation and functional monitoring goals. Chapter 7 evaluates which parts of the vision described in Chapter 3 that can be fulfilled by Rodon. The conclusion of this thesis is then summarized together with suggestions for future work in Chapter 8.

1.4 Contributions

The main contributions of this thesis are

Chapter 3: A presentation of a vision regarding how a diagnostic tool can be used during development and operations of aircraft subsystems.

Chapter 5: Analysis of how it is to work in Rodon, what benefits the tool has, problems that can occur, limitations, and which type of systems that are suitable to model.

Chapter 6: Presentation of a method with the purpose to decide which tests that should be implemented to achieve fault isolation and functional monitoring goals.

Chapter 7: Analysis of which parts of the vision described in Chapter 3 that can be fulfilled by Rodon.

Chapter 2

Theory

This chapter starts with a general introduction of the diagnosis problem. After that, concepts are defined so that model based diagnosis can be explained and hypothesis tests introduced.

2.1 Introduction to the diagnosis problem

A system in general has a nominal behavior, which can be described with equations stating how it operates normally when no faults are present. Systems are built up by components and these behave differently depending on if they function or not. The behavior of the system may deviate from the nominal behavior when components malfunction.

The diagnosis problem is to detect a fault in a system and to locate the cause of it (Frisk, Nyberg, 2002). The use of diagnosis can be several, e.g. to fulfill safety and environment requirements, protect machinery, improve availability and repairability. One approach to solve the diagnosis problem is model based diagnosis, which either can be an approach of type AI (Artificial Intelligence) or FDI (Fault Detection and Isolation). The AI approach will be described in Section 4.2 and the FDI approach will be described in the next section.

2.2 Model based diagnosis by FDI

As stated earlier a system, *sys*, consists of components $c_i, i = 1, 2, .., N$. Each component can behave differently over time and these behaviors can be grouped into component behavioral modes. To denote that a component c_i behaves according to one of its component behavioral modes *bm*, we will write $c_i = bm$.

Let $z = (u\ y)$ be a vector of all known signals, where *u* is control signals and *y* measurements. Control signals in a system can either be active, which means that they are controllable, or passive, which means that they are uncontrollable. The model equations describing $c_i = bm$ can be formulated as $M_i^{bm}(z) = 0$.

If component c_i has k_i different behavioral modes, then we write $c_i \in \{bm_1,...,bm_{k_i}\}$. Each component is assumed to behave exactly according to one of its component behavioral mode, i.e. $c_i \in \{bm_1,...,bm_{k_i}\}$. It is now possible to define the arbitrary system behavioral mode $BM = \{c_1 = bm_{j_1}, c_2 = bm_{j_2},...., c_N = bm_{j_N}\}$, where $1 \le j_i \le k_i, \forall i \in \{1,2,...,N\}$. The true system behavioral mode will be written as $sys = BM = \{c_1 = bm_{j_1}, c_2 = bm_{j_2},...., c_N = bm_{j_N}\}$.The system behavioral mode no fault (*NF*) will be frequently used in this thesis and is defined as $sys = NF = \{c_1 = nf, c_2 = nf,...., c_N = nf\}$, where *nf* stands for the component behavioral mode no fault.

If $sys = BM = \{c_1 = bm_{j_1}, c_2 = bm_{j_2},...., c_N = bm_{j_N}\}$ then the component behavioral modes define which model equations that describe the behavior of each component. The model equations of all components can be combined into a model describing the behavior of the whole system when being in *BM* and this model is denoted $M^{BM}(z) = 0$.

This combination can for *NF* be expressed as

$$M^{NF}(z) = \bigcup_{i \in \{1,2...,N\}} M_i^{nf}(z)$$

and the model equations are written as $M^{NF}(z) = 0$. The corresponding system behavioral mode set is for the model defined as $\Theta_{NF} = \left\{z \mid M^{NF}(z) = 0\right\}$ and consists all *z* consistent with $sys = NF$. In general, Θ_{BM} will denote all *z* consistent with $sys = BM$ for a model and Θ_{BM}^{sys} will denote all *z* consistent with $sys = BM$ for a system *sys*.

In order to perform MBD (Model Based Diagnosis) a premise is that the system has been expressed in model equations describing all components behavioral modes. If the model and system has system behavioral modes $\{BM_1,...,BM_k\}$, then the set of all possible observations is $Z = \Theta_{BM_1} \cup ... \cup \Theta_{BM_k}$.

The definition of a diagnosis is given next.

Definition 2.1:

> *Given an observation $z \in Z$, a system behavioral mode, BM, is a diagnosis if and only if $z \in \Theta_{BM}$.*

A purpose of an on board diagnosis system is given an observation *z* to find all diagnoses, i.e. to find all system behavioral modes that fulfill Definition 2.1. The function of the on board diagnosis system and how it interacts with the system is illustrated in Figure 1.

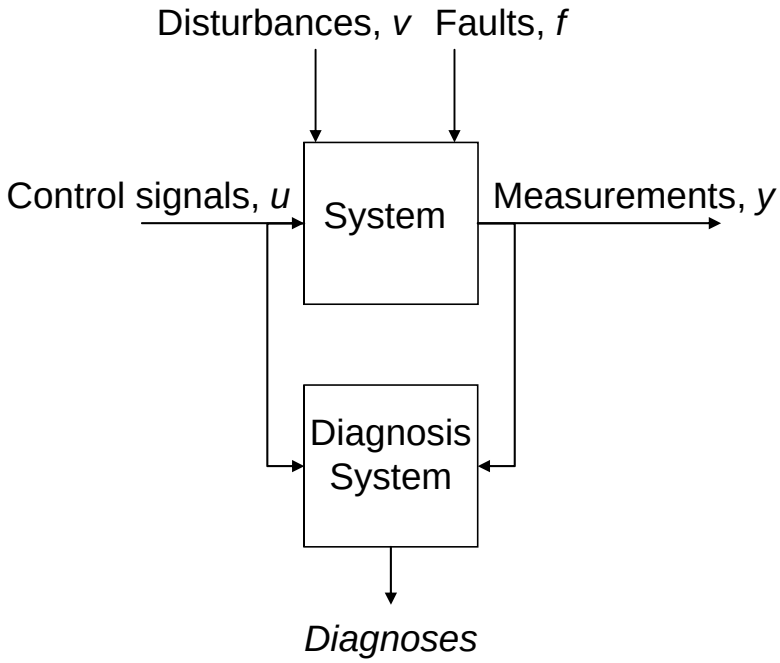


Figure 1. Illustration of how a diagnosis system functions and how it interacts with the system.

An illustration of an on board diagnosis system architecture to achieve FDI can be seen in Figure 2 (Blanke et al. 2003). The architecture of the on board diagnosis system consists of two parts: tests and a fault isolation logic block. Tests are constructed with the purpose to reject system behavioral modes that can not explain the system behavior and the fault isolation logic sorts and generates the diagnoses from the test results.

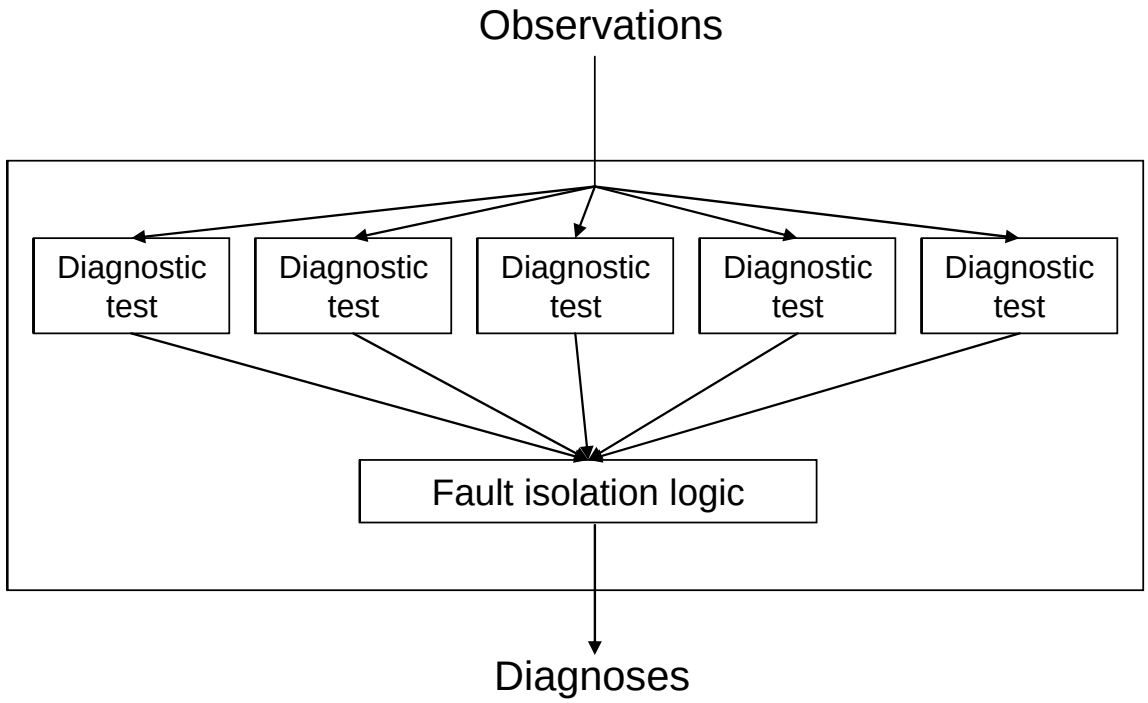


Figure 2. Architecture of an on board diagnosis system for FDI.

2.2.1 Hypothesis test

The model equations can together with observations, from sensors, be used to create tests. The purpose with the tests of an on board diagnosis system is to detect and alarm for certain behavioral modes by comparing the observations with the expected values of the observations. A test will react if this comparison renders a sufficient difference.

In diagnosis, a null hypothesis H^0 is an assumption that the system is behaving in accordance with one of the system behavioral modes in a set S^0 of system behavioral modes and its alternative hypothesis H^1 is that it behaves accordingly to one of the system behavioral modes in S^1. A hypothesis test T is used to decide if H^0 can be rejected or not. The null hypothesis is rejected if $z \in \Theta_T^C$, where Θ_T^C is the rejection region of T. If the test reacts, then H^0 will be rejected and the conclusion will be that H^1 is true and $sys \in S^1$.

The hypothesis can be written as:

$H^0 : sys \in S^0$ *Some behavioral mode in* S^0 *can explain observations*

$H^1 : sys \in S^1$ *No behavioral mode in* S^0 *can explain observations*

This means that the set Θ_T should be a superset of all Θ_{BM} where $BM \in S^0$.

Next we exemplify a proper design of a test including the choices of S^1 and Θ_T^C.

Assume a system, *sys*, that has four different system behavioral modes: *NF* (No Fault), F_1, F_2, and F_3. A hypothesis test T_1 has been created with the purpose to detect and alarm for F_1 and F_2. The rejection region of the test is $\Theta_{T_1}^C$ and can be seen in Figure 3 together with the behavioral mode sets.

Since Θ_{F_1} is a subset of $\Theta_{T_1}^C$ and Θ_{F_2} has a non empty intersection with $\Theta_{T_1}^C$, these two system behavioral modes, F_1 and F_2, will be included in S^1. Analogous, since Θ_{NF} and Θ_{F_3} are subsets of Θ_{T_1}, and Θ_{F_2} has a non empty intersection with Θ_{T_1}, these three system behavioral modes will be included in S^0. In summary the rejection region of the test results in $S^1 = \{F_1, F_2\}$ and $S^0 = \{NF, F_2, F_3\}$.

If the system behaves according to F_1 then $z \in \Theta_{F_1} \subset \Theta_{T_1}^C$ and this results in rejecting the null hypothesis and the decision will be that $sys \in S^1 = \{F_1, F_2\}$.

The null hypothesis will only be rejected for $sys = F_2$ when $z \in \Theta_{F_2} \cap \Theta_{T_1}^C$. This means that the test will fail to react and detect $sys = F_2$ when $z \in (\Theta_{F_2} \setminus \Theta_{T_1}^C)$ and as a consequence alarms will be missed.

The test will never react when $sys \in \{NF, F_3\}$ and the null hypothesis will not be rejected and the decision will be that $sys \in S^0 = \{NF, F_2, F_3\}$.

In summary this means that the following conclusions can be drawn from the test:

$z \in \Theta_{T_1}^C \rightarrow H^0 rejected, sys \in S^1 = \{F_1, F_2\}$

$z \notin \Theta_{T_1}^C \rightarrow H^0 not\ rejected, sys \in S^0 = \{NF, F_2, F_3\}$

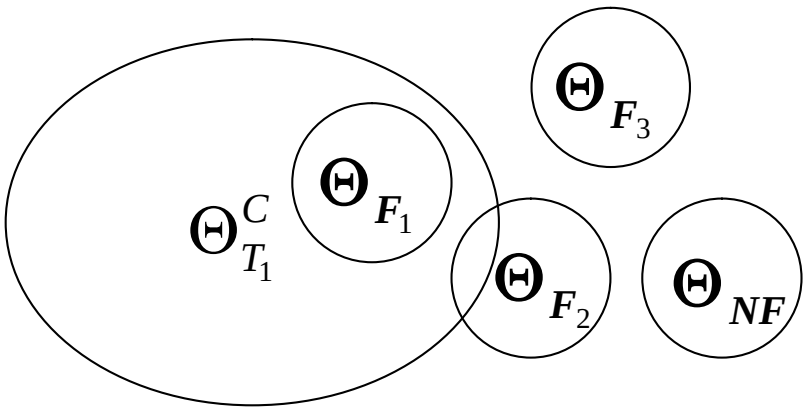


Figure 3. A system with four behavioral mode sets and the rejection region for the test used in the diagnosis system.

2.2.2 Decision structure

Let 0, X and 1 symbolize that a system behavioral mode will never, sometimes and always cause rejection of the null hypothesis respectively. Then the example presented in Section 2.2.1 can be described by the decision structure in Table 1.

Table 1. Decision structure of the test in the example in section 2.2.1.

	NF	F_1	F_2	F_3
T_1	0	1	X	0

Fault isolation is to determine which component or components that causes a faulty behavior. This can be done by using a decision structure. Assume that the diagnosis system in the example in Section 2.2.1 is supplemented with two new tests T_2 and T_3. The resulting decision structure can be viewed in Table 2.

Table 2. Decision structure of how behavioral modes NF, F_1, F_2, and F_3 affect the tests T_1, T_2, and T_3.

	NF	F_1	F_2	F_3
T_1	0	1	X	0
T_2	0	0	X	X
T_3	0	X	0	X

Two examples of how the information in the decision structure can be used for fault isolation analysis:

If tests T_1 and T_2 have reacted, the conclusion that can be drawn is $sys = F_2$ since no other system behavioral mode can cause both these tests to react. Hence, F_2 is said to be fault isolated.

If only test T_2 has reacted, the conclusion that can be drawn is that $sys = \{F_2, F_3\}$ since they both can cause the test to react. Hence, no fault isolation can be made since F_2 and F_3 both will be suspected system behavioral modes.

The purpose with the fault isolation logic is to perform this fault isolation analysis automatically based upon the decision structure and the values of the corresponding diagnostic tests.

2.2.3 Summary

This chapter has defined the basis for model based diagnosis by FDI. The on board diagnosis system was introduced and it includes diagnostic tests and fault isolation logic. It is of interest to determine whether or not a diagnostic tool can be used for construction of an on board diagnosis system for FDI. A vision will follow in the next chapter, regarding how a diagnostic tool could be used for diagnosis system development.

Chapter 3

Usage of a diagnostic tool for subsystem development and operations

An aircraft contains several complex subsystems containing diagnostics. This chapter presents a vision of how a diagnostic tool can be used during development and operation of subsystems. It concludes with a summary of the desired achievements of the tool.

3.1 The vision

This vision will refer to the development environment which is where the development of a system takes place and it contains various tools. Operations take place in the flight environment of the aircraft and the maintenance workstation, which is where maintenance is performed with the help of various tools. An aircraft subsystem has been developed from definitions and requires diagnostics to be integrated in the flight environment of the aircraft. Construction of the diagnosis system is performed in a development environment with the help of a diagnostic tool and a model of the subsystem is therefore implemented in the tool in the development environment.

It is important to analyze the behavior of the subsystem to determine which faults that can cause subsystem failure, which is when the function of the system significantly deviates from normal operation. This analysis is mostly performed manually at Saab today and the vision is to use a diagnostic tool instead. Found flaws will if necessary and possible be adjusted by redesign of the subsystem, resulting in a need to reiterate the modeling and analysis in the development environment. The model is upon completion passed on to the diagnostic tool of the maintenance workstation.

Faults that cause system failure are important to detect since they can result in loss or degradation of the subsystem functionality, which in a worst case scenario could be equivalent to loss of an aircraft. Detection of system failures are called functional monitoring and must be performed by the on board diagnosis system. This can hopefully be achieved by creating tests that react when functionality is lost. Tests in the on board diagnosis system can also be constructed with the purpose to isolate faults down to a specific replaceable system component (At Saab denoted as a line replaceable unit, LRU).

Tests for functional monitoring and for fault isolation must be designed for the on board diagnosis system, to achieve FDI, and this has been a manual process up to now but the vision is to use a diagnostic tool instead.

All designed tests, for fault isolation and functional monitoring purposes, constitute a specification of which tests that need to be implemented in the on board diagnosis system in the flight environment and also decide which sensors that must be implemented in the subsystem. Further sensors might be implemented and the diagnostic tool can perhaps provide an indication of suitable placements.

The decision structure for the designed tests is generated from the diagnostic tool and is used in the fault isolation logic, situated inside the on board diagnosis system, to provide diagnosis data which can be separated into two parts: functional monitoring data and fault isolation data. During flight, the sensors of the subsystem in the flight environment provide observations. These observations are recorded in the Maintenance Data Record (MDR), which can be seen as the hard drive of the aircraft for storage of operational data. Functional monitoring data is processed to inform the pilot of system failures and suitable recovery actions. Fault isolation and functional monitoring data is stored in the MDR.

Data from MDR is loaded into the MDR ground, in the maintenance workstation, after landing. During maintenance at ground, selected MDR data is inserted in the model of the diagnostic tool to achieve fault isolation through an AI model based diagnosis approach. The selected MDR data contains functional monitoring and fault isolation data of interest for the subsystem together with observations. The idea is to perform new measurements manually at ground in the subsystem and insert these in the model of the diagnostic tool to further reduce the suspected component behavioral modes. This means that fault isolation is performed at two instances, both on board by the diagnosis system and off board, and the reason is that there is a requirement of basic fault isolation on board in the flight environment without further aid of off board tools such as e.g. a maintenance workstation or a diagnostic tool.

The diagnostic tool provides the suspected component behavioral modes, summarized in the diagnostic report, and the faulty components are replaced by the technician. Finally, feedback is provided from the maintenance department on model improvements.

3.2 Desired features included in a diagnostic tool

The desired features included in a diagnostic tool can be summarized as:

- Performance of failure analysis
- Guidance for sensor placements
- Design and evaluation of tests for an on board diagnosis system
- Generation of the decision structure corresponding to the diagnostic tests of the on board diagnosis system. This decision structure is to be used for fault isolation
- Achieve high quality fault isolation during maintenance by an AI model based diagnosis approach

This thesis will try to determine if Rodon can fulfill the desired requirements of the vision, and an introduction of the tool and how it performs AI model based diagnosis will follow in the next chapter.

Chapter 4

Introduction to Rodon

This chapter describes how modeling and simulation is performed in Rodon. Rodon is a diagnostic modeling tool available from the company Sörman Information and Media AB. It is an object oriented tool that allows creation of components in the modeling language Rodelica, a derivate of Modelica.

4.1 Modeling

The work process in Rodon starts with the task of modeling all components that are needed in the model and are not included in the standard libraries. After this is completed, the model can be assembled graphically through drag and drop or by textual programming. The model can upon completion be used for simulation purposes and generation of data, which will be presented later on in this chapter.

4.1.1 Components

Different behaviors of a component are, as said before, called component behavioral modes. For each *bm* a set of equations holds. The component behavioral modes and their equations are implemented in Rodon when a component class is created. How the Rodelica code for a component class $X = bm \in \{nf, f_1, ..., f_N\}$ generally is built up is showed in Figure 4. The code is divided into two sections, where the first section is used for declaration of attributes and the second section, behavior, is where the different component behavioral mode equations $M_X^{bm}(z) = 0, \forall bm \in \{nf, f_1, ..., f_N\}$ are implemented.

```
Model X
    Declarations of attributes (public, protected or private)
Behavior
    if (behavioral mode nf)
            Equations that hold for nf

    if (behavioral mode f1)
            Equations that hold for f1
                    •
                    •
                    •
                    •
    if (behavioral mode fN)
            Equations that hold for fN
End Model X;
```

Figure 4. General code of how a component class is implemented in Rodon.

4.1.2 Creation of a model

A model can be created graphically by drag and drop or by textual programming. The object oriented feature in Rodon provides easy model alteration, e.g. if the model consists of several hundreds objects of the same type that need to be modified then it is only necessary to change the base class to alter all objects. The inheritance feature allows the user to create new altered class abstractions of already existing component base classes.

An example of a model can be seen in Figure 5. It is built up by a battery, wire, resistor (100 Ohm) and ground. All components in the model have a nominal, fault free, component behavioral mode and this means for the battery and ground that they provide 12 V respectively 0 V. The wire's nominal behavior is that the voltage is the same in both ends and that the sum of the current that flow in and out equals zero. The nominal behavior of the resistor is that Ohm's law applies. Furthermore the battery, the wire, the resistor and the ground all contain the behavioral mode disconnected, which states that no current can flow through the component.

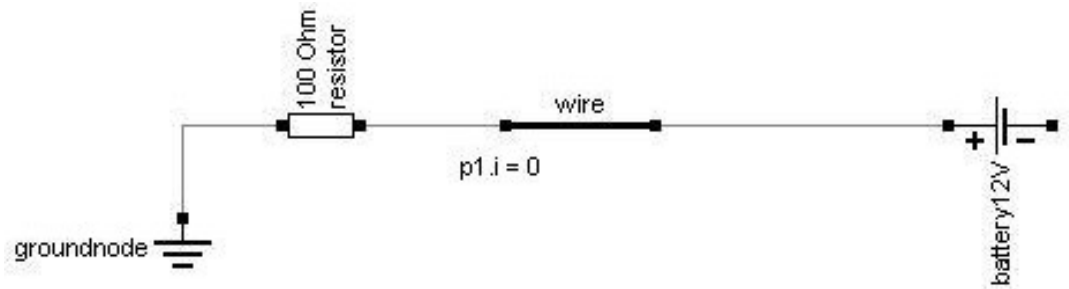


Figure 5. Illustration of a model in Rodon.

4.1.1 Connectors and connections

Consider electrical circuits containing resistors, capacitors and inductors. The formulas used to describe these circuits are basically the relations between the variables voltage *u*, Volt, and current *i*, Ampere. Generally these variables are called potential and flow quantities. For many different physical domains the physical constraints have the same fundamental structure (Ljung, Glad, 2004). In this thesis only electrical and hydraulic systems will be handled and in Table 3 the variable names for each respective quantity in both domains are listed.

Table 3. Variable names, potential and flow quantities for the two different physical domains handled in this thesis.

Domain	Potential quantity	Variable name	Flow quantity	Variable name
Electrical	Voltage (V)	U	Current (A)	i
Hydraulics	Pressure(P)	P	Mass flow rate(kg/s)	$\dot{m}$

Connector is a type of class defined in Rodon containing only declarations of potential and flow variables. Graphically connectors are represented as squared dots on the side of the component icons, see for example Figure 5. Its purpose is to create connections between the equations of different components, in order to generate the system behavioral mode equations $M^{BM}(z) = 0$ for each *BM*.

Connectors that are interlinked form a connection. For each connection, equations are generated with the purpose of describing how the connector variables will be propagated between each other. The connection equations follow these conventions (Rodon manual):

- All potential variables of a connection are set equal.
- A zero-balance equation is automatically created for all flow variables of a connection.
- The value of flow variables is positive, if the flow is directed into the connector of a component.

How the Rodelica code for a connector class *port* is generally built up is described in Figure 6.

```
Connector port
Potential quantity p;
Flow quantity f;
end port;
```

Figure 6. General description of how a connector class can be built up.

Connectors are declared as attributes in the component classes. Since connector classes have no behavioral modes, connection equations are independent of which system behavioral mode that is present and will never alter.

A general example: the connection *connect(p1,…, pN)* , where *p1,…,pN* all are connectors of class port, connects the ports so that they form one node, creating the equations (Bunus, 2002):

$$p1.p = ... = pN.p;$$
$$p1.f + ... + pN.f = 0;$$

Connectors of class pin, declared in Figure 7, are used for electrical systems.

```
Connector pin
Voltage u;
flow current i;
end pin;
```

Figure 7. Description of how a connector class for electrical systems can be built up.

For an electrical component *c* with pin *p1* the notation *c.p1.i* and *c.p1.u* is used for the current and voltage of the pin. As an example of an electrical component, the component class of a resistor is described in Figure 8.

```
Model resistor
pin        p1, p2;
Resistance   r;
Behavior
p1.i + p2.i = 0;
p1.u - p2.u = r*p1.i;
end resistor
```

Figure 8. Example code of a resistor class.

4.1.2 Property marks

In Rodon property marks are used for assigning roles to model attributes. If an attribute in a model has a property of special interest for analysis purposes it can be tagged with a property mark. This could be used for a value of a control signal, an observation, a component behavioral mode or a flag representing if a test has reacted. These are only a few of all the property marks that are defined in Rodon, and a full listing can be found in Appendix B. An application where the use of property mark is crucial is Auto simulation, described in section 4.4.1. The model in Figure 5 will be used in a small example to illustrate how a component attribute can be tagged with a property mark.

Let's assume that the current on the left hand side of the wire is measured by a sensor in the real system as is indicated in Figure 5. Then the attribute *wire.p1.i* should be tagged with an observation property mark in the Rodon model.

4.2 Model based diagnosis in Rodon

As said in Section 2.1, Rodon uses an AI approach to perform model based diagnosis and how this is done will be described in this section. The algorithm used is based on ATMS (Assumption Based Truth Maintenance System) (Hamscher et al., 1992). In summary ATMS is a data base containing the model equations, $M^{BM}(\hat{z}) = 0$, and an example of one is given in Section 4.2.1. Measurements $\hat{y}$ are inserted in the data base and these values are propagated during simulation until all intermediates have been calculated and fulfill a certain tolerance. Inconsistencies will result in conflicts including potential suspects.

4.2.1 Simulation

In Rodon simulation is done through local value propagation. Local value propagation is an iterative solving process, where solutions are approached by eliminations of impossible values by systematical examination of each single relation in the model. Rodon tries to solve the model equations for each component. The calculated values are then propagated back and forth between the different components in order to find solutions that hold for all equations in the modeled system. This is done until all variable values have been generated for

$$\hat{\Theta}_{BM}^{u_0} = \left\{ \hat{y} \,\middle|\, M^{BM}(u_0, \hat{y}) = 0 \right\},$$

Interval arithmetic is used to apply value ranges on variables.

Every component in Rodon can be set to comply with one of its behavioral modes. This allows the user to simulate a model with any of the implemented system behavioral modes. Hence, no new model is needed to simulate different behaviors. If nothing else is set, Rodon simulates the nominal fault free behavior $M^{NF}(u_0\ \hat{y}) = 0$ as a default assignment.

Consider the system modeled in Figure 5. For the system behavioral mode $NF = \{groundnode = nf, resistor = nf, wire = nf, battery = nf\}$, the model equations $M^{NF}(\hat{z}) = 0$, where component equations are expressed on the form $\langle equation, \{component\ behavioral\ mode\}\rangle$, are as follows:

$$\langle groundnode.p1.u = 0, \{groundnode = nf\}\rangle \quad (4.1)$$

$$\langle resistor.p1.u - resistor.p2.u = 100 \cdot resistor.p1.i, \{resistor = nf\}\rangle \quad (4.2)$$

$$\langle resistor.p1.i + resistor.p2.i = 0, \{resistor = nf\}\rangle \quad (4.3)$$

$$\langle wire.p1.u = wire.p2.u, \{wire = nf\}\rangle \quad (4.4)$$

$$\langle wire.p1.i + wire.p2.i = 0, \{wire = nf\}\rangle \quad (4.5)$$

$$\langle battery.p1 = 12, \{battery = nf\}\rangle \quad (4.6)$$

$$groundnode.p1.u = resistor.p1.u \quad (4.7)$$

$$groundnode.p1.i + resistor.p1.i = 0 \quad (4.8)$$

$$resistor.p2.u = wire.p1.u \quad (4.9)$$

$$resistor.p2.i + wire.p1.i = 0 \quad (4.10)$$

$$wire.p2.u = battery.p1.u \quad (4.11)$$

$$wire.p2.i + battery.p1.i = 0 \quad (4.12)$$

Equations (4.7)-(4.12) are created by the connector connections in accordance to Section 4.1.1.

After a simulation Rodon will return one of the following status messages, simulation successful, conflict detected or simulation aborted. The interpretation of each status messages is as follows:

- *Simulation successful* is returned when Rodon has completed all of its calculation processes without finding any contradictions.
- *Conflict detected* is returned as a simulation result when a variable contradiction has been localized during the solving process.
- *Simulation aborted* is returned when Rodon is not able to end the simulation. The cause of this can be that there is a calculation loop somewhere in the model.

4.2.2 Conflicts

A conflict in Rodon is, as stated earlier, generated when a contradiction of a variable relation is located. In the case when only control signals and behavioral modes are used as input signals, conflict detected means that there exists no solution $\hat{y}$ to $M^{BM}(u\,\hat{y}) = 0$, and is an indication of an incorrect model. If the Rodon model is to be used for diagnosis purposes a conflict should only be caused due to an inconsistency between the calculated model variable values and measured observations from the real system. Hence, a conflict detected should only be the result of simulation if a measurement $y \notin \hat{y}$ is inserted in the model $M^{BM}(u\,\hat{y}) = 0$.

An example of a conflict can be illustrated with the model in Section 4.2.1. The current through the wire has been measured to 0 A and is introduced in the Rodon model as

$$\text{wire.p1.i} = 0 \tag{4.13}$$

is added as a condition in the solving process. A conflict is generated during simulation because the measurement is inconsistent with the system behavior *NF*. Equations (4.3), (4.10), and (4.13) results in

$$resistor.p1.i = 0 \tag{4.14}$$

This results together with Ohm's law, equation (4.2), that there can not be any potential difference over the resistor

$$resistor.p1u = resistor.p2.u \tag{4.15}$$

The value of *groundnode.p1.u* propagated through equations (4.1), (4.2), (4.7), (4.9) and (4.15) result in

$$wire.p1.u = 0 \tag{4.16}$$

At the same time

$$wire.p1.u=12 \tag{4.17}$$

is propagated from the *battery.p1.u* through equations (4.4), (4.6), and (4.11). This results in a conflict since equation (4.16) and (4.17) contradict each other. Rodon marks all components in which equations are used to calculate the variable or variables in conflict. These components are all potential suspects, in which faults can be present, since they all affect the calculation of the variable. In this example all components will be possible suspects.

4.2.3 Diagnosis

A diagnosis can be computed in Rodon when a conflict has occurred during simulation. As described in Section 4.2.2 Rodon will mark the components in which equations have been used to calculate the variable or variables where the contradictions have occurred. To compute a diagnosis, the tool evaluates different system behavioral modes in order to find which of the marked components that should remain as suspects. If a system behavioral mode is inconsistent with the observations then it is ruled out as a diagnosis in the process. The system behavioral modes that are consistent with the observations are presented as the diagnosis results.

To illustrate this, a diagnosis can be computed on the conflicts in the example in Section 4.2.2. All components in the system have been marked as possible suspects. The model equations will be altered for each system behavioral mode that Rodon tests. When testing the system behavioral mode
$wire.disconnected = \{groundnode = nf, resistor = nf, wire = disconnected, battery = nf\}$
the model equations (4.4) and (4.5) from Section 4.2.1 are, since a disconnection means that no current can flow trough the wire, replaced by the equations:

$$\langle wire.p1.i = 0, \{wire = disconnected\}\rangle \quad (4.18)$$

$$\langle wire.p2.i = 0, \{wire = disconnected\}\rangle \quad (4.19)$$

This change leads to consistency between the model and the observation *wire.p1.i = 0* which results in *wire = disconnected* being a diagnosis.

Other system behavioral modes will also be consistent with the observation and Rodon's complete list of diagnoses of single faults will be that any of the components *groundnode, wire, resistor,* or *battery* could be disconnected.

4.3 Dynamic simulation

A fault in one component can cause other components to break and malfunction. Consider the system that has been modeled in Figure 9. If *wire1= short to ground,* the high current between *wire1* and *battery* will cause the fuse to blow. As a result, there will no longer be a current flowing through the circuit.

Rodon features the possibility to simulate this kind of casual sequence of events mentioned in the example above. This is done by dividing each course of event into a frame. An attribute a_i can be updated between frames if it is complemented with an attribute $a_i Next$, which sets the value of a_i in the next frame. In Rodon this is called pseudo dynamic simulation and can be illustrated with an example on the system introduced in Figure 9.

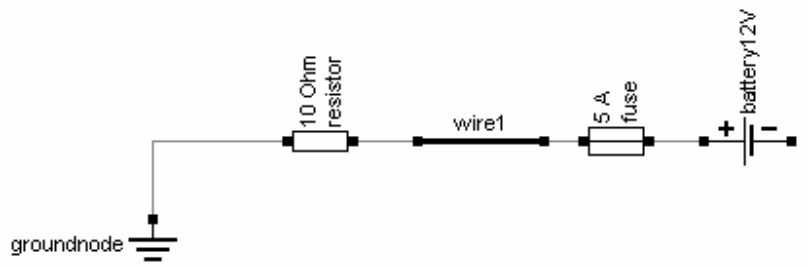


Figure 9. A model of an electrical circuit containing a battery, fuse, wire, resistor and a ground node.

The *fuse* has an attribute $state \in \{ok, blown\}$, complemented with an attribute $stateNext \in \{ok, blown\}$, which controls the current flowing through it. If *state=ok*, then the current can flow through the fuse, but if *state=blown*, then no current will flow through the fuse. A too large current through the fuse will result in *stateNext=blown*. When the model behavior of the nominal fault free case is simulated the current through the circuit is 1.2 A. Simulating the component behavioral mode *wire1=short to ground* results in a high current between *wire1* and the *battery* and will set *stateNext=blown* in *frame1*. The value of *stateNext* in frame1 will set *state=blown* in *frame2*. As a result, no current will be able to flow through the fuse. The fuse status in the two different frames is shown in Figure 10.

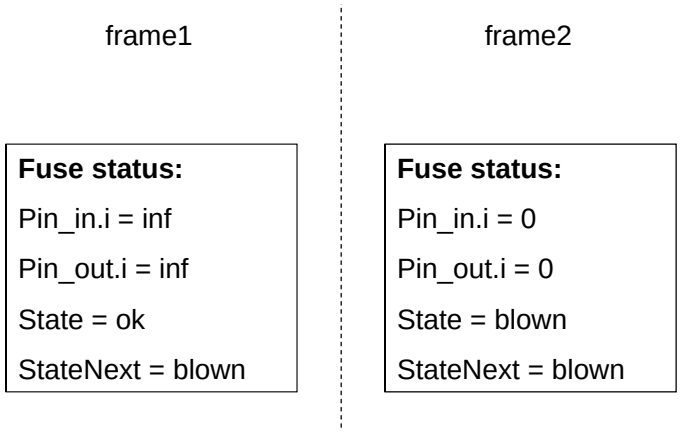


Figure 10. An example of how a fuse status changes between different frames when it blows.

The example above describes what could be called an event based simulation of how a system evolves over time. This simulation process is called pseudo dynamic simulation and it generates the attribute values for each event, but is not a dynamic simulation in the sense that it does not describe how a system behavior varies with time in between events. However, the same feature can also be used for time varying dynamic simulation by updating variables with the *Next* suffix. This allows dynamic simulation by the user of discretized state space systems.

4.4 Applications

In this section different Rodon applications are presented.

4.4.1 Auto simulation

Auto simulation is a feature in Rodon that is used to analyze how different combination of predefined input signals will affect the output signals. To be able to use a certain attribute in the auto simulation it needs to have been given a property mark. Which attributes that are going to be used as inputs and outputs are sorted by their property marks and defined by the user in a definition file. There are in Rodon two combinatorial possibilities available for input signals, cross product or concatenation. The first one means that all combinations of the input signals are simulated and the second that each one is simulated separately. This is perhaps best described through an example.

Consider an arbitrary system controlled by the user with two switches, switch 1 and switch 2. These switches can either be off, in position 1, or in position 2, and their position will be defined as an input signal. If these are combined as the cross product then auto simulation will simulate and decide the resulting outputs for the following nine different control signal combinations:

Switch 1	Switch 2
off	off
off	pos 1
off	pos 2
pos 1	off
pos 1	pos 1
pos 1	pos 2
pos 2	off
pos 2	pos 1
pos 2	pos 2

If the input instead were defined as the concatenation then the control signal combinations would be:

Switch 1	Switch 2
off	off
pos 1	off
pos 2	off
off	pos 1
off	pos 2

The auto simulation principle is illustrated in Figure 11 where control signals *u* and behavioral modes *BM* have been chosen as inputs to calculate selected outputs *y*. The chosen input signals are combined as cross products.

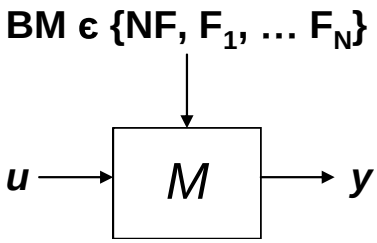


Figure 11. The auto simulation feature in Rodon calculates the selected outputs, y, for selected inputs, u and BM.

All data that is generated by auto simulation is stored in a SDB (State Database). An example of an SDB can be seen in Table 4 where the control signal *u* and component behavioral modes have been chosen as inputs and are listed in the first two columns from the left. In the three remaining columns calculated outputs of *wire1.p1.i, wire1.p1.u, and wire2.p2.i*, which have been tagged with the observation property mark, are listed.

Table 4. SDB generated for a model where *u* and behavioral modes *bm* have been chosen as inputs and *wire1.p1.i, wire1.p1.u,* and *wire2.p2.*i are calculated outputs.

Behavioral modes	U	wire_1.p1.i	Wire_1.p1.u	wire_1.p2.i
System OK	off	0	12	0
bulb disconnected	off	0	12	0
Groundnode_1 disconnected	off	0	12	0
switch disconnected	off	0	12	0
switch pin_short	off	[0.83166 0.83168]	[11.975 11.977]	[-0.83168 -0.83166]
wire_1 disconnected	off	0	12	0

wire_1 short_to_gnd	off	[416.66 416.67]	0	0
wire_1 short_to_batt	off	0	[11.999 12]	0
wire_2 disconnected	off	0	12	0
wire_2 short_to_gnd	off	0	12	0
wire_2 short_to_batt	off	0	12	0
wire_3 disconnected	off	0	12	0
wire_3 short_to_gnd	off	0	12	0
wire_3 short_to_batt	off	0	12	0
System OK	on	[0.83166 0.83168]	[11.975 11.977]	[-0.83168 -0.83166]
bulb disconnected	on	0	12	0
Groundnode_1 disconnected	on	0	12	0
switch disconnected	on	0	12	0
switch pin_short	on	[0.83166 0.83168]	[11.975 11.977]	[-0.83168 -0.83166]
wire_1 disconnected	on	0	12	0
wire_1 short_to_gnd	on	[416.66 416.67]	0	0
wire_1 short_to_batt	on	[0.3723 0.47518]	[11.986 11.99]	[-0.83259 -0.83237]
wire_2 disconnected	on	0	12	0
wire_2 short_to_gnd	on	[416.66 416.67]	0	[-416.67 -416.66]
wire_2 short_to_batt	on	[0.3723 0.47518]	[11.986 11.99]	[-0.47518 -0.3723]
wire_3 disconnected	on	0	12	0
wire_3 short_to_gnd	on	[0.83166 0.83168]	[11.975 11.977]	[-0.83168 -0.83166]
wire_3 short_to_batt	on	[0.83166 0.83168]	[11.975 11.977]	[-0.83168 -0.83166]

4.4.2 Auto generation of a model from a specification list

A specification list is a listing of system components. Rodon features the possibility to automatically generate a model from a specification list. E-Cad is an electrical modeling tool and can be used to generate specification lists, called E-Cad lists, from which Rodon models can be automatically generated.

4.4.3 Decision tree

The purpose with a decision tree is to aid the technician with fault isolation. It provides measurement suggestions to decrease the number of suspected faulty components. Decision trees can be created from an SDB generated with auto simulation, described in Section 4.4.1. No extra modeling is needed. An example of a Rodon generated decision tree can be seen in Figure 12.

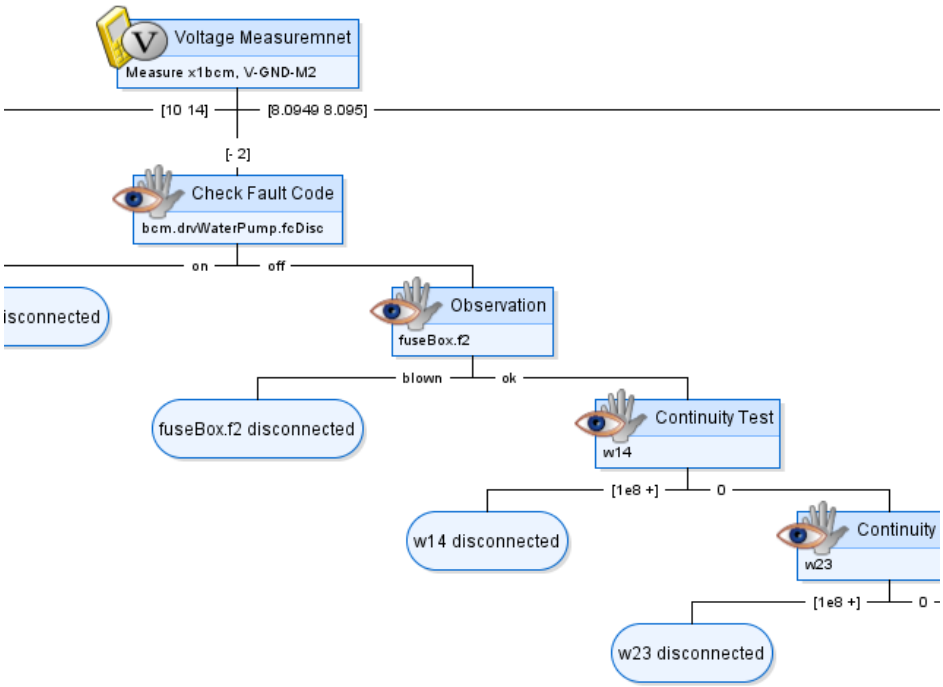


Figure 12. An example of a decision tree generated with Rodon.

4.4.4 Fault tree analysis

Fault tree analysis illustrates the relation between a critical event and the cause of it. It is used to perform quantitative and qualitative analysis of complex systems. Rodon can be used for fault tree analysis, but requires a different modeling approach then the one described earlier in this chapter and will not be further investigated in the thesis.

4.4.5 Failure mode effect analysis

The SDB data can be filtered in Rodon to be used as a decision basis for a FMEA (Failure Mode Effect Analysis), where the effects off all failure modes are listed.

4.4.6 Test implementation

When modeling a system, it is possible to implement diagnostic tests in the model. A test is designed to react if the specified conditions are fulfilled. In Rodon a flag is used to indicate if a test has reacted, and the flag is property marked as a failure code *fc*. If the test *T* with the rejection region Θ_T^C has been implemented in Rodon, then the failure code fc_T will be set true when $z \in \Theta_T^C$. This can be summarized as

$$T: \quad z \in \Theta_T^C \rightarrow fc_T = true$$

4.4.7 Dirigent

As said in Section 4.4.6, in Rodon a flag is used to indicate if a test has reacted, and the flag is property marked as a failure code *fc*. For a test *T* that has been implemented in a model of system *sys*, Rodon offer the possibility to generate a list of the system behavioral modes S^1 that can cause the test to react. This is done in the feature Dirigent which through simulations investigates which behavioral modes that are consistent with the condition $fc_T = true$. A system behavioral mode *BM* is added to the set S^1 if $\Theta_{BM} \in \Theta_T^C$. The information generated by Dirigent

$$fc = true \rightarrow sys \in S^1$$

is in Rodon called decision rules and the behavioral modes in the set S^1 are called suspects. An analogy can be made with decision structures described in Section 2.2.2, since the information a decision rule contain is very similar to the information in a decision structure. The only difference is that a diagnostic rule does not describe how well the rejection region Θ_T^C of a test *T* covers the set Θ_{BM} for a behavioral mode $BM \in S^1$.

4.4.8 Automatic code generation

Rodon features the possibility to automatically generate *c struct code* of diagnostic rules generated in Dirigent, described in Sections 4.4.7. This c struct code can be generated in the feature DRsimulator. The resulting code defines which faulty components that can cause each implemented test to react.

4.5 Summary

In this chapter an introduction to Rodon has been presented. The purpose has been to give an understanding to how Rodon works and which applications that are included. The following chapter will describe how it is to work with Rodon, which benefits the tool has, problems that can occur, and limitations.

Chapter 5

Using Rodon

This chapter will describe how it is to work in Rodon, what benefits the tool have, problems that can occur, limitations, and which type of systems that are suitable to model.

5.1 Generally

Analyzing a system manually, in order to predict which faults that can lead to serious failures is a problem whose complexity increases with the size of the system. The advantage of using a diagnostic modeling tool such as Rodon is that it gives an opportunity to easy structurize this problem. By using the already existing knowledge on component level of which faults that can occur in order to build a Rodon model, the effects each fault will have on the system can be automatically generated, see Section 4.4.1. Because each component is modeled locally, no analytical calculations need to be made by the user in order to derive model equations for the whole system, since this is done by Rodon, see Section 4.2.

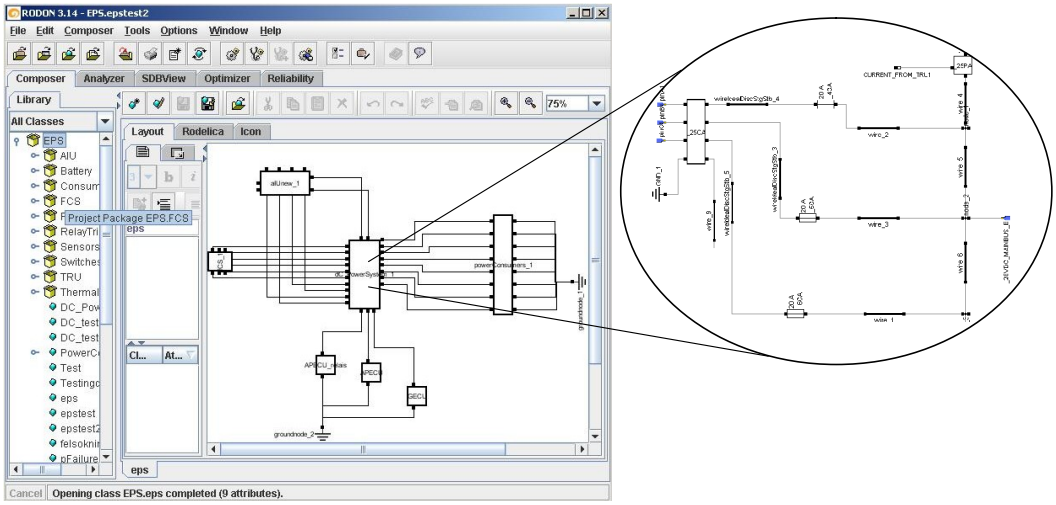


Figure 13. An illustration of hierarchal modeling in Rodon.

The use of connector variables makes the tool quite intuitive for users with modeling background because of the close analogy with bond graphs. More information about bond graphs can be found in (Ljung, Glad 2004). For modeled systems with high complexity Rodon offers a possibility to divide the model into different submodels, and by this way create a hierarchal model. This makes it easier to verify the model since each sub model can be verified separately, and it also creates a good overview. An example of this is showed in Figure 13.

Rodon is, as stated earlier, based on the programming language Rodelica which provides an object oriented modeling approach and this allows the user great freedom when creating class components and which equations they should abide. A necessary condition for utilizing components in a model is that the equations are consistent.

Modeling freedom is an advantage since the user is not depending on somebody else developing component libraries and there are no restrictions, other than syntactical, when creating components. At the same time modeling freedom has drawbacks since it is easy to lose perspective on how different components can be combined to form a consistent model.

5.2 Modeling

Before modeling can start in Rodon, behavior equations need to be specified of how each component in the system behaves. The specification might include not only the nominal fault free behavior but also knowledge of which faults that can occur and how they will affect the system.

Depending on the analysis purpose different level of details of system modeling is required. During the design phase when analyzing how different faults affects a system, e.g. by using an SDB generated from a model as a decision base for FMEA, the exact knowledge of how a fault affect a system might not be needed, since the objective is to capture the primary characteristics. But if a model is to be used for fault isolation for maintenance purposes more details are needed in order to achieve a high fault isolation performance, this will be discussed further in sections 5.2.3 - 5.2.5.

Describing how the failure modes influence the system can be a difficult task since the description should be general and valid for all variations of the fault. One approch to solve this is through modeling of degradation and another is through implementation of undescribed failures modes. These two different approaches will be discussed in the two following sections.

5.2.1 Modeling of degradation with failure parameter

Modeling degradation is to describe how much a fault affects a system in a general way so that it is valid for all variations of the fault.

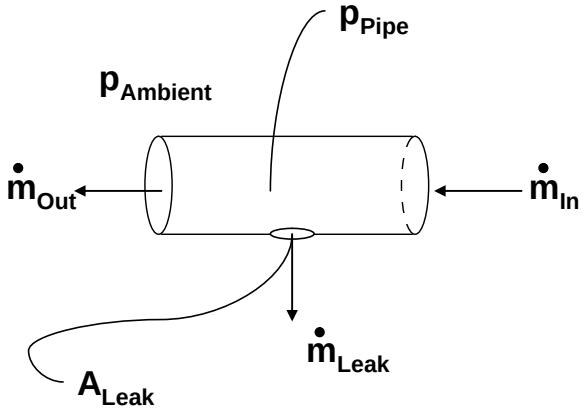


Figure 14. Illustration of a pipe.

An example could be the modeling of a component pipe illustrated in Figure 14. Assume that $p_{Pipe} > p_{Ambient}$. Nominal, fault free, behavior *nf* is that the mass flow rates equal $\dot{m}_{In} = \dot{m}_{Out}$. The pressure p_{Pipe} and temperature T is approximated to be constant throughout the pipe. One fault that can occur is the failure mode leakage f_{Leak}. The effect of a leakage will be that the mass flow rates equal $\dot{m}_{In} = \dot{m}_{Out} + \dot{m}_{Leak}$ and that the pressure in the pipe will drop. The mass flow rate $\dot{m}_{Leak}$ will differ depending on the leakage area A_{Leak} of the hole according to the equation:

$$\dot{m}_{Leak} = \frac{A_{Leak} \cdot C\sqrt{p_{Pipe}^2 - p_{Ambient}^2}}{\sqrt{T}}$$

where *C* is a constant of proportionality and *T* is the temperature inside the pipe.

By introducing the failure parameter, A_{Leak}, a general analytical description can be made that captures a wide range of the variation of the behavior that the fault can have. Hence, the failure mode F_{Leak} can be described generally by a failure parameter, $A_{Leak} \neq 0$.

From a model based diagnosis point of view this is a robust approach since all possible behaviors of a leaking pipe are solutions to the model.

5.2.2 Undescribed component behavioral modes

For components with faults that have unknown behaviors Rodon offers the possibility to implement undescribed behavioral modes. An undescribed behavioral mode captures observations that can not be explained by already described behavioral modes, the set Θ_{Fx} is represented by the striped area in Figure 15, where it also covers the set Θ_{NF}.

Undescribed behavioral modes should be used with caution since every component in which it is introduced will be able to explain inconsistent observations. This will increase the number of suspects and decrease the diagnosis performance.

It should be said that the undescribed behavioral modes sole purpose is to be used for fault isolation. The use of undescribed behavioral modes can cause trouble when generating a SDB to be used for design related analysis. Because an undescribed failure mode is consistent with all possible values, the gathered information of an SDB, that have been generated from a model containing components with undescribed behavioral modes, will be reduced. This since the value ranges of variables it affects will be infinite.

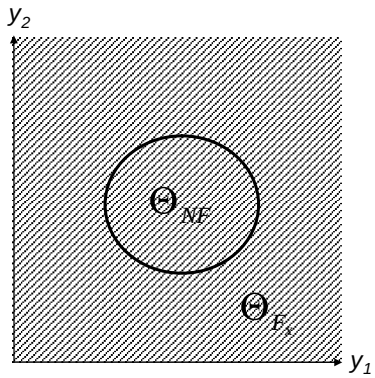


Figure 15. Illustration of the coverage of an undescribed behavioral mode F_x.

5.2.3 Consistent modeling in Rodon

It is of great importance that the model used for model based diagnosis in Rodon describes as much of the real system behavior as possible. Differences between the model and the system will reduce diagnosis performance. Reasons for inconsistencies can be many e.g. stochastic variations in signals, unmodeled component parameter uncertainty, and model simplifications.

For a system with three behavioral modes *NF*, F_1, and F_2 the differences between the model and the real system is illustrated in Figure 16, where the behavioral modes are observed through measurements y_1 and y_2. An observation $z \in \Theta_{NF}$ has been measured in the real system, as illustrated in Figure 16, and is inserted in the model. Since $z \notin \hat{\Theta}_{BM}$ for all of the implemented behavioral modes $BM \in \{NF, F_1, F_2\}$, Rodon will not be able to generate any diagnosis result. In this case, the consequence of having a model inconsistent with the real system is that Rodon can not generate any diagnosis at all. But in other cases it could lead to that the correct diagnosis is missing.

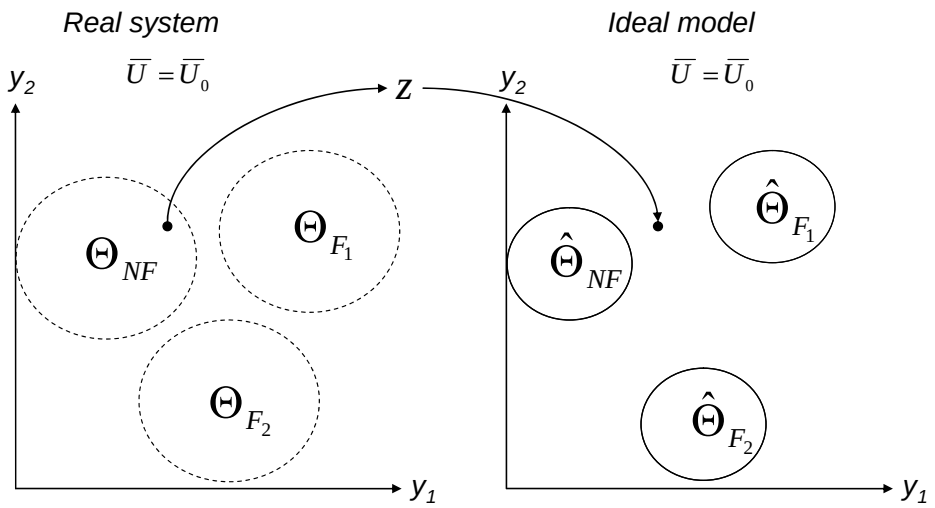


Figure 16. The differences between an inconsistent model and the real system, where the value sets of behavioral modes *NF*, F_1, and F_2 are observed through measurements y_1 and y_2 and z is an observation from the real system inserted in the model.

In Rodon interval arithmetic is used to make the models as consistent as possible to the real systems. By introducing intervals for stochastic variations in signals, component variables or imprecise knowledge of parameters, the value sets of each system behavioral mode will increase. The aim is to have a model so that all possible behaviors of the real system are solutions to the model. Hence, create a model so that $\hat{\Theta}_{BM_i} = \Theta_{BM_i}^{System}, \forall i$ is created.

5.2.4 Effects of interval use

Handling stochastic variations with interval arithmetic is however not unproblematic.

The question is how interval arithmetic is to be used most efficient in order to handle stochastic variation, without creating larger value sets for the behavioral modes then necessary.

Here follows an example of how multiplicative effects can occur when intervals are introduced in both local component parameters as well as connector variables. Consider a model of a system containing a series of resistors as showed in Figure 17, where the voltage u_1 and u_2 are measured, and $u_1 > u_2$.

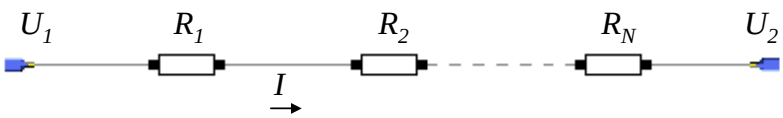


Figure 17. A model containing a series of resistors.

In the real system there is a stochastic variation for each nominal resistor resistance, r_i, and for each measured voltage, u_j, where $i = 1,\ldots, N$ and $j = 1, 2$. Since the true value of r_i, and u_j is not known their tolerances, ε_{Ri} and ε_{Uj}, are described in the model by introducing intervals, R_i and U_j, where $R_i = [r_i - \varepsilon_{Ri} \quad r_i + \varepsilon_{Ri}]$ and

$U_j = \lfloor u_j - \varepsilon_{Uj} \quad u_j + \varepsilon_{Uj} \rfloor$.

In the Rodon model, the equations of different components are linked together with connectors as described in Section 4.1.1. The interval R_i is a local parameter in each component and the total resistance of the circuit, $R_{Tot} = R_1 + \ldots + R_N$, will because of interval arithmetic be

$$R_{Tot} = \left[\sum_{i=1}^{N} r_i - \sum_{i=1}^{N} \varepsilon_{Ri} \quad \sum_{i=1}^{N} r_i + \sum_{i=1}^{N} \varepsilon_{Ri} \right] = \left[\min(R_{Tot}) \quad \max(R_{Tot}) \right]$$

Since current I is determined by the relation $\Delta U = R_{Tot} \cdot I$, where $\Delta U = U_1 - U_2$, this results in the value set

$$I = \left[\frac{\min(\Delta U)}{\max(R_{Tot})} \quad \frac{\max(\Delta U)}{\min(R_{Tot})} \right] = \left[\frac{(U_1 - \varepsilon_{U1}) - (U_2 + \varepsilon_{U2})}{\sum_{i=1}^{N} r_i + \sum_{i=1}^{N} \varepsilon_{Ri}} \quad \frac{(U_1 + \varepsilon_{U1}) - (U_2 - \varepsilon_{U2})}{\sum_{i=1}^{N} r_i - \sum_{i=1}^{N} \varepsilon_{Ri}} \right]$$

To get an estimation of the size of these multiplicative effects, assume that measurements $u_1 = 20V$ and $u_2 = 10V$ have a stochastic variation of $\varepsilon_{U_1} = \varepsilon_{U2} \pm 1V$ due to measurement noise. Furthermore assume that the system contains 4 resistors, each with resistance $r = 100\ ohm$ with a stochastic variation of $\varepsilon_{R_i} = \pm 5\ Ohm$. This leads to a current

$$I = \left[\frac{(20V - 1V) - (10V + 1V)}{4 \cdot 100\,ohm + 4 \cdot 5\,ohm} \quad \frac{(20V + 1V) - (10V - 1V)}{4 \cdot 100\,ohm - 4 \cdot 5\,ohm} \right] = [0.019\,A \quad 0.032\,A].$$

Hence, the calculated current *I* will have a value set range of $0.025 \pm 0.0065\,A$, which means a deviation of $\pm 26\,\%$. Because Rodon calculates the effects that the maximum deviation will have on the system, there is a great risk that the calculated value set is not proportional to how the real value of the system current will vary. Most likely the model will include the real system behavior but if the value sets of the behavioral modes is too large it affects the diagnosis performance. An insensitive model will have trouble separating the fault free behavior from a faulty behavior, which will lead to that *NF* will be the diagnosis even for a faulty system.

For the example system discussed in Section 5.2.3, the consequences of these multiplicative effects are illustrated in Figure 18. An observation *z* is measured in the real system and inserted in the Rodon model. Due to insensitivity of the model, *z* will be consistent with both the modeled behavioral mode sets of *NF*, and F_2.

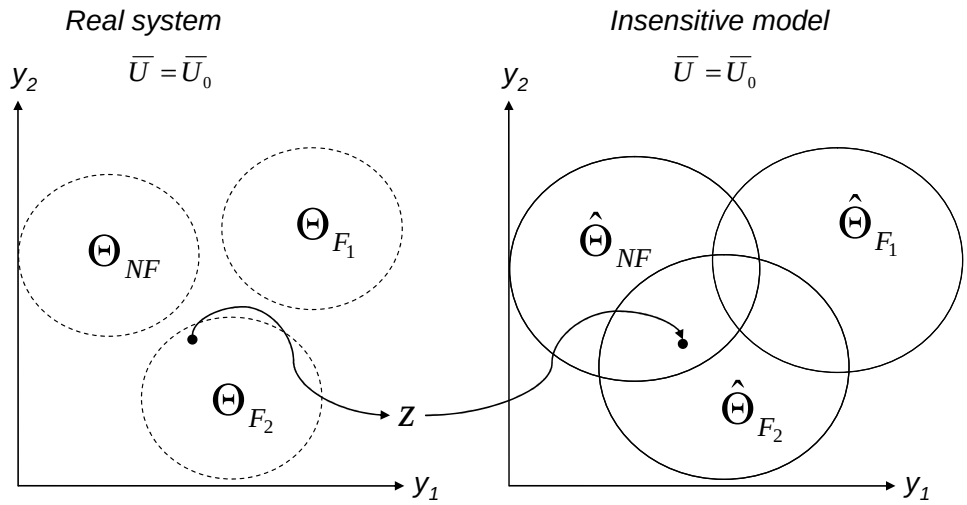


Figure 18. The differences between an insensitive model and the real system, where the value sets of behavioral modes NF, F_1, and F_2 are observed through measurements y_1 and y_2 and z is an observation from the real system inserted in the model.

5.2.5 Thresholds for measurement

An approach to reduce multiplicative effects while still handling model uncertainty could be to only introduce intervals for measured variables and by this way try to cover both measurement noise and model uncertainty.

Consider a modeled system where the measurement $y = y_0 + w$ is to be inserted in the model. y_0 is the real system value and depends on the input signal *u* as $y_0 = G_0(u)$, and *w* is measurement noise. In the model, $\hat{y}$ is calculated by the function $\hat{y} = G(u)$. If $\Delta G(u)$ is a model uncertainty, and satisfies $G_0(u) = G(u) + \Delta G(u)$, then $y - \hat{y} = \Delta G(u) + w$ is the variation that needs to be handled. Assuming that $\Delta G(u) + w = \varepsilon(u)$, then the thresholds $\min(\varepsilon(u))$ and $\max(\varepsilon(u))$ can be used to introduce an interval in the model to improve robustness so that $y \in \left[\hat{y} + \min(\varepsilon(u)) \quad \hat{y} + \max(\varepsilon(u))\right]$.

Since one cause of measurement variation could be model uncertainty, as seen above, there is also a great risk that the same uncertainty is represented in both the local parameters and in the measurements when the approach discussed in Section 5.2.4 is used. Another advantage of using the approach discussed in this section is that the risk of having the same uncertainty added more then once vanishes.

It should be said that the process of estimating suitable intervals is not trivial. Since the variation will most likely vary depending on the values of control signals, a great deal of sampled measurement data will have to be gathered for many different control signal combinations. The possibility of generating measurement data for faulty behaviors will also be limited. Because the number of faulty system behavioral modes increases with the size of the system, it might not be financially sound to estimate thresholds for each and every set $\Theta_{BM}^{u=u_0}$. Especially not since the wide range of variation a faulty behavior can have, see Section 5.2.1, increases the complexity of the problem even more. The most realistic approach will probably be to estimate the thresholds $\min(\varepsilon(u))$ and $\max(\varepsilon(u))$ for the nominal fault free behavior, then use the same thresholds for all behavioral modes, and hope that the diagnosis performance will be sufficient enough.

5.3 Simulation

Rodon is developed primary to simulate stationary behavior of systems. However it is possible to perform dynamic simulations with Rodon. When it comes to analyzing dynamic systems, an evaluation need to be made in order to chose a suitable simulation approach. For some dynamic systems, it might be sufficient with a model based on a static approximation of the dynamic behavior, but in many cases the dynamics will be necessary to include in the model. The two different approaches above will be discussed, in Sections 5.3.1 and 5.3.2, with the example described below. Consider a system $sys \in \{NF, F\}$. The behavior of the systems is described by the dynamic equations,

$$y + \dot{y} = u - f$$

$$u = a$$

$$f = \begin{cases} 0 & 0 \le t \le \tau \\ b & \tau < t \end{cases}$$

where u is a constant input signal, y the output signal, and f is a signal describing the behavior of the fault F. When time $t < \tau$ then $sys = NF$. At time $t = \tau$ fault F occurs and starts to affect the system. The system behavior is roughly outlined in Figure 19, where $u = 100$, $f = 30$, and $\tau = 0.3$.

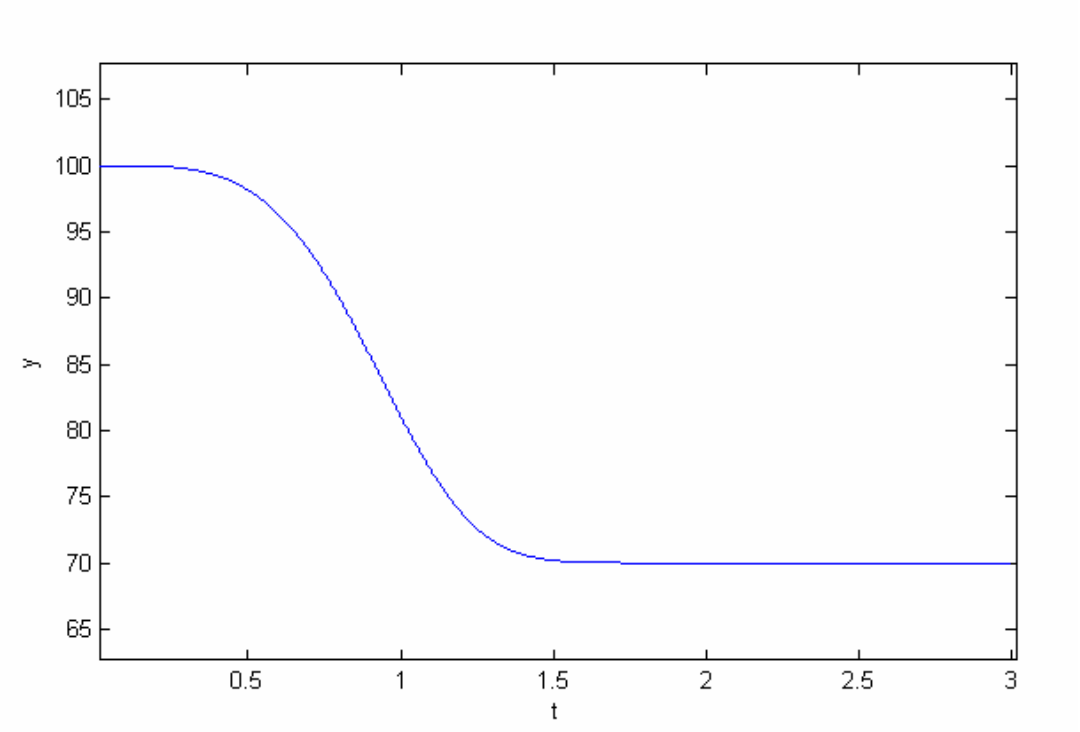


Figure 19. A plot of how y varies with time influenced by fault f.

5.3.1 Dynamic

In order to make a dynamic simulation in Rodon a discretization of time into intervals of size *dt* is needed. This is done in Rodon by dividing the simulation into different frames, see Section 4.3. After this, the user needs to implement how derivatives of dynamic variables should be approximated and updated between the frames. Here follows an example of how the dynamic behavior of system sys, described in Section 5.3, could be implemented with a forward Euler approximation in Rodon.

$$\hat{y} + \dot{\hat{y}} = u - f$$

$$\hat{y}Next = \hat{y} + dt \cdot \dot{\hat{y}}$$

The smaller intervals *dt* used during simulation, the better approximation of the real system behavior will be achieved. Rodon does not have any plotting features so calculated values of $\hat{y}, \dot{\hat{y}}$ from a simulation will be gathered in tables. For model based diagnosis purposes this might not be a limitation since the only information requested from a simulation is suspected candidates. However, in design phase the lack of plotting feature will make it hard to analyze the generated simulation data.

This might seem trivial but if simulation data is to be exported to other tools, with plotting features, the use of interval arithmetic in Rodon might cause problems since most plotting functions uses vectors of single numeric values as inputs, not intervals.

The use of interval arithmetic in Rodon will also cause problems when it comes to verifying the model quality. Assume that a model for system *sys = NF* is to be validated. Measurements have been collected from the real system during a time window *t = 1,…, N*. A commonly used method for this is to calculate the mean of the *prediction error*, $y - \hat{y}$, where y is the real system value and $\hat{y}$ the calculated model value.

The validity of the model quality when *sys = NF* is estimated by the function *V*:

$$V = \frac{1}{N}\sum_{t=1}^{N} \| y[t] - \hat{y}[t] \|$$

When validating a Rodon model, the question that needs to be answer is what interpretation of this function that can be made when $\hat{y}$ is an interval.

As said before these problems discussed here might seem trivial. Still they will have to be dealt with.

5.3.2 Static

A static approximation of the example described in Section 5.3 is achieved by setting $\dot{y} = 0$ in the model. This means that the static simulation result that Rodon generates can be interpreted as the value of *y* when all system transients have disappeared. To describe how the fault *F* is going to affect the system the two behavioral modes *NF* and *F* has to be simulated separately. For the fault free case the system behavior is described by the following equations:

$$y = u$$

For fault mode *F*, the system behavior is described by equations:

$$y = u - f$$

If this approach is to be useful the system can not be controlled more rapid then the systems own time constant because this results in the system not reaching a stationary behavior. It is up to the user to determine if a static approach will be sufficient or not. For many applications in Rodon this is a necessary approach. E.g. the generation of SDB, described in Section 4.4.1, is generated through static simulations.

5.3.3 Validation of a model

The quality of the model will affect the diagnosis performance in Rodon and therefore it is of importance to validate the model. Here are some suggestions on methods to follow and criteria to check when verifying a model.

- Simulate the nominal fault free case. Make sure that no variables in the model have undefined values. This should hold for all control signal combinations.

- Use the auto simulation feature to simulate all single faults that can occur in the model for all control signal combinations. The percentage of simulations that has resulted in conflicts should be very low. Each case that causes a conflict should be carefully investigated.

- In the case that a simulation has been aborted the feature *constraint net analyzer* is strongly recommended for debugging, where the calculation processes can be viewed and loops detected.

5.4 Applications

Rodon offers flexible simulation opportunities that suite different analysis approaches.

During development it is possible to analyze what impact a certain component behavioral mode *bm* and control signal combination *u* will have on the system observations *y* and failure codes *fc*, see Figure 20. In Section 3.1 this was referred to as failure analysis but the same approach could also be used for generation of decision trees to aid the technicians in maintenance, as described in Section 4.4.3.

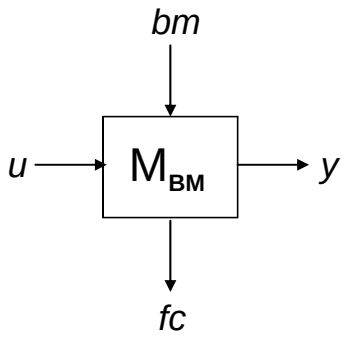


Figure 20. A simulation approach in Rodon that is useful in many design phase applications. Where control signals *u*, behavioral modes *bm*, are inputs and failure codes *fc*, and calculated observations *y*, are outputs.

Failure codes that have been set and observations that have been measured in the real system can be used together with the model to get all diagnosis, see Figure 21. This can be useful in maintenance for fault isolation purposes, but the same approach can also be used to generate decision structures for implemented tests.

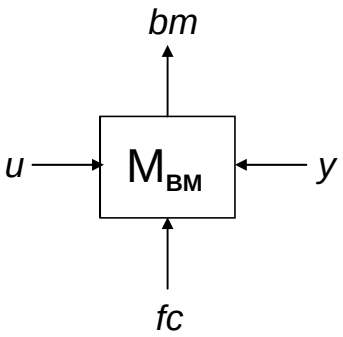


Figure 21. A simulation approach in Rodon that is used for fault isolation and generation of decision structures for implemented tests. Where control signals *u*, failure codes *fc*, and observations *y*, are as inputs and behavioral modes *bm* are outputs.

These aspects make Rodon useful during both development and operations.

5.4.1 Limitations

As discussed in Section 5.3, Rodon is developed primary to analyze stationary behavior of systems. Dynamic simulations can be made and used for fault isolation on dynamic systems, but most of the applications in Rodon are designed for use on static models.

Another limitation is that some Rodon applications require discrete control signals. Consider auto simulation described in Section 4.4.1, which is a significant application in Rodon. It generates state space databases for static models, by simulating each combination of control signal *u*, and behavioral modes *bm*, as illustrated in Figure 20. An analog varying control signal will in auto simulation be interpreted by Rodon as a discrete control signal with an interval including the whole value range of the analog signal. A solution to this could be to map the value range of each analog control signal into discrete working points.

5.5 Summary

In this chapter a general overview of how it is to work in Rodon have been given. Benefits that can be gained when using different Rodon applications have been described, but also problems and limitations that can occur depending on modeling and simulation purposes.

To conclude, the criterions for a model to be useful in all Rodon applications are:

- The model needs to be static.
- The model needs to have discrete control signals.
- The model needs to have well defined system behavioral modes.

For fault isolation on physical systems the only limitation will be how well the real system behavior is described by the model, as discussed in sections 5.2.3 - 5.2.5.

In the next chapter a method is given with the purpose to decide which tests and sensors that should be implemented to achieve fault isolation and functional monitoring goals.

Chapter 6

A method to compute all tests from the SDB

In this chapter a general method will be presented with the purpose to generate all tests that can be implemented in the on board diagnosis system to achieve FDI. These tests are generated from the information of the system state space. After this, a possible implementation of the method in Rodon will be discussed together with limitations. The chapter will conclude with how Rodon can be used together with a modified version of the method to generate all tests.

6.1 Overview of the method

The method can be used for a system with an arbitrary number of sensors and *m* system behavioral modes to generate all possible tests. The first part of the method is to compute the system state space. This is performed by determining the set $\boldsymbol{Z}_{model} = \Theta_{BM_1} \cup \Theta_{BM_2} \cup ... \cup \Theta_{BM_m}$, where $\Theta_{BM_i} = \{(u, y) \mid M^{BM_i}(u, y) = 0\}$. This is put into words equivalent to determining how the model responds to different system behavioral modes. This information is further processed in the second part of the method that generates and gathers all possible tests in a decision structure, which can be used for diagnostic system design.

6.2 Example of how to generate $\boldsymbol{Z}_{model}$

How to generate $\boldsymbol{Z}_{model}$ will be described through an example of an arbitrary system, *sys*, with three system behavioral modes, *NF*, F_1, and F_2, and two sensors y_1 and y_2. For simplicity it is assumed that the system only can be controlled with $u = u_0$ and the corresponding system behavioral mode sets can be seen in Figure 22. The SDB contains the set $\boldsymbol{Z}_{model}$, which in this case is determined as

$$\Theta_{NF} \cup \Theta_{F_1} \cup \Theta_{F_2} =$$

$$\{y \mid M^{NF}(u_0, y) = 0\} \cup \{y \mid M^{F_1}(u_0, y) = 0\} \cup \{y \mid M^{F_2}(u_0, y) = 0\}$$

In the example it is assumed that the model is a perfect description of the system, hence $\Theta_{BM_i} = \Theta_{BM_i}^{sys}, \forall i$.

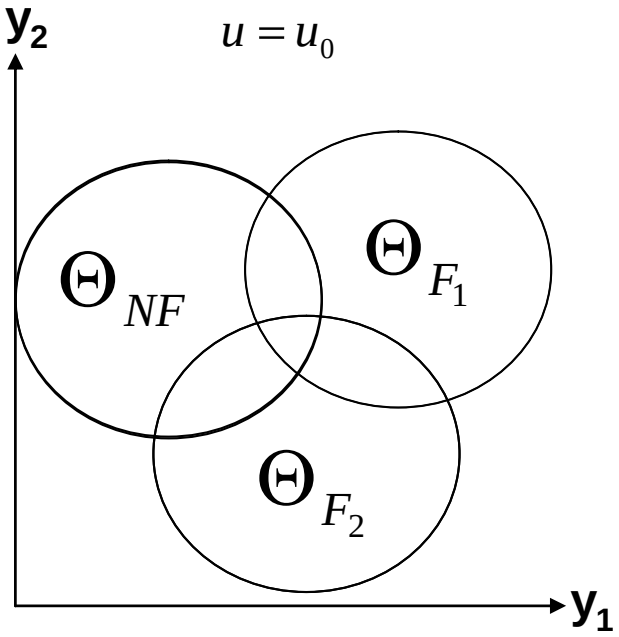


Figure 22. View of the system state space, $Z_{model} = \Theta_{NF} \cup \Theta_{F_1} \cup \Theta_{F_2}$.

The process of determining the system state space will be formalized in next section but the basic idea is to calculate the corresponding measurements *y* of the sensors, for each control signal $u = u_0$ and each system behavioral mode *BM*, with the help of the model $M^{BM}(u, y) = 0$.

6.3 System state space generation algorithm

This section contains the system state space generation algorithm, which can be used to generate information of the system state space.

6.3.1 Definition

Given a system with the corresponding model $M^{BM_k}(u, y) = 0$,
where $BM_k \in \{NF, F_1, ..., F_m\}$ and $y = (y_1, ..., y_i, ..., y_p)$ are calculated expected measurements from sensors. Control signals are u_k, $k = 1,..,i,..,n$. Each control signal u_i can be chosen according to all possible choices included in U_i,
$u_1 \in U_1, u_2 \in U_2, ..., u_i \in U_i, ..., u_n \in U_n$. All possible control signal combinations can be defined as $U = U_1 \times U_2 \times ... \times U_i \times ... \times U_n$. The currently iterated control signal is denoted u_0, and $u_0 \in U$.

6.3.2 Algorithm part

The input is the model equations for all system behavioral modes, and all control signals, of interest. The output is information of the system state space in the form of a table, called SSP (System State Space).

$$Input: U, M^{BM_k}(z) = 0, \forall BM_k \in \{NF, F_1, ..., F_m\}$$
$$Output: Table\ SSP\ (System\ state\ space)$$
$$Initiation: j = 1;$$
1. *for all* $u_0 \in U$
2. *for all* $BM_k \in \{NF, F_1, ..., F_m\}$
3. *find* y *consistent with* $M^{BM_k}(u_0, y) = 0;$
4. $SSP_j := (BM_k, u_0, y_1, ..., y_i, ..., y_p);$
5. $j = j++;$

end

end

Initiation is performed by setting the SSP row index $j = 1$. The algorithm is iterated for each control signal combination u_0 in U (on line 1) and each system behavioral mode (on line 2). The model is used every iteration to find consistent expected measurements (on line 3). The current iterated system behavioral mode and control signal combination is together with calculated measurements inserted in the *j*:th row of the SSP (on line 4). After this, *j* is incremented (on line 5). The resulting SSP table will consist of $(m+1) \cdot number\ of\ control\ signals\ in\ U$ rows and $(p+2)$ columns.

6.4 Example of how to generate all tests

When the SSP has been computed, it can be used for test generation. This is because the SSP contains all the information of the system state space *Z* and states how all sensors will be affected by every possible control signal and system behavioral mode combination. The test generation is achieved by dividing the system state space into all possible subsets and creating a test for each and every subset. An illustration of how the example in Section 6.2 is divided into subsets, $\Theta_i^C, i = 1,...,7$, can be found in Figure 23.

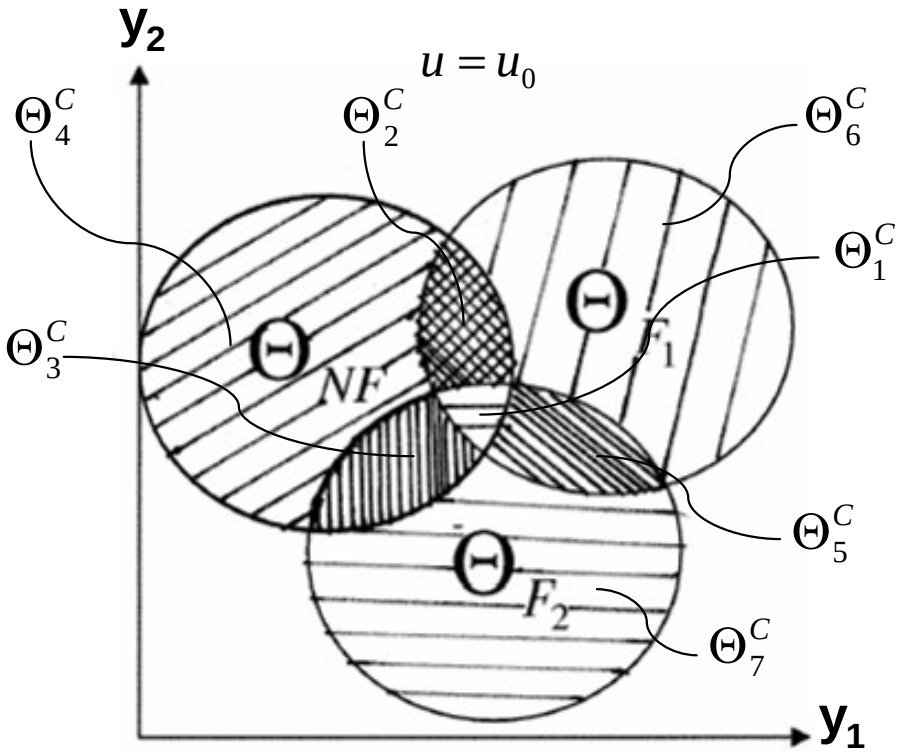


Figure 23. Example of how the original set is divided into subsets.

The resulting tests of the example are listed in the order of how they are generated for the example above and can be seen in Table 5.

Table 5. The tests that can be generated for the example.

$Initially: Z = \Theta_{NF} \cup \Theta_{F_1} \cup \Theta_{F_2}$	Test	Rejection region	Decision
$\Theta_1^C = Z \cap \Theta_{NF} \cap \Theta_{F_1} \cap \Theta_{F_2}$	T_1	$z \in \Theta_{T_1}^C = \Theta_1^C \rightarrow$	$sys \in S^1 = \{NF, F_1, F_2\}$
$Z = Z \setminus \Theta_1^C$			
$\Theta_2^C = Z \cap \Theta_{NF} \cap \Theta_{F_1}$	T_2	$z \in \Theta_{T_2}^C = \Theta_2^C \rightarrow$	$sys \in S^1 = \{NF, F_1\}$
$Z = Z \setminus \Theta_2^C$			
$\Theta_3^C = Z \cap \Theta_{NF} \cap \Theta_{F_2}$	T_3	$z \in \Theta_{T_3}^C = \Theta_3^C \rightarrow$	$sys \in S^1 = \{NF, F_2\}$
$Z = Z \setminus \Theta_3^C$			
$\Theta_4^C = Z \cap \Theta_{NF}$	T_4	$z \in \Theta_{T_4}^C = \Theta_4^C \rightarrow$	$sys \in S^1 = \{NF\}$
$Z = Z \setminus \Theta_4^C$			
$\Theta_5^C = Z \cap \Theta_{F_1} \cap \Theta_{F_2}$	T_5	$z \in \Theta_{T_5}^C = \Theta_5^C \rightarrow$	$sys \in S^1 = \{F_1, F_2\}$
$Z = Z \setminus \Theta_5^C$			
$\Theta_6^C = Z \cap \Theta_{F_1}$	T_6	$z \in \Theta_{T_6}^C = \Theta_6^C \rightarrow$	$sys \in S^1 = \{F_1\}$
$Z = Z \setminus \Theta_6^C$			
$\Theta_7^C = Z \cap \Theta_{F_2}$	T_7	$z \in \Theta_{T_7}^C = \Theta_7^C \rightarrow$	$sys \in S^1 = \{F_2\}$
$Z = Z \setminus \Theta_7^C$			

These tests can be gathered in a decision structure. The process of dividing the original set into subsets and creating tests for each and every subset is formalized in next section but the basic idea can be described as follows.

Outer loop: *For every control signal combination* u_0 *of U*

Calculate the subset, Z_{sub}*, of the system state space, Z, defined by the current iterated control signal combination as* $Z_{sub} = \{z \mid z \in Z, u = u_0\}$

Inner loop: *Divide* Z_{sub} *into all possible subsets and create a test for each subset.*

Include each test in the decision structure

This means for the example above, that the outer loop only will be iterated once since there is only one control signal combination. It should be noted that Z = Z_{sub} in this case. The inner loop will be iterated seven times, since Z_{sub} can be partitioned into seven non-empty subsets.

6.5 Test generation algorithm

This section contains the test generation algorithm, which can be used to divide the system state space into subsets and create a test for each and every subset. The output is all generated tests and they are gathered in a decision structure.

6.5.1 Definition

The same notation as in Section 6.3.1 will be used here.

6.5.2 Algorithm part

The input is the SSP and all system behavioral modes and control signals of interest. The output is a decision structure table, which is on the form that can be seen in Section 2.2.2. Current iterated control signal is denoted u_0 and all control signals *U*. A binary number $x = (x_1\, x_2 .. x_l .. x_m)$ is used to keep track of which subset of Z_{sub} that is currently searched. A new set is necessary to define, which for $u = u_0$ and the system behavioral mode $BM_k \in \{NF, F_1, ..., F_m\}$ is:

$$\Theta_{BM_k}^{u=u_0} = \{y \mid M^{BM_k}(u_0, y) = 0\}$$

$Input: U, SSP, \{NF, F_1, ..., F_m\}$
$Output: Table, decision\ structure$
$Initiation: j = 1;$
1. *for all* $u_0 \in U$
2. *Use the SSP to decide* $Z_{sub} = \bigcup_{BM_k \in \{NF, F_1, ..., F_m\}} \Theta_{BM_k}^{u=u_0};$
3. *set* $x = (11.....1);$
4. *while*$(x > 0)$
5. *Use the SSP to decide* $\Theta_{current}^{C} = \bigcap_{k: x_k = 1} \Theta_{BM_k}^{u=u_0} \cap Z_{sub};$
6. *if* $(\Theta_{current}^{C} \neq \varnothing)$
7. *save the rejection region of the test* $: \Theta_{T_j}^{C} = \Theta_{current}^{C};$
8. *save the decision of the test* $: S^1 = \{BM_k \mid x_k = 1\};$
9. *decision structure*$_j := (\Theta_{T_j}^{C}, S^1);$
10. *Update* $Z_{sub} := Z_{sub} \setminus \Theta_{current}^{C};$
11. $j++;$
end
12. $X--;$
end
end
$j_{end} = j;$

Initiation is performed by setting the decision structure row index $j = 1$. The outer loop of the algorithm is iterated for each control signal combination u_0 in U (on line 1). In the outer loop, the SSP is used to decide Z_{sub}, which is the subset of the system state space for the current iterated control signal (on line 2). Furthermore, the binary number x is initiated each outer loop iteration (on line 3). If x is expressed binary then it can be said to represent which behavioral mode subset that is being explored in the inner loop. For the example in Section 6.4, x would be initiated as x=<1 1 1>. The binary number x=<1 1 1> represents that the intersection of all behavioral modes will be searched during the first iteration of the inner loop, $\Theta_{NF} \cap \Theta_{F_1} \cap \Theta_{F_2}$. After the inner loop has finished, x will be decremented to x=<1 1 0>, which means that the intersection of behavioral modes $\Theta_{NF} \cap \Theta_{F_1}$ instead will be searched during the second iteration. The inner loop will continue with searching the current behavioral mode intersection and decrementing x, until x = 0.

The inner loop starts by calculation of the currently searched region, as the intersection between the system behavioral mode sets, indicated by *x*, and the set Z_{sub} (on line 5). If the currently searched region is non-empty (on line 6) then it is saved as the rejection region of a test (on line 7). If the system is behaving according to one of the behavioral modes that are included in the searched subset this can cause the created test to react. Implicitly this means that *x* indicates which behavioral modes that can cause the test to react, which if expressed binary is equivalent to the row information of a decision structure, and the decision then taken. This decision is saved (on line 8) and inserted together with the rejection region of the test on the *j*:th row of the decision structure (on line 9). The set Z_{sub} is updated by removing the rejection region of the created test (on line 10) and the row index *j* is incremented (on line 11). The final step of the inner loop is to decrement *x* (on line 12). The inner loop starts over again if the condition $x > 0$ is fulfilled (on line 4), if not, then the outer loop is reiterated provided that there are control signal combinations in *U* left to iterate.

The resulting decision structure table will consist of j_{end} rows and $(m+2)$ columns. The value of j_{end} can not be predetermined, and therefore the number of rows will only be apparent after the algorithm has finished.

6.6 Implementation in Rodon and limitations

There are some limitations of how Rodon can be used in the method. The system state space algorithm can be performed automatically and the result will be an SDB table (see Section 4.4.1), but this is only an approximation of the SSP. This is because the SDB only contains value intervals describing how each and every variable can vary independent of any other variable. For further clarification, assume a system, *sys*, which has two system behavioral modes, *NF* and *F*, and two sensors y_1 and y_2. The system can only be controlled as $u = u_0$. A model of the system has been developed, and $\boldsymbol{Z}_{model}$ is the set $\Theta_{NF} \cup \Theta_F = \Theta_{NF}^{u=u_0} \cup \Theta_F^{u=u_0}$.

Assume that one model equation for *NF* is $y_1^2 + y_2^2 \le r^2, r = 2$ and the SDB states for *NF* that $y_1 \in [-2\ 2]$ and $y_2 \in [-2\ 2]$, then these value ranges define a region that contains values that are inconsistent with the model equation. Pick e.g. the point $(y_1, y_2) = (2,2)$ for which $y_1^2 + y_2^2 = 2^2 + 2^2 = 8 > 4$. These inconsistent points would not be included in the region if the SDB described the dependencies between the variables as the SSP will do.

The test generation algorithm can not be achieved in Rodon, but the SDB can be used as input in the algorithm since it is an approximation of the SSP. The SDB contains regions

$$\hat{\Theta}_{BM_k}^{u=u_0} = \{y_i \in \Delta y_i \mid 1 \le i \le p\}, \forall k, \forall u_0 \in U, where\ \Delta y_i = [\min(y_i)\ \max(y_i)]$$

The corresponding theoretical SDB for *sys* can be seen in Table 6. Figure 24 show the system behavioral modes, the resulting intervals, and the corresponding SDB set (dashed rectangles) for *sys*.

Table 6. The theoretical SDB for the example that is used in the test generation algorithm.

System behavioral modes	Control signal u	y_1	y_2
Θ_{NF}	u_0	$\Delta y_{1,NF} = \left[\min_{y_1}\left(\Theta_{NF}^{u=u_0}\right) \max_{y_1}\left(\Theta_{NF}^{u=u_0}\right) \right]$	$\Delta y_{2,NF} = \left[\min_{y_2}\left(\Theta_{NF}^{u=u_0}\right) \max_{y_2}\left(\Theta_{NF}^{u=u_0}\right) \right]$
Θ_{F}	u_0	$\Delta y_{1,F} = \left[\min_{y_1}\left(\Theta_{F}^{u=u_0}\right) \max_{y_1}\left(\Theta_{F}^{u=u_0}\right) \right]$	$\Delta y_{2,F} = \left[\min_{y_2}\left(\Theta_{F}^{u=u_0}\right) \max_{y_2}\left(\Theta_{F}^{u=u_0}\right) \right]$

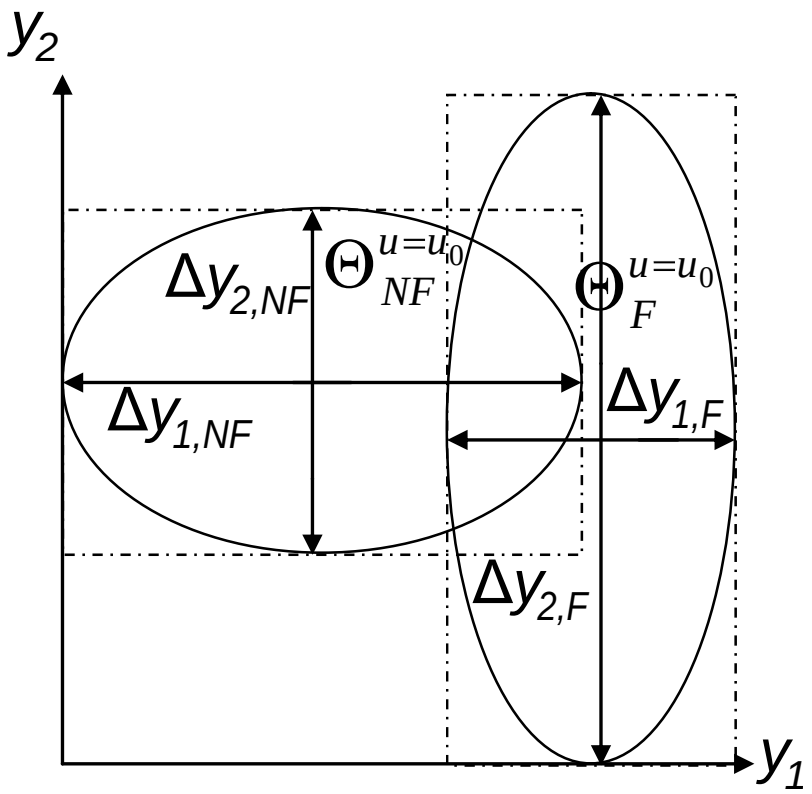


Figure 24. The system behavioral modes of the example and the resulting SDB intervals, which indicate the SDB regions (dashed boxes) that are used in the test generation algorithm.

The intervals of the figure define the SDB regions

$$\hat{\Theta}_{NF}^{u=u_0} = \{y \mid y_1 \in \Delta y_{1,NF},\, y_2 \in \Delta y_{2,NF}\} \text{ and } \hat{\Theta}_{F}^{u=u_0} = \{y \mid y_1 \in \Delta y_{1,F},\, y_2 \in \Delta y_{2,F}\}$$

If the information of the SDB is used in the test generation algorithm then the result will be the rejection regions shown in Figure 25, and the following tests, listed in the order of how they are created with the algorithm:

Test	Rejection region	Decision
T_1	$z \in \Theta_{T_1}^C = \hat{\Theta}_{NF}^{u=u_0} \cap \hat{\Theta}_{F}^{u=u_0} \rightarrow$	$sys \in \{NF, F\}$
T_2	$z \in \Theta_{T_2}^C = \hat{\Theta}_{NF}^{u=u_0} \setminus \hat{\Theta}_{F}^{u=u_0} \rightarrow$	$sys = NF$
T_3	$z \in \Theta_{T_3}^C = \hat{\Theta}_{F}^{u=u_0} \setminus \hat{\Theta}_{NF}^{u=u_0} \rightarrow$	$sys = F$

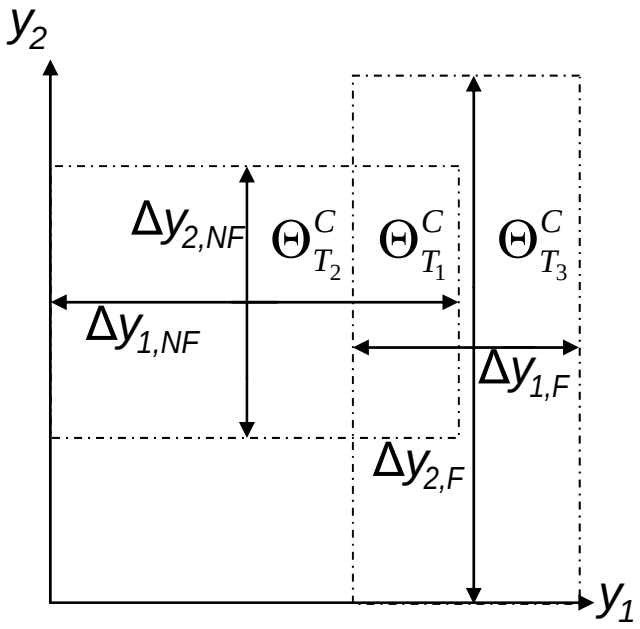


Figure 25. The three resulting rejection regions when the test generation algorithm is used.

For the example above the use of the test generation algorithm has been performed manually. However, for a bigger SDB this manual implementation would be very hard to achieve and therefore it would be necessary to automate the algorithm. The test generation algorithm has not been automated in this thesis, since this was not included in the scope, and because of time shortage. Instead, a variation of the algorithm was implemented and will be presented in next section.

6.7 Modification of the test generation algorithm

The SDB in Table 6 will be used to illustrate the steps of the modified test generation algorithm. Each sensor column $y_i, \forall i,$ contains a listing of intervals $\Delta y_{i_1},...,\Delta y_{i_l},....,\Delta y_{i_N}$, where N is the *number of control signals in* U times the *number of system behavioral modes*. Hence, each interval corresponds to one certain system behavioral mode and control signal combination.

These intervals can intersect each other. The intervals calculated from the same control signal combination are for each column divided until no such intersections exist, which results in intervals $\Delta y'_{i_1},...,\Delta' y_{i_l},....,\Delta' y_{i_K}$, where $K \geq N$. This means that the same steps are followed as in the test generation algorithm with the difference being that only one sensor dimension at a time, is processed.

For the example depicted in Figure 24, where $\Delta y_{2,NF}$ intersects $\Delta y_{2,F}$ and $\Delta y_{1,NF}$ intersects $\Delta y_{1,F}$, this would result in the subintervals shown in Figure 26. A test is created for each generated interval of each column, which results in six tests for the example.

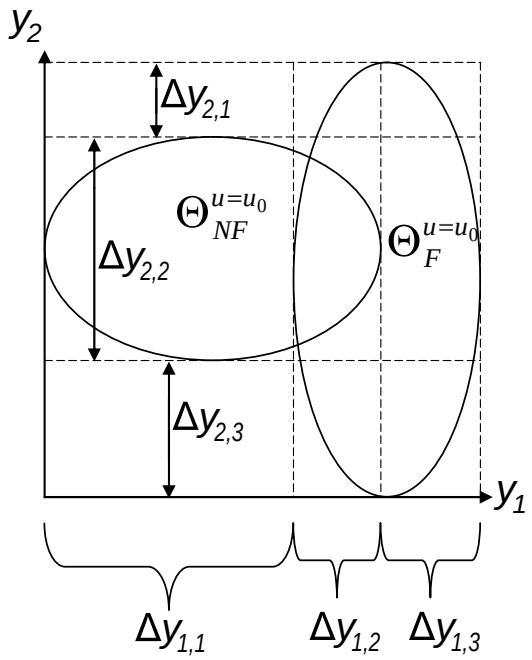


Figure 26. Resulting subintervals when the modified test generation algorithm is used.

The following tests can be created in the example:

$$T_1 : z \in \Theta_{T_1}^C = \left\{ z \mid u = u_0, y_1 \in \Delta y_{1,1} \right\} \quad \rightarrow sys = S^1 = \{NF\}$$
$$T_2 : z \in \Theta_{T_2}^C = \left\{ z \mid u = u_0, y_1 \in \Delta y_{1,2} \right\} \quad \rightarrow sys \in S^1 = \{NF, F\}$$
$$T_3 : z \in \Theta_{T_3}^C = \left\{ z \mid u = u_0, y_1 \in \Delta y_{1,3} \right\} \quad \rightarrow sys = S^1 = \{F\}$$
$$T_4 : z \in \Theta_{T_4}^C = \left\{ z \mid u = u_0, y_1 \in \Delta y_{2,1} \right\} \quad \rightarrow sys = S^1 = \{F\}$$
$$T_5 : z \in \Theta_{T_5}^C = \left\{ z \mid u = u_0, y_1 \in \Delta y_{2,2} \right\} \quad \rightarrow sys \in S^1 = \{NF, F\}$$
$$T_6 : z \in \Theta_{T_6}^C = \left\{ z \mid u = u_0, y_1 \in \Delta y_{2,3} \right\} \quad \rightarrow sys = S^1 = \{F\}$$

For all tests, except T_5, the decision when a test does not react will be $sys \in S^0 = \{NF, F\}$. However, if T_5 does not react the decision will be $sys = S^0 = \{F\}$, since $\Theta_{NF}^{u=u_0}$ is a subset of $\Theta_{T_5}^C$.

The resulting decision structure for the tests above can be seen in Table 7.

Table 7. Decision structure for the tests generated through the modified method.

Test	***NF***	***F***
T_1	X	0
T_2	X	X
T_3	0	X
T_4	0	X
T_5	1	X
T_6	0	X

6.8 Summary

This chapter presented a general method to generate the system state space, which could be used to generate all tests. Rodon could be used to generate the SDB which constitutes an approximation of the system state space. This SDB could be used with the test generation algorithm manually but this approach turned out to be very cumbersome if the SDB contained much information. A recommendation is to automate the test generation algorithm if it is to be of practical use. The test generation algorithm was modified to better suite a manual approach. An example was presented on which both the original and modified test generation algorithm was used. The achieved diagnostic performance was in this case the same for the two approaches. This thesis leaves no guarantee that the achieved diagnostic performance always will be the same.

The modified test generation algorithm will be used in Section 7.1 and 7.2. The next chapter will analyze different parts of the vision presented in Chapter 3, and determine whether or not they are fulfilled.

Chapter 7

Vision analysis

This chapter will try to determine if Rodon fulfills the desired achievements, presented in Section 3.2. It will start with the use of Rodon during developments: for failure analysis, guidance of sensor placements, design and evaluation of tests for an on board diagnosis system, and generation of decision structures of designed tests. After this follows a section about the use of Rodon for fault isolation during operations, before the chapter is concluded with a summary.

7.1 Failure analysis

The possible uses of failure analysis are many. As said in Section 3.1, it is important to analyze the behavior of a system in order to determine which faults that can cause critical system failures. During early development of a system this is done to detect design flaws, but it is also used for developed systems to design diagnostic tests for functional monitoring and fault isolation, and one method for this is described in Chapter 6.

Complex systems demand systematical failure analysis and two common methods to structurize the work are FTA (Fault Tree Analysis) and FMEA (Failure Mode Effect Analysis). FTA is briefly described in Section 4.4.4 and will not be analyzed in this thesis. It is mentioned here since Rodon features an FTA application, but this application uses another type of model compared to the one described in this thesis.

FMEA on a system is done by investigating each component that the system consists of and list which faulty behaviors they can have. When that is done each component fault behavior is analyzed in order to determine the effects it will have on the whole system. To be able to do this, detailed knowledge of the system and all of it components is needed. The result from the FMEA is then summarized in a chart.

By using the already existing knowledge on component level of which faults that can occur and how they will behave to build a Rodon model, the effects each fault will have on the system can be automatically generated through auto simulation and the result gathered in an SDB as described in Section 4.4.1. Since the SDB contains the knowledge of which behavioral modes that will deviate from *NF*, it can then be used as a decision basis for FMEA. This will probably ease the work of FMEA, because the analysis of how a fault affects the system is performed manually today. It is also likely that the quality of the FMEA will increase since the risk of missing fault effects will decrease.

To conclude, Rodon is very useful for failure analysis and thereby fulfills this requirement of the vision.

7.2 Guidance of sensor placements for FDI

This section will discuss how Rodon can be used to decide optimal sensor placement in a system. The sensor placement choice is intertwined with achieved diagnosis performance, since the choice determines which tests that can be created in the on board diagnosis system. Therefore, optimality is here defined as given a diagnosis performance specification to minimize the number of sensors.

If the Rodon model describes the system behavior well, then it can be used to give an indication of how sensors should be placed to achieve a certain diagnosis performance. Possible sensor placements are marked with the property mark *observation* in the Rodon model and can of course be an arbitrary choice. The process of deciding optimal sensor placement starts by generating the SDB of the model including all potential sensors. This SDB contains an approximation of what every sensor in theory will measure in each system behavioral mode, and because of this it can be analyzed to create a decision structure that contains all possible tests for every sensor, as described in Section 6.7. This decision structure can be analyzed to decide if a sensor choice exists that accomplishes the desired diagnosis performance.

Assume a system, *sys*, in which we want to find the optimal sensor placement, emanating from some initial sensor placement points, in order to achieve a desired diagnosis performance. The system contains the components and component behavioral modes listed in Table 8.

Table 8. Component and component behavioral modes in the modeled system and the description of their model equations.

Component	**Behavioral mode**	**Description**
Battery	No fault (*nf*)	$i_{in} + i_{out} = 0$
Wire	No fault (*nf*)	$u_{in} = u_{out}$, $i_{in} + i_{out} = 0$
Wire	Short to ground (*short2gnd*)	$u_{in} = u_{out} = 0$
Wire	Short to battery (*short2batt*)	$u_{in} = u_{out} = u_{battery} = 12V$
Switch	No fault (*nf*)	Desired position = Actual position
Switch	Disconnected (*disc*)	Actual position = switch open
Switch	Pin short (*pin_short)*	Actual position = switch closed
Bulb	No fault (*nf*)	$u_{in} = u_{out}$, $i_{in} + i_{out} = 0$
Bulb	Disconnected (*disc*)	$i_{in} = i_{out} = 0$
Ground	No fault (*nf*)	Kirchhoff: $i_{in} + i_{ground} = 0$ $u_{ground} = 0$
Ground	Disconnected (*disc*)	$i_{in} = i_{ground} = 0$

The model of the system can be seen in Figure 27, in which the circles marks initial current and voltage sensor points. The desired diagnosis performance of the on board diagnosis system is to detect when the switch malfunctions and deviates from *nf*.

$$\hat{y} = \begin{pmatrix} wire1.p2.i \\ wire1.p2.u \\ wire2.p1.i \\ wire2.p1.u \\ wire3.p2.i \\ wire3.p2.u \end{pmatrix}$$

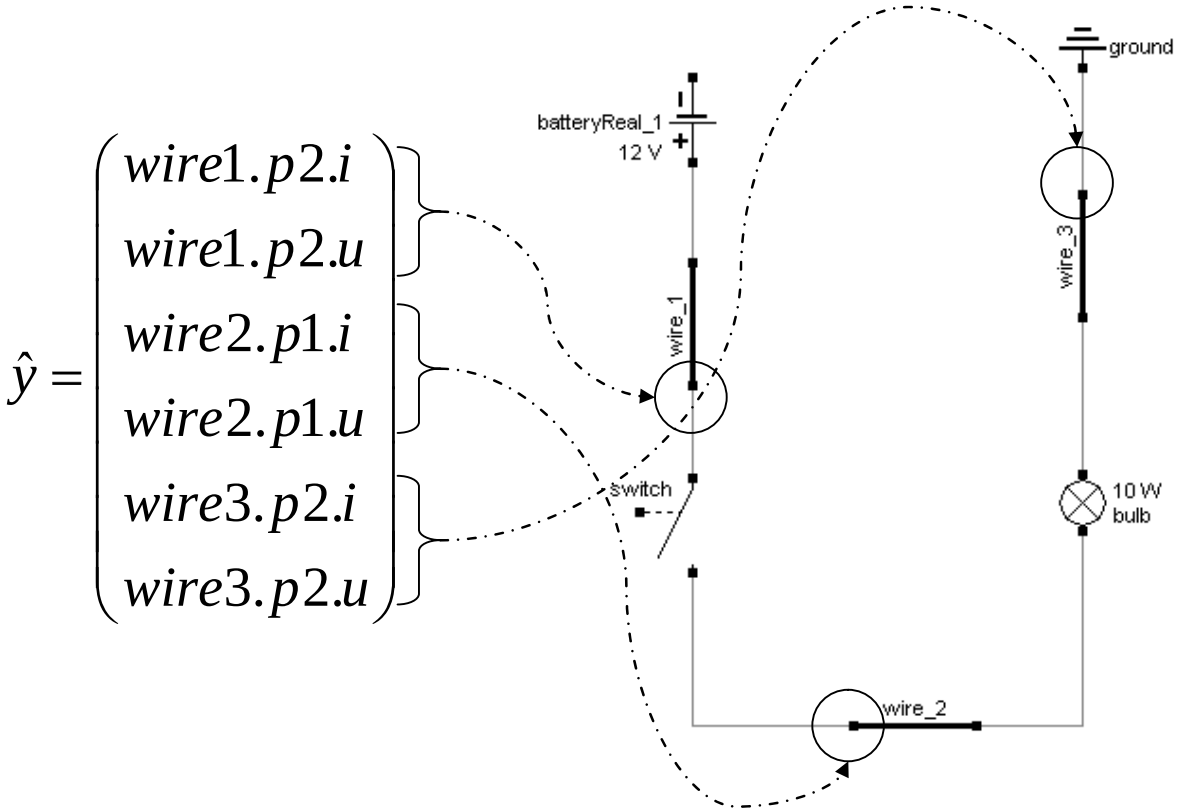


Figure 27. The model of the system and the defined possible measurement points, which are marked by circles.

The system behavioral modes are defined in Table 9.

Table 9. Listing of how the system behavioral modes are defined in the example.

NF	All components behave according to *nf*
F_1	Bulb disconnected, [1]
F_2	Ground disconnected, [1]
F_3	Switch disconnected, [1]
F_4	Switch pin short, [1]
F_5	Wire_1 disconnected, [1]
F_6	Wire_1 short to ground, [1]
F_7	Wire_1 short to battery, [1]
F_8	Wire_2 disconnected, [1]
F_9	Wire_2 short to ground, [1]
F_{10}	Wire_2 short to battery, [1]
F_{11}	Wire_3 disconnected, [1]
F_{12}	Wire_3 short to ground, [1]
F_{13}	Wire_3 short to battery, [1]

System behavioral modes of interest are F_3 and F_4, since the desired diagnosis performance is to detect when the switch malfunctions. Analogous with the nomenclature in Chapter 2, the model is defined as:

[1] All other components behave according to the component behavioral mode *nf*

$$\hat{y} = \begin{pmatrix} wire1.p2.i \\ wire1.p2.u \\ wire2.p1.i \\ wire2.p1.u \\ wire3.p2.i \\ wire3.p2.u \end{pmatrix}, desired\ switch\ position, u \in \{off, on\}. z = (u\ \hat{y})$$

$$BM_{example} = \{NF, F_1, ..., F_{13}\}$$

$$Z = \left\{ z \mid \bigcup_{\forall BM \in BM_{example}} M^{BM}(z) = 0 \right\}$$

The resulting SDB for the model and suggested sensor placements can be seen in Table 10. This SDB is manually processed in accordance with the modified test generation algorithm in Section 6.6. Column wire_1.p2.i will be processed for illustration purposes. A restriction is that only intervals outside the interval of *NF* are used for creation of tests. These tests are implemented in Dirigent, see Section 4.4.7, which provides the system behavioral modes that can cause each test to react. The following tests can be created:

$$T_1 : z \in \Theta_{T_1}^C = \{z \mid wire\_1.p2.i \in \{-0.83168 - 0.83166\}, u = off\} \quad T_1 \rightarrow sys = F_4$$

$$T_2 : z \in \Theta_{T_2}^C = \{z \mid wire\_1.p2.i = 0, u = on\} \quad T_2 \rightarrow sys \in \{F_1, F_2, F_3, F_5, F_6, F_8, F_{11}\}$$

$$T_3 : z \in \Theta_{T_3}^C = \{z \mid wire\_1.p2.i \in \{-0.83259 - 0.83237\}, u = on\} \quad T_3 \rightarrow sys = F_7$$

$$T_4 : z \in \Theta_{T_4}^C = \{z \mid wire\_1.p2.i \in \{-416.67 - 416.66\}, u = on\} \quad T_4 \rightarrow sys = F_9$$

$$T_5 : z \in \Theta_{T_5}^C = \{z \mid wire\_1.p2.i \in \{-0.47518 - 0.3723\}, u = on\} \quad T_5 \rightarrow sys = F_{10}$$

This same process is repeated for every column. The result is that all possible tests have been created and can be translated into the decision structure in Table 11.

Table 10. SDB generated in Rodon for the possible sensor placements.

Fault State	U	wire_1.p2.i	wire_1.p2.u	wire_2.p1.i	wire_2.p1.u	wire_3.p2.i	wire_3.p2.u
NF	off	0	12	0	0	0	0
F_1, Bulb disconnected	off	0	12	0	[0 12]	0	0
F_2, ground disconnected	off	0	12	0	[0 12]	0	[0 12]
F_3, switch disconnected	off	0	12	0	0	0	0
F_4, switch pin_short	off	[-0.83168 -0.83166]	[11.975 11.977]	[0.83166 0.83168]	[11.975 11.977]	[0.83166 0.83168]	0
F_5, Wire_1 disconnected	off	0	[0 12]	0	0	0	0
F_6, wire_1 short_to_gnd	off	0	0	0	0	0	0
F_7, wire_1 short_to_batt	off	0	[11.999 12]	0	0	0	0
F_8, wire_2 disconnected	off	0	12	0	[0 12]	0	0
F_9, wire_2 short_to_gnd	off	0	12	0	0	0	0
F_{10}, wire_2 short_to_batt	off	0	12	0	[11.974 11.976]	[0.83159 0.83161]	0

F_{11}, wire_3 disconnected	off	0	12	0	[0 12]	0	[0 12]
F_{12}, wire_3 short_to_gnd	off	0	12	0	0	0	0
F_{13}, Wire_3 short_to_batt	off	0	12	0	0	0	0
NF	on	[-0.83168 -0.83166]	[11.975 11.977]	[0.83166 0.83168]	[11.975 11.977]	[0.83166 0.83168]	0
F_1, bulb disconnected	on	0	12	0	12	0	0
F_2, ground disconnected	on	0	12	0	12	0	12
F_3, switch disconnected	on	0	12	0	0	0	0
F_4, switch pin_short	on	[-0.83168 -0.83166]	[11.975 11.977]	[0.83166 0.83168]	[11.975 11.977]	[0.83166 0.83168]	0
F_5, wire_1 disconnected	on	0	0	0	0	0	0
F_6, wire_1 short_to_gnd	on	0	0	0	0	0	0
F_7, wire_1 short_to_batt	on	[-0.83259 -0.83237]	[11.986 11.99]	[0.83237 0.83259]	[11.986 11.99]	[0.83237 0.83259]	0
F_8, wire_2 disconnected	on	0	12	0	12	0	0

F_9, wire_2 short_to_gnd	on	[-416.67 -416.66]	0	[416.66 416.67]	0	0	0
F_{10}, wire_2 short_to_batt	on	[-0.47518 -0.3723]	[11.986 11.99]	[0.3723 0.47518]	[11.986 11.99]	[0.83237 0.83259]	0
F_{11}, wire_3 disconnected	on	0	12	0	12	0	12
F_{12}, wire_3 short_to_gnd	on	[-0.83168 -0.83166]	[11.975 11.977]	[0.83166 0.83168]	[11.975 11.977]	[0.83166 0.83168]	0
F_{13}, wire_3 short_to_batt	on	[-0.83168 -0.83166]	[11.975 11.977]	[0.83166 0.83168]	[11.975 11.977]	[0.83166 0.83168]	0

Table 11. Resulting decision structure generated with the test generation algorithm based on possible sensor placements.

Tests possible to implement	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}
$T_1 : z \in \Theta^C_{T_1} =$ $\{z \mid wire\_1.p2.i \in [-0.83168\ -0.83166],$ $u = off\}$	0	0	0	1	0	0	0	0	0	0	0	0	0
$T_2 : z \in \Theta^C_{T_2} =$ $\{z \mid wire\_1.p2.u \in [11.975\ 11.977], u = off\}$	0	0	0	1	X	0	0	0	0	0	0	0	0
$T_3 : z \in \Theta^C_{T_3} =$ $\{z \mid wire\_1.p2.u = 0, u = off\}$	0	0	0	0	X	1	0	0	0	0	0	0	0
$T_4 : z \in \Theta^C_{T_4} =$ $\{z \mid wire\_2.p1.i \in [0.83166\ 0.83168], u = off\}$	0	0	0	1	0	0	0	0	0	0	0	0	0
$T_5 : z \in \Theta^C_{T_5} =$ $\{z \mid wire\_2.p1.u \in [11.974\ 11.9749], u = off\}$	X	X	0	0	0	0	0	X	0	X	X	0	0
$T_6 : z \in \Theta^C_{T_6} =$ $\{z \mid wire\_2.p1.u \in [11.975\ 11.976], u = off\}$	X	X	0	X	0	0	0	X	0	X	X	0	0
$T_7 : z \in \Theta^C_{T_7} =$ $\{z \mid wire\_2.p1.u \in [11.976 11.977], u = off\}$	X	X	0	X	0	0	0	X	0	0	X	0	0
$T_8 : z \in \Theta^C_{T_8} =$ $\{z \mid wire\_3.p2.i \in [0.83166\ 0.83168], u = off\}$	0	0	0	1	0	0	0	0	0	0	0	0	0
$T_9 : z \in \Theta^C_{T_9} =$ $\{z \mid wire\_3.p2.i \in [0.83159\ 0.83161], u = off\}$	0	0	0	0	0	0	0	0	0	1	0	0	0
$T_{10} : z \in \Theta^C_{T_{10}} =$ $\{z \mid wire\_3.p2.u > 0.01, u = off\}$	0	X	0	0	0	0	0	0	0	0	X	0	0
$T_{11} : z \in \Theta^C_{T_{11}} =$ $\{z \mid wire\_1.p2.i \in [-0.83259\ -0.83237],$ $u = on\}$	0	0	0	0	0	0	1	0	0	0	0	0	0
$T_{12} : z \in \Theta^C_{T_{12}} =$ $\{z \mid wire\_1.p2.i \in [-416.67\ -416.66], u = on\}$	0	0	0	0	0	0	0	0	1	0	0	0	0
$T_{13} : z \in \Theta^C_{T_{13}} =$ $\{z \mid wire\_1.p2.i \in [-0.47518\ -0.3723],$ $u = on\}$	0	0	0	0	0	0	0	0	0	1	0	0	0

Tests possible to implement, Continued	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}
$T_{14} : z \in \Theta_{T_{14}}^{C} =$ $\{z \mid wire\_1.p2.i = 0, u = on\}$	1	1	1	0	1	1	0	1	0	0	1	0	0
$T_{15} : z \in \Theta_{T_{15}}^{C} =$ $\{z \mid wire\_1.p2.u = 0, u = on\}$	0	0	0	0	1	1	0	0	1	0	0	0	0
$T_{16} : z \in \Theta_{T_{16}}^{C} =$ $\{z \mid wire\_1.p2.u = 12, u = on\}$	1	1	1	0	0	0	0	1	0	0	1	0	0
$T_{17} : z \in \Theta_{T_{17}}^{C} =$ $\{z \mid wire\_1.p2.u \in [11.986\ 11.99], u = on\}$	0	0	0	0	0	0	1	0	0	1	0	0	0
$T_{18} : z \in \Theta_{T_{18}}^{C} =$ $\{z \mid wire\_2.p1.i \in [416.66\ 416.67], u = on\}$	0	0	0	0	0	0	0	0	1	0	0	0	0
$T_{19} : z \in \Theta_{T_{19}}^{C} =$ $\{z \mid wire\_2.p1.i \in [0.8323767\ 0.83259],$ $u = on\}$	0	0	0	0	0	0	1	0	0	0	0	0	0
$T_{20} : z \in \Theta_{T_{20}}^{C} =$ $\{z \mid wire\_2.p1.i \in [0.3723\ 0.47518],$ $u = on\}$	0	0	0	0	0	0	0	0	0	1	0	0	0
$T_{21} : z \in \Theta_{T_{21}}^{C} =$ $\{z \mid wire\_2.p1.i = 0, u = on\}$	1	1	1	0	1	1	0	1	0	0	1	0	0
$T_{22} : z \in \Theta_{T_{22}}^{C} =$ $\{z \mid wire\_2.p1.u \in [11.986\ 11.99], u = on\}$	0	0	0	0	0	0	1	0	0	1	0	0	0
$T_{23} : z \in \Theta_{T_{23}}^{C} =$ $\{z \mid wire\_2.p1.u = 0,$ $u = on\}$	0	0	1	0	1	1	0	0	1	0	0	0	0
$T_{24} : z \in \Theta_{T_{24}}^{C} =$ $\{z \mid wire\_2.p1.u = 12, u = on\}$	1	1	0	0	0	0	0	1	0	0	1	0	0
$T_{25} : z \in \Theta_{T_{25}}^{C} =$ $\{z \mid wire\_3.p2.i \in [0.83237\ 0.83259], u = on\}$	0	0	0	0	0	0	1	0	0	1	0	0	0
$T_{26} : z \in \Theta_{T_{26}}^{C} =$ $\{z \mid wire\_3.p2.i = 0, u = on\}$	1	1	1	0	1	1	0	1	1	0	1	0	0
$T_{27} : z \in \Theta_{T_{27}}^{C} =$ $\{z \mid wire\_3.p2.u = 12, u = on\}$	0	1	0	0	0	0	0	0	0	0	1	0	0

Since it was of interest to detect and isolate switch malfunctions, the decision structure must be analyzed to see if there are tests or combination of tests that accomplishes this criterion. For fault isolation of switch pin short, F_4, the following tests can be chosen for implementation:

$$T_1 : z \in \Theta_{T_1}^C = \{z \mid wire\_1.p2.i \in [-0.83168\ -0.83166], u = off\},$$
$$T_4 : z \in \Theta_{T_4}^C = \{z \mid wire\_2.p1.i \in [0.83166\ 0.83168], u = off\},$$
$$T_8 : z \in \Theta_{T_8}^C = \{z \mid wire\_3.p2.i \in [0.83166\ 0.83168], u = off\}.$$

For fault isolation of switch disconnected, F_3, the following combination of tests can be implemented, which is represented as taking the intersection of the row representing T_{16} with the inverse of the row representing T_{24} in Table 11:

$$T_{16} : z \in \Theta_1^C = \{z \mid wire\_1.p2.u = 12, u = on\},$$
$$T_{24} : z \in \Theta_2^C = \{z \mid wire\_2.p1.u = 12, u = on\},$$
$$T_{F_3} = T_{16} \wedge \neg T_{24} : z \in \Theta_{F_3}^C = \Theta_{T_{16}}^C \cap \Theta_{T_{24}} \quad T_{F_3} \rightarrow sys = F_3$$

For isolation of F_3, this means that the two sensor measurements wire_1.p2.u and wire_2.p1.u will be implemented and can be used for further test creation.

Unfortunately, none of the signals wire_1.p2.u and wire_2.p1.i is included in the tests that can be implemented to isolate if the switch is behaving according to pin short. This means that one more sensor is needed to be able to isolate the system behavioral mode switch pin short as well, and is decided by the test chosen for isolation of F_4. In the example the test for isolation was chosen as:

$$T_{F_4} : z \in \Theta_{F_4}^C = \{z \mid wire\_3.p2.i \in [0.83166\ 0.83168], u = off\}$$

In summary the sensor placement sufficient to achieve detection of all faults in the switch will be:

- wire_1.p2.u
- wire_2.p1.u
- wire_3.p2.i

As a consequence of this sensor placement choice, additional tests can be implemented in the diagnosis system. All tests possible to implement, for the given sensor placement choice, are listed in the decision structure in Table 12. It is possible to analyze how these tests can be combined for fault isolation, by using the Rodon feature DRsimulator.

It can be concluded that Rodon can be used for guidance of sensor placement which was one of the desired features of the vision.

Table 12. Resulting decision structure from the chosen optimal sensors placement.

Tests possible to implement	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}
$T_1 : z \in \Theta_{T_1}^C =$ $\{z \mid wire\_1.p2.u \in [11.975\,11.977], u = off\}$	0	0	0	1	X	0	0	0	0	0	0	0	0
$T_2 : z \in \Theta_{T_2}^C =$ $\{z \mid wire\_1.p2.u = 0, u = off\}$	0	0	0	0	X	1	0	0	0	0	0	0	0
$T_3 : z \in \Theta_{T_3}^C =$ $\{z \mid wire\_2.p1.u \in [11.974\,11.9749], u = off\}$	X	X	0	0	0	0	0	X	0	X	X	0	0
$T_4 : z \in \Theta_{T_4}^C =$ $\{z \mid wire\_2.p1.u \in [11.975\,11.976], u = off\}$	X	X	0	X	0	0	0	X	0	X	X	0	0
$T_5 : z \in \Theta_{T_5}^C =$ $\{z \mid wire\_2.p1.u \in [11.976\,11.977], u = off\}$	X	X	0	X	0	0	0	X	0	0	X	0	0
$T_6 : z \in \Theta_{T_6}^C =$ $\{z \mid wire\_3.p2.i \in [0.83166\ 0.83168], u = off\}$	0	0	0	1	0	0	0	0	0	0	0	0	0
$T_7 : z \in \Theta_{T_7}^C =$ $\{z \mid wire\_3.p2.i \in [0.83159\ 0.83161], u = off\}$	0	0	0	0	0	0	0	0	0	1	0	0	0
$T_8 : z \in \Theta_{T_8}^C =$ $\{z \mid wire\_1.p2.u = 0, u = on\}$	0	0	0	0	1	1	0	0	1	0	0	0	0
$T_9 : z \in \Theta_{T_9}^C =$ $\{z \mid wire\_1.p2.u = 12, u = on\}$	1	1	1	0	0	0	0	1	0	0	1	0	0
$T_{10} : z \in \Theta_{T_{10}}^C =$ $\{z \mid wire\_1.p2.u \in [11.986\,11.99], u = on\}$	0	0	0	0	0	0	1	0	0	1	0	0	0
$T_{11} : z \in \Theta_{T_{11}}^C =$ $\{z \mid wire\_2.p1.u \in [11.986\,11.99], u = on\}$	0	0	0	0	0	0	1	0	0	1	0	0	0

$T_{12} : z \in \Theta_{T_{12}}^{C} =$ $\{z \mid wire\_2.p1.u = 0, u = on\}$	0	0	1	0	1	1	0	0	1	0	0	0	0
$T_{13} : z \in \Theta_{T_{13}}^{C} =$ $\{z \mid wire\_2.p1.u = 12, u = on\}$	1	1	0	0	0	0	0	1	0	0	1	0	0
$T_{14} : z \in \Theta_{T_{14}}^{C} =$ $\{z \mid wire\_3.p2.i \in [0.83237\ 0.83259], u = on\}$	0	0	0	0	0	0	1	0	0	1	0	0	0
$T_{15} : z \in \Theta_{T_{15}}^{C} =$ $\{z \mid wire\_3.p2.i = 0, u = on\}$	1	1	1	0	1	1	0	1	1	0	1	0	0

7.3 Test design

This section will discuss how diagnostic tests can be designed for the on board diagnosis system. Design of a test is equivalent to determining the rejection region of the test, Θ_T^C, and determining which system behavioral modes that can cause the test to react, S^1.

Tests for on board use can, as earlier stated, both be used for fault isolation and for functional monitoring purposes. Assuming that a sensor placement choice has been made, the general process for both test types is to decide a rejection region of the test.

The difficulty when designing tests lies in the choice of the rejection region, since this affects the achieved diagnosis performance. Test rejection regions for fault isolation and for functional monitoring purposes can be chosen manually by the engineer.

Another way to choose the rejection regions for fault isolation purposes is by using the method in Section 6.6 or the method in Section 6.7. The steps presented in these algorithms can be followed to generate all diagnostic tests for fault isolation purposes. These tests will all be of limit checking type, which means that they decide whether or not a sensor value is above or below a certain value. This leads to the system state space being divided into angular regions, which the method uses as rejection regions.

The chosen rejection regions can then be implemented in Dirigent, which automatically decides the set S^1. If this is done for all tests desired in the diagnosis system, then a decision structure can be created. This decision structure can be analyzed to decide if the diagnosis performance is sufficient.

In summary, Rodon is used to generate the SDB, which is used as input to the method. The method provides a structured approach to generate all limit checking tests and the resulting decision structure. This means that, since Rodon provides the necessary input to the method, Rodon in combination with the method can be used for test design.

The conclusion that can be drawn is that Rodon can be used in combination with the method for design and evaluation of tests. Rodon does not fulfill the requirement of the vision concerning design of tests, but can provide the set S^1 automatically. It is hard to say which diagnostic performance that is achieved by using the method in combination with Rodon. An evaluation would be of interest and is a recommendation for future work.

7.4 Generation of decision structures

As stated earlier in Chapter 3, Saab is interested in automatic generation of the decision structure for designed diagnostic tests that are to be implemented in the on board diagnosis system for FDI. This decision structure contains information of which system behavioral modes that can cause each test to react. The decision structure is to be used in the fault isolation logic block during operation together with the values of the diagnostic tests and the desired output is the diagnosis that is consistent with the values of the tests. A generic fault isolation logic block with software must be constructed to achieve the process described above.

An example can perhaps clarify the function of the block. Assume that three diagnostic tests, T_1, T_2, and T_3, are used in the diagnosis system of a subsystem during operation. There are four system behavioral modes that the subsystem can behave according to: *NF*, F_1, F_2, and F_3. The decision structure can be seen in Table 13.

Table 13. Decision structure of an example that illustrates the function of the fault isolation logic block.

	NF	F_1	F_2	F_3
T_1	0	X	X	0
T_2	0	0	X	X
T_3	0	0	0	X

At one juncture the values of the test are $T =< T_1\ T_2\ T_3 >=< 0\,1\,1 >$. The task for the software in the fault isolation logic block will then be to reach the conclusion that $sys = F_3$. To conclude, the purpose with the fault isolation logic block is to perform an automatic fault isolation analysis and present the diagnosis based upon the values of the diagnostic tests and the decision structure in use.

7.4.1 Rodon generation of decision structures

If the diagnostic tests, that are to be implemented in the on board diagnosis system, are implemented in Rodon, according to Section 4.4.6, then Dirigent can be used to generate what is called diagnostic rules in Rodon, as described in Section 4.4.7. The diagnostic rules are listed in a code file and contain the same information as a decision structure. This diagnostic rule file can be used as input in the feature DRsimulator to generate a corresponding file, which is a list in a c source format that also contains the same information as a decision structure. This c source code can be used in the fault isolation logic block. The process described above is illustrated in Figure 28.

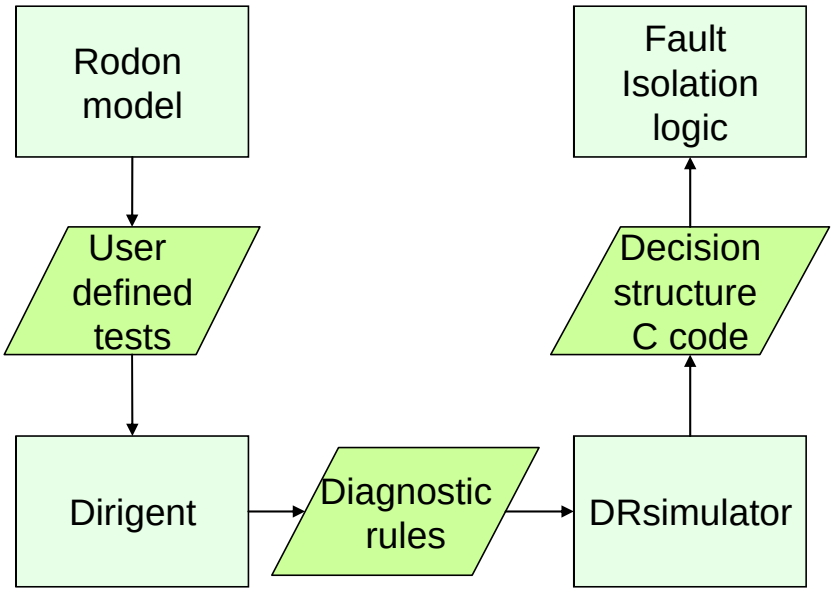


Figure 28. Illustration of the process how the Rodon model is used together with the features Dirigent and DRsimulator to generate decision structure code, which is used as input to the fault isolation logic block.

7.4.2 Diagnostic rules

Pseudo code for how the diagnostic rules are listed can be seen in Figure 29 and a real example of code can be found in Appendix A.

Failure codes:
1: fc_1
…….
N: fc_N

System behavioral modes:
1: FM_1
…….
M: FM_M

Diagnostic Rules:
R_1:
if (fc_1 = true)
suspected (System behavioral mode set_1)
…….
R_N:
if (fc_N = true)
suspected (System behavioral mode set_N)

Figure 29. Illustration of the generated diagnostic rules expressed in pseudo code

The pseudo code consists of failure codes $fc_i, i = 1,..., N$. These failure codes correspond to the tests that are to be implemented in the on board diagnosis system in a system, *sys*. Furthermore, the pseudo code contains a listing of possible system behavioral modes that *sys* can behave according to. If T_1 is the only test of the diagnosis system and corresponds to the failure code fc_1 then fc_1 will be included in the failure code section. If T_1 indicates the system behavioral modes F_1 and F_2, then these will be listed in the system behavioral modes part. This will also result in the part with diagnostic rules only being filled up with one rule since there only is one diagnostic test. This rule will contain the condition $if\ (fc_1 = true)$ and include the system behavioral modes F_1 and F_2 as the suspects. If test T_1 reacts in the diagnosis system, the result will be that the failure code fc_1 will be set true, and the decision will then be $sys = S^1 = \{F_1, F_2\}$. If the diagnosis system was to be supplemented with one more test

$T_2 : z \in \Theta_{T_2}^C = \{z \mid u = 2, y_2 = 3\}, z \in \Theta_{T_2}^C \rightarrow fc_2 = true \Leftrightarrow sys \in S^1 = \{F_2, F_3\}$

then the supplements in the code would be an extension of failure codes with fc_2 and the inclusion of system behavioral modes F_3, and a new additional diagnostic rule. This new rule would have the condition $if\ (fc_2 = true)$ and suspected system behavioral modes F_2 and F_3.

7.4.3 Decision structure c code

The generated diagnostic rule file can be used in the feature DRsimulator to generate a c source code file, which for the diagnostic rule code in Appendix A will be:

```
#define DR_SYSTEM_SOURCE_C_
#include "drSystemSource.h"

static const unsigned int data[] = {

    /* Choice 0 */
    1,
    /* Choice 1 */
    2,
    /* Choice 2 */
    3,
    /* Choice 3 */
    4, 5,
    /* Choice 4 */
    6,
    /* Choice 5 */
    7,
    /* Choice 6 */
    8,
    /* Choice 7 */
    9,
    /* Choice 8 */
    10,
    /* Choice 9 */
    11,
```

```
        /* Choice 10 */
        12,
        /* Choice 11 */
        13, 14,
        /* Choice 12 */
        15,
        /* Choice 13 */
        16,
        /* Choice 14 */
        17,
        /* Choice 15 */
        18,
        /* Choice 16 */
        19,
        /* Choice 17 */
        20,
        /* Choice 18 */
        21,
        /* Choice 19 */
        22,
        /* Choice 20 */
        23,
        /* Choice 21 */
        24,
        /* Choice 22 */
        25,
        /* Choice 23 */
        26,
        /* Choice 24 */
        27,
        /* Choice 25 */
        28,
        /* Choice 26 */
        29, 30,
        /* Choice 27 */
        31,
        /* Choice 28 */
        32,
        /* Choice 29 */
        33,
        /* Choice 30 */
        34,
        /* Choice 31 */
        35,
        /* Choice 32 */
        36,

        /* Candidates of rule 0 */
        10, 11, 13, 28, 29, 32, 33, 36,
        /* Candidates of rule 1 */
        3, 4, 9,
        /* Candidates of rule 2 */
        1, 2, 6, 7, 8, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
        27, 30, 31, 34, 35,
        /* Candidates of rule 3 */
        5
};

static const struct sChoiceSource choices[] = {
```

```
        {&data[0], 1},
        {&data[1], 1},
        {&data[2], 1},
        {&data[3], 2},
        {&data[5], 1},
        {&data[6], 1},
        {&data[7], 1},
        {&data[8], 1},
        {&data[9], 1},
        {&data[10], 1},
        {&data[11], 1},
        {&data[12], 2},
        {&data[14], 1},
        {&data[15], 1},
        {&data[16], 1},
        {&data[17], 1},
        {&data[18], 1},
        {&data[19], 1},
        {&data[20], 1},
        {&data[21], 1},
        {&data[22], 1},
        {&data[23], 1},
        {&data[24], 1},
        {&data[25], 1},
        {&data[26], 1},
        {&data[27], 1},
        {&data[28], 2},
        {&data[30], 1},
        {&data[31], 1},
        {&data[32], 1},
        {&data[33], 1},
        {&data[34], 1},
        {&data[35], 1}};

static const struct sChoiceSpaceSource choiceSpace = {&choices[0], 33};

static const struct sRuleSource rules[] = {
        {'S', "*1***", &data[36], 8, 1},
        {'S', "**1**", &data[44], 3, 2},
        {'S', "***1*", &data[47], 25, 3},
        {'S', "****1", &data[72], 1, 4}};


const struct sDrSystemSource drSystemSource =
{"0.0.1", "", &choiceSpace, &rules[0], 4};


const struct sDrSystemSource drSystemSource =
{"0.0.1", "", &choiceSpace, &rules[0], 2};
```

This source code is generated from a diagnostic rule file that contains 36 system behavioral modes and four diagnostic rules. These 36 system behavioral modes describe how 33 components can behave. The code starts by filling a data struct with which system behavioral modes that can cause malfunction of which component, and which system behavioral modes that are indicated by the diagnostic rules. In the code, choice *i* means that component *i* is faulty and is listed together with the system behavioral modes that can cause malfunction of component *i*. The choice data is inserted in the data struct choices[]. Choices can be said to represent the knowledge of which faulty components that a system behavioral mode indicates. This information is then included in the struct choiceSpace[].

The diagnostic rules data is inserted in the struct rules[], in which 'S' represents that the system behavioral modes indicated by the diagnostic rules are of type suspect, which means that they indicate components suspected of being faulty. Instead of 'S', 'C' can be used and means that the diagnostic rule is of the type that clears components from being faulty. The struct rules[] also include a string that states which failure code that is included in the condition part of the diagnostic rule. For the example code above there are four failure codes and the first string "*1***" means that fc_1 is included in the condition part of the first rule. The second string "**1**" means that fc_2 is included in the condition part of the second rule. For a string in general "*a_1…a_n", a_i can either be the symbol * or 0 or 1. If the symbol of a_i is 1 then fc_i is included and if it is 0 the negation of fc_i is included. If the symbol of a_i is * then it does not matter whether fc_i is true or not, which is equivalent with do not care (dc).

After the string follows the reference of where the listing of the suspected system behavioral modes can be found in data[], for the diagnostic rule. The following number states how many places in data[] that contain these suspected system behavioral modes. The number that ends each row indicates which rule that the information is valid for.

A conclusion is that since the source code file contains the same information as a decision structure, the code is suitable to use in the generic fault isolation block. This means that Rodon includes the desired feature, of the vision, to generate the decision structure corresponding to the diagnostic tests of the diagnosis system.

The Dirigent feature is very useful when it comes to analyzing functional monitoring diagnostic tests. Such tests are constructed to indicate loss of function and which system behavioral modes that cause the loss are generally unknown. This can be solved with Rodon since it can be used to decide these system behavioral modes.

7.5 Fault isolation

7.5.1 Fault isolation through hypothesis tests

In Gripen the hypothesis tests of the on board diagnosis system are primary designed to detect loss of functionality in a system, but some tests are also designed for fault isolation purposes. An investigation is made for each hypothesis test with the purpose to list possible failure modes that can cause it to react. When all tests have been investigated this knowledge is translated into a fault isolation flow chart with the purpose to generate a set of suspected faulty components depending on which tests that have reacted. This flow chart is then translated into fault isolation software. By logging tests that reacts during flight, fault isolation can hopefully be achieved in maintenance by using the software. If not, additional measurements need to be made manually by technicians. Suitable measure points are chosen based on documented decision trees or the technicians own experience.

Fault isolation at Saab today can be illustrated as in Figure 30.

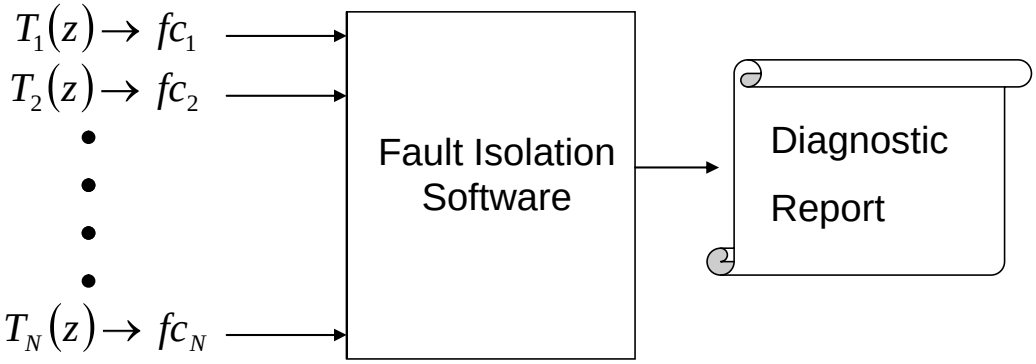


Figure 30. Fault isolation at Saab today. Fault isolation software is feed with failure codes to determine faults.

7.5.2 Fault isolation with Rodon

Off board diagnostics can be performed with Rodon through MBD with an AI approach, which is, as earlier stated, the process to compare if measurements from the system are consistent with the ones calculated from the model. An inconsistency indicates a faulty system. Observations and failure codes are loaded into the model and a diagnosis statement is generated which indicates the component or components suspected of being faulty. In the last case the number of suspects can be further reduced through additional measurements, which can be suggested by Rodon.

Fault isolation with Rodon can be illustrated as in Figure 31.

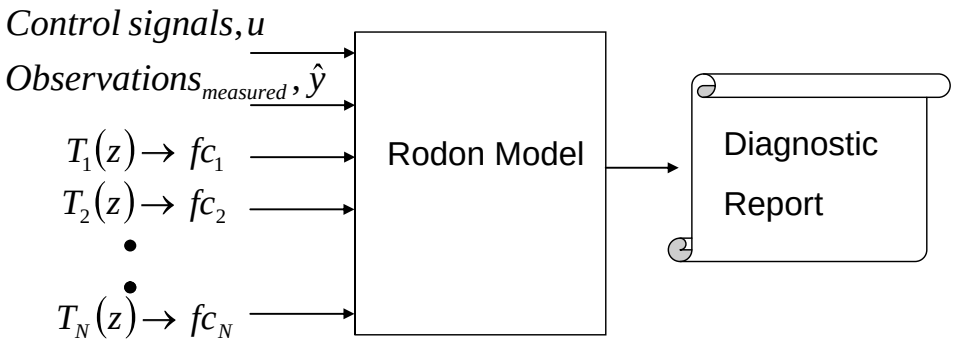


Figure 31. Fault isolation with Rodon. Observations and failure codes are feed to the Rodon model to determine the system faults.

7.5.3 Comparison of the two fault isolation methods

Achieved fault isolation for the method with hypothesis tests is dependant on how the tests are designed. It can either be a residual or a limit check and these two types of the hypothesis fault isolation method will be compared separately with the Rodon fault isolation method.

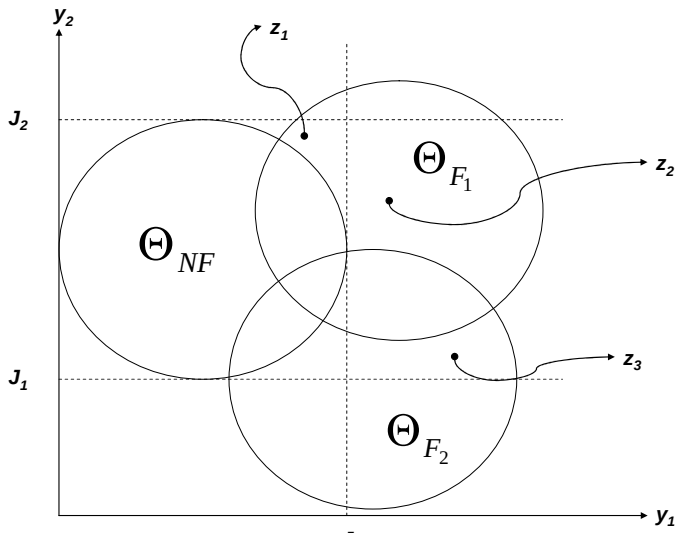
7.5.3.4 Rodon versus limit checking

Assume a system, *sys*, with three behavioral modes: *NF*, F_1, and F_2. It is monitored through two sensors, y_1 and y_2. Three hypothesis tests T_1, T_2, and T_3, with thresholds J_1, J_2, and J_3 are used in the on board diagnosis system according to the method described in Section 7.5.1. If no test reacts the decision of the fault isolation logic will be $sys = NF$. An illustration of the behavioral modes of the system and the thresholds can be found in Figure 32. The tests are defined as:

$$T_1: \quad z \in \Theta_{T_1}^C = \{z \mid y_2 < J_1\} \rightarrow sys = S^1 = F_2$$

$$T_2: \quad z \in \Theta_{T_2}^C = \{z \mid y_2 > J_2\} \rightarrow sys = S^1 = F_1$$

$$T_3: \quad z \in \Theta_{T_3}^C = \{z \mid y_1 > J_3\} \rightarrow sys \in S^1 = \{F_1, F_2\}$$



Figur 32. A system with three behavioral modes on which off board diagnostics is performed.

Two different observations $z_1$and z_2 have been measured at two different junctures in the system when it is behaving in accordance with $sys = F_1$. Observation z_3 has been measured when $sys = F_2$. Observation z_2 and z_3 will both affect T_3 to react and the decision will for both be $sys \in \{F_1, F_2\}$. No tests will react for observation z_1 and this leads to the conclusion that $sys \in \{NF, F_1, F_2\}$.

If a Rodon model that describes the system well instead is used for fault isolation, the observations z_1, z_2, and z_3 would respectively lead to the strongest conclusion that $sys = NF$, $sys = F_1$, and $sys = F_2$. The reason for this is that Rodon uses all information and dependencies between variables, contrary to the method with limit checking hypothesis tests. To conclude, fault isolation performance achieved by Rodon if it terminates will be better then the method with limit checks.

7.5.3.5 Rodon versus residuals

Tests can be designed so that their rejection regions Θ_{Ti}^{C} cover the system behavioral modes Θ_{BMi} in a better way then limit checking tests. The process of how to generate these is to first derive and formulate model equations of the system. Some of the variables in the equations will be measured and thereby considered to be known. The model equations are used to analytically eliminate unknown variables and the remaining equations, which are only dependant on measured variables, can be used to generate test quantities called residuals. These residuals constitute the hypothesis tests of the on board diagnosis system and are used for MBD through FDI.

A more detailed description of how to create residuals will not be further discussed. Instead it is assumed that residuals tests have been created with rejection regions that cover the system behavioral modes perfectly. For system behavioral mode F_1 and F_2, this means residual tests with rejection region $\Theta_{T_{F_1}}^{C} = \Theta_{F_1} \setminus (\Theta_{NF} \cup \Theta_{F_2})$ and $\Theta_{T_{F_2}}^{C} = \Theta_{F_2} \setminus (\Theta_{NF} \cup \Theta_{F_1})$.

Fault isolation performance achieved by correctly constructed residuals will be of high quality, but they demand more design effort since tests must be derived. In comparison with Rodon, for which there is no need for test construction, the use of residuals demands more labor.

It should be stressed that the process of creating all residuals manually is very cumbersome and the risk of missing some is severe. If an automatic approach is used to generate all residuals then this problem is solved. But some residuals can turn out to be instable, since equations might be inverted during design, and thereby unusable. If stabilized they might be of use but the fault isolation performance will be degraded and as a result some parts of the system behavioral modes will not be covered by any tests. On the contrary there is no guarantee that the solving process will be stable in Rodon or be jammed in calculation loops.

Both Rodon and residuals will provide high quality fault isolation since they are model based diagnosis approaches. The use of residuals demands extra work since hypothesis tests need to be derived. One of the advantages with using residuals is that they can be used for real time on board diagnostics and that the design of them provides a mean to handle stochastics by setting the threshold in accordance with desired false alarm probability. The pros and cons with the different approaches are summarized in Table 14.

Table 14. Pros and cons for different fault isolation approaches.

	Pros	Cons
Use of limit checks	• Can be performed in real time for on board functional monitoring • No absolute need to derive a model	• Degraded fault isolation performance
Use of residuals	• Can be performed in real time for on board functional monitoring • Provides a mean to handle stochastic variations when constructing tests • High quality fault isolation	• Demands extra work to design residuals • Stability might be difficult to obtain
Use of Rodon	• No need for test construction • High quality fault isolation	• Can not be performed in real time for on board functional monitoring • Might result in unstable solving processes and calculation loops

7.5.4 Can MBD with an AI approach be used on board?

MBD with an AI approach for on board fault isolation would in theory be a very powerful approach since the isolation performance would be of high quality. A drawback is that the calculation time is unknown since a consistent system behavioral mode can be found quickly or slowly depending on the search path. There is also a risk of calculation loops, in which case the calculation time will be infinite. Furthermore, propagation will continue until the user decided tolerance is achieved and the time for this is very hard, perhaps even impossible, to estimate. The conclusion drawn is that MBD with an AI approach will not be suitable for on board use for real time systems with fast calculation time demands, i.e. an aircrafts control system, since it can not be guaranteed that the search will finish in time. For real time systems with slower calculation time demands, i.e. monitoring of cabin systems in commercial aircrafts, MBD with an AI approach might possibly be suitable for on board use.

7.6 Summary

The vision presented in Chapter 3 included the following desired features of a diagnostic tool:

- Performance of failure analysis
- Guidance for sensor placements
- Test design for an on board diagnosis system
- Generation of the decision structure corresponding to the diagnostic tests of the on board diagnosis system. This decision structure is to be used for fault isolation
- Achieve high quality fault isolation during maintenance by an AI model based diagnosis approach

Section 7.1 reached the conclusion that it is possible to perform failure analysis with the help of Rodon. How Rodon can be used for guidance for sensor placements was discussed in the next section. Section 7.3 determined that Rodon can provide the conclusions S^1 for implemented tests, but not design tests. The next section reached the conclusion that Rodon can be used to generate the decision structure corresponding to the diagnostic tests of the on board diagnosis system. This decision structure can be used in the fault isolation of the diagnosis system. The concluding section dealt with how Rodon can be used for high quality fault isolation.

Chapter 8

Discussion

The main goal of this thesis project was to evaluate the functinality of the diagnostic tool Rodon and decide how the tool could be used for development and operations at Saab Aerosystems. Chapter 3 presented a vision of what was desired from a diagnostic tool. Chapter 5 included a general analysis of the tool, including what benefits the tool have, problems that can occur, limitations, and which type of systems that are suitable to model. Chapter 7 analyzed what parts of the vision that Rodon can fulfill. This chapter will conclude with recommendations for future work.

8.1 Conclusion

Rodon is an interesting diagnostic tool since it can be used for several various purposes, during development and operations. The tool performs MBD with methods from AI, see Section 4.2, for fault isolation which is a powerful approach.

A model based development approach will be required if Rodon is introduced. Detailed models of systems of interest will be necessary to create, including the nominal behavior of the system and faulty behaviors. Two different ways to model faulty behaviors was described in sections 5.2.1 and 5.2.2. In order to achieve high quality fault isolation it is necessary to have complete and consistent models. Such models require much work to accomplish and this was discussed in sections 5.2.3 - 5.2.5.

Preferable characteristics of systems to be modeled in Rodon are, according to Section 5.4.1, that they should be static, use discrete control signals, and have well defined system behavioral modes.

Which parts of the vision that Rodon fulfills is summarized in Section 7.6 and concludes that the tool can be used for failure analysis, guidance of sensor placements, evaluation of tests, generation of decision structures, and high quality fault isolation. Design of tests is a gap in the vision that Rodon does not fulfill, and therefore two different methods were presented in sections 6.6 and 6.7 that utilizes Rodon to generate all possible limit checking tests.

In conclusion, Rodon can be very useful in several different aspects if introduced, but benefits gained by using Rodon will have to be compared to the labor cost of creating good models.

8.2 Future work

The scope of this thesis project has been to analyze what Rodon is capable of and how it can be used during development and operations. This has been performed by providing a general overview and coverage of the tool, which can serve as a base for future studies. Gained benefits are of interest to further explore and a recommendation is a case study following from developments to operations. This case study should include the modeling of suitable systems, both static and dynamic, for which various features of the tool are used, so that a deeper analysis can be achieved.

If the method described in Section 6.6 is to be of practical use for generation of diagnostic tests, then the algorithm must be automated. It is of interest to further explore the process of generation of decision structures for use in the fault isolation logic in the on board diagnosis system, as described in Section 7.4.

Bibliography

[1] Nyberg M., Frisk E., *Model Based Diagnosis of Technical Processes*, Linköping, 2002

[2] Blanke M., Kinneart M., Lunze J., Staroswiecki M., *Diagnosis and Fault-Tolerant Control*, Springer-Verlag, Berlin, Germany, 2003. ISBN 3540010564.

[3] Ljung L., Glad T., *Modellbygge och Simulering*, Studentlitteratur, Lund, 2004. ISBN 9144318715.

[4] Hamscher W., Console L., de Kleer J., *Readings in Model-Based Diagnosis*, Morgan Kaufmann Publishers, San Fransisco, CA, USA, 1992. ISBN 1558602496

[5] Bunus P., *Debugging and Structural Analysis of Declarative Equation-Based Languages*, Department of Computer and Information Science Linköpings universitet, 2002. ISBN 9173733822

Appendix A

/** This file was generated with RODON!*/

Variables (4):

1 aIUnew_1.warning_101_ONE_RECTIFIER_US <GR: !FC>

2 aIUnew_1.warning_105_BATT_CHARGE_MALF <GR: !FC>

3 aIUnew_1.warning_106_BATT_DRAIN <GR: !FC>

4 aIUnew_1.warning_107_BATT_CHARGE_INCORR_ACT <GR: !FC>

Candidates (36):

1 dC_PowerSystem_1/_2PA disconnected <W: 1>

2 dC_PowerSystem_1/_6PA disconnected <W: 1>

3 dC_PowerSystem_1/_16PA/coil disconnected <W: 1>

4 dC_PowerSystem_1/_16PA/switch disconnected <W: 1>

5 dC_PowerSystem_1/_16PA/switch pin_short <W: 1>

6 dC_PowerSystem_1/_24PA/coil disconnected <W: 1>

7 dC_PowerSystem_1/_24PA/switch pin_short <W: 1>

8 dC_PowerSystem_1/groundnode_2 disconnected <W: 1>

9 dC_PowerSystem_1/groundnode_3 disconnected <W: 1>

10 dC_PowerSystem_1/tRUtest2_1 notOK <W: 1>

11 dC_PowerSystem_1/tRUtest2_2 notOK <W: 1>

12 dC_PowerSystem_1/wireIdealDiscStgStb_19 short_to_gnd <W: 1>

13 dC_PowerSystem_1/wireIdealDiscStgStb_21 disconnected <W: 1>

14 dC_PowerSystem_1/wireIdealDiscStgStb_21 short_to_gnd <W: 1>

15 dC_PowerSystem_1/wireIdealDiscStgStb_22 short_to_gnd <W: 1>

16 dC_PowerSystem_1/wireIdealDiscStgStb_23 short_to_gnd <W: 1>

17 dC_PowerSystem_1/wireIdealDiscStgStb_25 short_to_gnd <W: 1>

18 dC_PowerSystem_1/wireIdealDiscStgStb_26 short_to_gnd <W: 1>

19 dC_PowerSystem_1/wireIdealDiscStgStb_28 short_to_gnd <W: 1>

20 dC_PowerSystem_1/wireIdealDiscStgStb_29 short_to_gnd <W: 1>

21 dC_PowerSystem_1/wireIdealDiscStgStb_30 short_to_gnd <W: 1>

22 dC_PowerSystem_1/wireIdealDiscStgStb_31 disconnected <W: 1>

23 dC_PowerSystem_1/wireIdealDiscStgStb_32 short_to_gnd <W: 1>

24 dC_PowerSystem_1/wireIdealDiscStgStb_34 disconnected <W: 1>

25 dC_PowerSystem_1/wireIdealDiscStgStb_35 disconnected <W: 1>

26 dC_PowerSystem_1/wireIdealDiscStgStb_64 short_to_gnd <W: 1>

27 dC_PowerSystem_1/wireIdealDiscStgStb_65 disconnected <W: 1>

28 dC_PowerSystem_1/wireIdealDiscStgStb_74 disconnected <W: 1>

29 dC_PowerSystem_1/wireIdealDiscStgStb_75 disconnected <W: 1>

30 dC_PowerSystem_1/wireIdealDiscStgStb_75 short_to_gnd <W: 1>

31 dC_PowerSystem_1/wireIdealDiscStgStb_76 short_to_gnd <W: 1>

32 dC_PowerSystem_1/wire_4 disconnected <W: 1>

33 dC_PowerSystem_1/wire_5 disconnected <W: 1>

34 dC_PowerSystem_1/wire_11 disconnected <W: 1>

35 dC_PowerSystem_1/wire_12 disconnected <W: 1>

36 groundnode_1 disconnected <W: 1>

Rules (4):

R1: aIUnew_1.warning_101_ONE_RECTIFIER_US

if (1)

suspect (10 11 13 28 29 32 33 36)

R2: aIUnew_1.warning_105_BATT_CHARGE_MALF

if (2)

suspect (3 4 9)

R3: aIUnew_1.warning_106_BATT_DRAIN

if (3)

suspect (1 2 6 7 8 11 12 14 15 16 17 18 19 20 21 22 23 24 25 26 27 30 31 34 35)

R4: aIUnew_1.warning_107_BATT_CHARGE_INCORR_ACT

if (4)

suspect (5)

Appendix B

Property marks

Here are a collocation of some predefined property marks.

Inputs

Name	Usage
sw	Variables which sets the value of a control signal. In electrical systems this is often switch positions, therefore “sw”.
sw-d	Control signal variables with default value.
a	Action, this property mark can be used for actions where sw and sw-d are inappropriate.
fm	Behavioral mode. Variable represents different types of component behavioral modes.

Outputs

Name	Usage
obs	Used for variables which represents an observable quantity.
fc	Failure code.
t	Test result. General property mark for variables that represents the result of tests.
t-vis	Result of a manually visual check.
t-cont	Result of a manually continuity test of a wire.
t-short	Result of a manually short to ground test of a wire.
t-curr	Result of a manually inline current measurement.
m-r	Result of a manually measurement result.