Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2017

Parsing and Validation of Modelica Models Utilising Fault Diagnosis

Karin Lockowandt

LINKÖPING UNIVERSITY

Master of Science Thesis in Electrical Engineering

Parsing and Validation of Modelica Models Utilising Fault Diagnosis

Karin Lockowandt

LiTH-ISY-EX--17/5032--SE

Supervisor: **Viktor Leek**
ISY, Linköpings universitet
Ylva Nilsson
Saab Aeronautics

Examiner: **Erik Frisk**
ISY, Linköpings universitet

Division of Vehicular Systems
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

Models have become an indispensable tool within most industrial sectors and are used to reduce costs, enhance the performance of a system etc. The computer support within modelling is extensive, whereof the programming language Modelica is eminent, especially for multi-domain models. Dymola, a commercial program, is built on Modelica and is foremost used for simulation purposes, but many applications for which models are useful are not supported by Dymola. Instead other tools, e.g. Matlab, could be used to exploit the full potential of a model, which means that it first would be needed to be translated. This master's thesis examines one of the possible ways to accomplish this. Specifically the possibility to translate Modelica-models via an XML file, generated by Dymola, is examined. The structure and content of this file is explored, and based thereupon a software is implemented in Python, which successfully translates the models constituting the base for this thesis. Specifically the method was developed on a model of a sub-system of Saab 39 Gripen air-plane.

Besides porting models between different languages, it is of great interest to determine how well a model describes the system on which it is based. Hence a new method for model validation is developed using the Matlab Fault Diagnosis Toolbox, which also determines the Matlab syntax of the Modelica translation. The novelty with the developed method, compared to traditional model validation methods, is that it is equation based. It is meant to point out specifically which equations are poorly fitted to validation data. On a simple example model the method was successfully used to isolate a poorly fitted equation. This is accomplished by introducing faults to the equations and generating residuals, based on sets of over-determined equations. As a measure of the modelling error the estimation error of the simulated residuals is used, which are weighted together depending on the fault properties of the residuals.

Acknowledgments

Firstly I would like to show my gratitude to all my colleagues at Saab. A special thanks goes to my supervisor Ylva Nilsson and Ingela Lind for your guidance.

I would also like to extend my thanks to my examiner Erik Frisk and supervisor Viktor Leek at Linköping university for all the help and encouragement during this time. I would like to thank Dag Brück for taking the time to answer my questions concerning Dymola, and for taking interest in my work.

Furthermore I would like to extend my thanks to my family and friends. Especially Sandy, since without your support, I would never have made it this far. Lastly I would like to thank you Olle, for supporting me in all my endeavours and for your unconditional love.

Linköping, April 2017
Karin Lockowandt

Contents

Notation **ix**

1 Introduction **1**
1.1 Objectives and Limitations 1
1.2 Contributions 2
1.3 Thesis Outline 2

2 Background **3**
2.1 Model of the Secondary Environmental Control System 4
2.1.1 Model Description 4

3 Theoretical Background on Fault Diagnosis **7**
3.1 Structural Analysis and Fault Diagnosis 7
3.1.1 Dulmage-Mendelsohn Decomposition 8
3.1.2 Minimally Structural Overdetermined Sets 9
3.1.3 Detectability and Isolability 9
3.1.4 Residual Generation and Selection 9
3.1.5 Fault Sensitivity Matrix 10
3.2 Matlab Fault Diagnosis Toolbox 11

4 Introduction to Modelica **13**
4.1 Modelica Environments 13
4.2 Reusage of Code 14
4.2.1 Connectors 14
4.2.2 Inheritance 14
4.3 Extracting Models from Dymola 14
4.4 Why XML? 15
4.4.1 XML and ModelicaXML 15

5 Parsing of Modelica Models **17**
5.1 XML to Matlab Parser 17
5.1.1 Limitations 17
5.1.2 Structure and Content of Dymola-based ModelicaXML . . 18

5.1.3 Symbols . . . 19
5.1.4 Equations . . . 20
5.1.5 Expressions . . . 23
5.2 Conclusion and Discussion . . . 24

6 Model Validation Method **27**
6.1 Introduction of an Example . . . 27
6.1.1 High Fidelity Model . . . 27
6.1.2 Low Fidelity Model . . . 29
6.2 Introducing Faults . . . 29
6.3 Residual Generation and Selection . . . 31
6.3.1 Diagnostic Properties . . . 31
6.3.2 Set of MSOs . . . 31
6.4 Model Validation Measures . . . 33
6.5 Model Validation Results of Example Model . . . 33
6.5.1 Estimation Phase . . . 33
6.5.2 Validation Phase . . . 35
6.6 Conclusion And Discussion . . . 37

7 Properties of the SECS Model **39**
7.1 Model Information . . . 39
7.2 Structural Information . . . 40
7.3 Fault Diagnostic Properties . . . 43

8 Summary and Future Work **45**
8.1 Future Work . . . 45
8.1.1 Parsing of Modelica Models . . . 45
8.1.2 Model Validation . . . 45

Bibliography **47**

Notation

Fault Diagnosis Notation

Abbreviation	Meaning
$\boldsymbol{X}$	The set of all unknown variables
$\boldsymbol{Z}$	The set of all known variables
$\boldsymbol{F}$	The set of all fault variables
SO	Set of Overdetermined Equations
MSO	Minimal Set of Overdetermined Equations
FDT	Fault Diagnosis Toolbox
ODE	Ordinary Differential Equation
DAE	Differential Algebraic Equation
FSM	Fault Sensitivity Matrix

Model Notation

Abbreviation	Meaning
ECS	Environmental Control System
SECS	Secondary Environmental Control System
PECS	Primary Environmental Control System
LL	Liquid Loop
HFM	High Fidelity Model
LFM	Low Fidelity Model

Parsing Notation

Abbreviation	Meaning
XML	eXtensible Markup Language
XSD	XML Schema Definition
XMP	XML to Matlab Parser

1 Introduction

Modelling has become an essential tool in most industrial sectors. Possible applications are many and they are mostly used to evaluate not yet built systems or already existing ones. A well fitted model can predict how a system will react under certain premisses without having to perform experiments on the system itself. Simulations in a computer environment is much more cost effective and decreases the risk to influence man and environment negatively.

Models can be used for many purposes besides simulation, one being model based diagnosis, which is the field within which this thesis is conducted. The idea behind model based diagnosis is to use the model as well as measurements to monitor processes in a system. In this thesis the theories and methods originating from this field are used in a new method for model validation. This is achieved by introducing faults to the model equations. Thereafter residuals are generated based on structural properties of the model. These are simulated and evaluated according to the developed method.

There are many tools available for modelling, suited for different applications. Regarding multi-domain modelling, the language Modelica is especially useful since it supports a wide range of domains, for example electrical, mechanics, thermal etc. However, programs built on the Modelica language are mainly used for simulation. By translating models from Modelica to other languages suitable for analysis and calculations, e.g. Matlab, the benefits of both languages can be exploited. The aspect of extracting and translating, or parsing models from Modelica to Matlab is explored in this thesis.

1.1 Objectives and Limitations

The first question this thesis aims to answer is how the parsing from Modelica to Matlab can be done. In the method investigated the model is first translated

to XML by Dymola. Therefore one of the questions that has to be answered is how different structures in Modelica are portrayed in the XML translation. The second question is how different constructions in Modelica are to be interpreted in Matlab. The implementation is based on a set of sample models, and the constructions appearing in those state the limitations of this thesis. This thesis is not meant to cover the entire Modelica language, but serves as a proof of concept and provides a foundation for further development. The third question is if it is possible to determine which parts of a model that have inadequate fidelity, utilising structural methods and fault diagnosis. More specifically how isolation and weighting of the residuals can be achieved and to which extend this can be used as a model validation method. Since the FDT used in Matlab only handles low-index problems this thesis will be limited to only handling those as well.

1.2 Contributions

There is a need to translate models from Modelica to other languages. Previously, automatic extraction of model variables and equations was not possible. This thesis offers an approach on how the model can be extracted automatically on an equation based format. It also contributes with documentation on how Modelica is translated to the XML file used. Furthermore, a novel method on model validation is presented, enabling the user to isolate which equations, describing the model, that are more or less accurate in comparison to measurement data.

1.3 Thesis Outline

In Chapter 2, the background to the thesis is described, including a description of the model, which is part of the set of sample models. The following chapter includes theory on structural analysis, fault diagnosis and a Matlab toolbox based thereon, relevant for the methods elaborated in Chapter 5 and 6. A short introduction to the modelling language Modelica is given in Chapter 4. Chapter 5 describes the conducted analysis, and method developed to extract and translate models from Modelica to Matlab. The foundation of the software described in this chapter was developed by the author, whereas contributions were made by Petter Lannerhed, in the framework of an adjacent master's thesis[12]. His contributions were:

- the handling of array variables and parameters.
- handling faults appearing in more than one equation.
- interpreting For-loops.

An elaboration of the developed method for model validation is presented in Chapter 6, including an evaluation of the method. Chapter 7 depicts the structural properties of the model presented in Chapter 2. Lastly the master's thesis is summarised in Chapter 8, where propositions of future work is given as well.

2

Background

There is a great need for translating models between different programming languages to exploit the features present in various programs. When it comes to modelling multi-domain physical systems, Modelica is one of the more versatile languages[3], but from an analytical point of view Modelica has its limitations. Matlab on the contrary, is a widely used language and tool for analysis within engineering, but is inferior to Modelica for some types of modelling. Therefore it is advantageous to model in Modelica and thereafter or during the modelling process use Matlab for analysis. The Matlab Fault Diagnosis Toolbox (FDT) is especially useful for structural analysis and fault diagnostics. To exploit this method the model has to be translated from Modelica to the FDT. This can be done either manually, which might prove to be very time consuming, or automatically. The automatic way is preferred for two reasons: speed and robustness. Speed is improved since everything is handled by the computer, and robustness is improved since the influence of the human factor is removed. Although the languages share similarities[3], Modelica specialises in modelling and simulation, and Matlab in numerical calculations. Thus the languages completely differ in some areas, making it a non-trivial task to translate models.

When drawing analytical conclusions about properties of a system from a model it is important that the model is well fitted to the system. If the model inaccurately reflects the actual system any conclusions drawn from the model might be void. Therefore an essential part of modelling is model validation[9]. Traditionally, one way of doing this is by comparing how well the output of a simulation fits the measurements from a real life experiment. This gives an indication of how well the model performs in its entirety, but conveys little information of which parts of the model are well-fitted. A novel method, developed using the FDT, is presented in this thesis.

2.1 Model of the Secondary Environmental Control System

To model the air supply and cooling system, i.e the Environmental Control System (ECS) of the air-plane Saab 39 Gripen, Saab uses the modelling language Modelica. The ECS can be divided into a Primary (PECS) system, a Secondary (SECS) system and a Liquid Loop (LL), allowing each sub-system to be modelled separately. For different purposes more or less detailed models of the system is necessary. Therefore a detailed high fidelity (HFM) and a reduced low fidelity model (LFM) was developed. The HFM is used mainly for testing performance requirements, while the LFM is mostly used in simulators. This thesis focuses mainly on the LFM SECS model.

2.1.1 Model Description

In Figure 2.1 the main part of the SECS-model is depicted. Hot and pressurised air is taken from the engine and cooled down. The cold air is used to cool down the liquid in the LL, which in turn is used to cool equipment, not depicted in the schematics. Three types of sensors are used to monitor the system, namely ten temperature sensors, four pressure sensors, and one accelerometer. The high amount of sensors monitoring the system makes the model useful for model-based fault diagnosis.

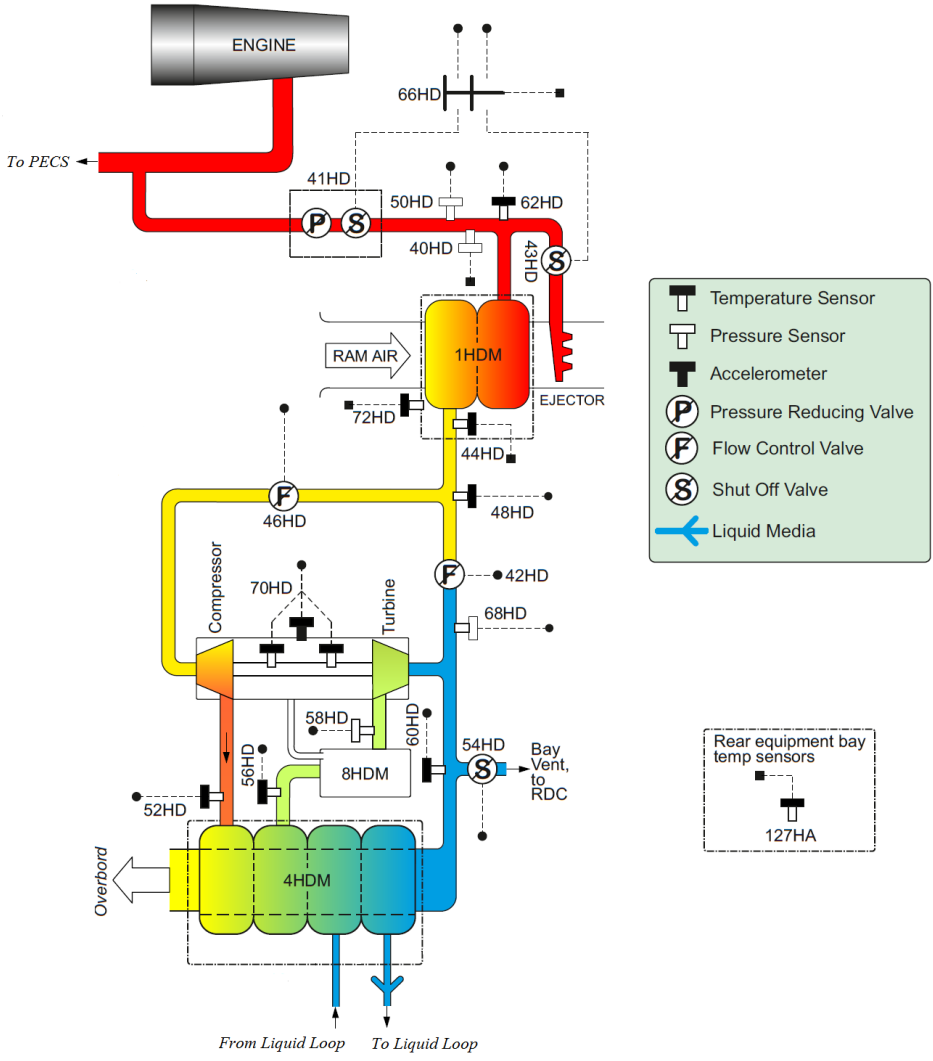

Figure 2.1: *Schematic of the Secondary Environmental Control System.*

3

Theoretical Background on Fault Diagnosis

Model based diagnosis is used to detect faults appearing in a system[13]. A fault refers to a deviation of the expected output of a system. To be able to perform fault diagnosis the system needs to be measured or observed. Besides using the observations as a basis for fault detection, knowledge of the system can be used to enhance diagnostic properties. This can be done by modelling the system and use the information obtained in that model combined with observations to generate a diagnostic statement.

3.1 Structural Analysis and Fault Diagnosis

The structural model of an equation-based model describes which variables are contained in which equation and can be represented by an incidence matrix[5]. E.g. for a model, that contains the variables x_1, x_2, x_3, y_1 and y_2, and is described by the equations

$$e_1 : y_1 = x_1 \tag{3.1}$$
$$e_2 : y_2 = x_2 \tag{3.2}$$
$$e_3 : x_1 = 2x_3, \tag{3.3}$$

the structural information is depicted by Table 3.1.

It is notable that the structural information only considers which variables are contained in the different equations and not in which manner these appear, e.g. the value 2 in 3.3 is not regarded. This means that from a structural perspective the equation $y_1 = x_1$, would yield the same result as e.g. $y_1 = e^{x_1}$. Structural analysis is based on manipulating and analysing the structural information contained in the models. The information obtained can be used to determine how a diagnosis system can be designed.

Table 3.1: *Structural information of the example given by equation 3.1-3.3.*

	x_1	x_2	x_3	y_1	y_2
e_1	x			x	
e_2		x			x
e_3	x		x		

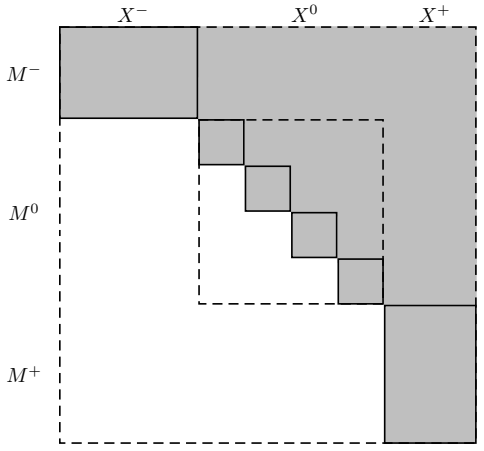

Figure 3.1: *An illustration of a Dulmage-Mendelsohn decomposition, where M^- represents the under-determined part, M^0 the exactly determined, and M^+ the over-determined. Source: [11].*

3.1.1 Dulmage-Mendelsohn Decomposition

The Dulmage-Mendelsohn Decomposition[2], visualised in Figure 3.1, is obtained by reordering the equations in the structural model. The decomposition consists of three parts, an under-determined part M^-, an exactly determined part M^0, and lastly an over-determined part M^+[6][11]. The associated sets of unknown variables are denoted with X. A sub-set becomes over-determined when the subset contains redundancy, which arises if there exists more than one way to determine a variable using only observations[13, pages 26-27]. For example, if a model consists of a variable x_1, a parameter k, the observations y_1 and y_2, which are defined by

$$e_1 : y_1 = x_1$$
$$e_2 : y_2 = kx_1,$$

the variable x_1 can be determined both with e_1 and e_2, i.e. the model contains redundancy. From a diagnostic point of view the over-determined part is the most valuable since it can be monitored.

3.1.2 Minimally Structural Overdetermined Sets

If a set of equations has more equations than unknowns, it is a Structural Over-determined (SO) set. Furthermore, a set is Minimally Structural Over-determined (MSO), if there are no proper subsets that are SO sets[10]. MSO sets are subsets of the model which are testable, i.e. they can be used as a basis for creating residuals.

Since the number of MSO sets, and thereby also the computational complexity, increase exponentially with the increase of redundancy in the model, it is not applicable for models with high redundancy[5].

3.1.3 Detectability and Isolability

Whether a fault is structurally detectable and/or isolable can be determined from the structural information. If a fault is present, and the system behaviour is distinguishable from the fault free case, the fault is detectable. This implies that it is isolable from the fault free case. For a fault to be isolable from other faults, firstly it has to be detectable. A fault f_i is isolable from another fault f_j if f_i, but not f_j is part of an over-determined set. Definition 2 and 3 in [7, page 4] gives a more formal approach.

3.1.4 Residual Generation and Selection

There are many ways to design and generate residuals. Residuals describe the differences between measurements and estimates, calculated from model equations, which in a fault free case should be close to zero. In the faulty case the measurements and estimates should differ, leading to residuals that diverge from zero. In this thesis two approaches are used: the sequential residual generation and differential-algebraic observers. These are well suited for automatic code generation[5], which reduces the time spent on residual design.

Sequential Residual Generation

A sequential residual is based on an SO set, for example an MSO set. The set is split up into two parts, the first being an exactly determined set of equations g_i^1 with regards to the unknown variables x, and the other, g_i^r acts as the residual equation

$$g_i^1(x, z, f) = 0,\ i = 1, ..., n_1$$
$$g_i^r(x, z, f) = 0,\ i = 1, ..., n_r.$$

The exactly determined part g_i^1 is used to calculate the estimate $\hat{x}$, which is used in the residual itself:

$$r = g_i^1(\hat{x}, z, f) = 0.$$

If the set studied has dynamics in it, these have to be handled either by integration or derivation, which leads to larger computational uncertainties than if no dynamic is present. In those cases an observer can result in smaller estimation errors.

Observer Based Residual Generation

A residual implemented as an observer estimates all unknown variables and computes the residual. If the unknown variables are divided into x_1, containing the states, and x_2, containing the algebraic variables, the model can be described by

$$g_i(dx_1, x_1, z, f) = 0,\ i = 1, ..., n$$
$$dx_1 = \frac{d}{dt}x_1,\ i = 1, ..., m.$$

Accordingly, the observer can be described by

$$\begin{aligned} \dot{\hat{x}}_1 &= g_1(\hat{x}_1, \hat{x}_2, z) + K(\hat{x}, z)g_r(\hat{x}_1, \hat{x}_2, z) \\ 0 &= g_2(\hat{x}_1, \hat{x}_2, z) \\ r &= g_r(\hat{x}_1, \hat{x}_2, z), \end{aligned}$$

where K is the feedback gain. If the observer is on DAE-form an ODE solver can be used in Matlab. It should be noted that the ODE solver only handles low-index problems, i.e. when the model equations only have to be differentiated once to obtain the DAE-form. Therefore only low-index problems are studied in this thesis.

One of the big disadvantages with observers is that K has to be determined, which seldom is a straight forward task. A proposal on how K can be determined, for model validation purposes, is given in Chapter 6.

3.1.5 Fault Sensitivity Matrix

A set of equations is sensitive to a fault f_i, if it is present in one, or more, of the equations of the set. A residual, based on a given set of equations, will have the same fault sensitivity properties as the set itself. This information is useful for isolation purposes, and can be expressed as a fault sensitivity matrix (FSM). E.g. if the fault variables f_1, f_2 and f_3 are added to the equations 3.1-3.3 the following model is obtained:

$$e_1 : y_1 = x_1 + f_1 \tag{3.4}$$
$$e_2 : y_2 = x_2 + f_2 \tag{3.5}$$
$$e_3 : x_1 = 2x_3 + f_3. \tag{3.6}$$

If the set $\{e_1,e_3\}$ is used to form a residual r_1, and a residual r_2 is based on e_2, the fault sensitivity is described by Table 3.2, and the FSM is given by:

Table 3.2: *Fault sensitivity of the example given by equation 3.4-3.6.*

	f_1	f_2	f_3
r_1	x		x
r_2		x	

$$\text{FSM} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

3.2 Matlab Fault Diagnosis Toolbox

The development of a Matlab toolbox, called Fault Diagnosis Toolbox (FDT)[5], based on the theories in this section, is an ongoing process, taking place at Linköping university. This toolbox is very useful when performing structural analysis and fault diagnosis on a system, as demonstrated in Chapter 6 and 7. When using the FDT, the system to be analysed has to be defined as a model object according to a given syntax. E.g. can the model described by equation 3.4-3.6, in Section 3.1.5, be defined as:

```
modelDef.type = 'Symbolic';
modelDef.x = {'x1','x2','x3'};
modelDef.f = {'f1','f2','f3'};
modelDef.z = {'y1','y2'};

syms(modelDef.x{:});
syms(modelDef.f{:});
syms(modelDef.z{:});

modelDef.rels = {y1==x1, y2==x2, x1==2*x3};

model = DiagnosisModel( modelDef );
```

The syntax is described in detail in [5].

4

Introduction to Modelica

Modelica is an object-oriented equation-based programming language used to model and simulate natural or artificial systems[8]. It is a-causal, meaning that the user defines the physical characteristics of a system with equations. These are sorted by the compiler, making the model solvable, which in turn means that it can be simulated. The code generation is to a great extent done automatically and the modelling effort is thereby heavily reduced[4]. The basic idea behind Modelica is that a model consists of components, which can be modelled separately and connected to form a model of the entire system. Models, as well as components, functions etc., are declared as classes.

4.1 Modelica Environments

Models can be represented visually as combined icons in the diagram view, or as Modelica code in the text view. Although the diagram is a helpful tool for the programmer, the Modelica text view is of greater interest for this thesis since it reveals the equations and variables of the model. An example model named `Example` in Modelica code can be defined as:

```
model Example
 Real x;
 Real y;
 parameter Real k=1;
equation
 x=sqrt(4)*k;
 y=2*x+4;
end Example;
```

All variables, constants and parameters must be declared. In the example case these are named `x` and `k` and are of type `Real`. The body, which defines the model,

can contain equations, algorithms and initial equations, which are declared in the respective section. In the example above an `equation` is demonstrated. Usually most equations are contained in this section, but equations that are only valid at initialisation are declared under `initial equation`. In the `algorithm` section the assignment statements are imperative, which means that the assignments change the program's states. This can be compared to the declarative statements in the `equation` section, which corresponds to mathematical logic. Instead of an equality sign an assignment sign (:=) is used in the `algorithm` section. Like many other programming languages Modelica supports the use of `if`, `for`, `while` etc. and a variety of different functions. All statements must be terminated with a semicolon, as shown in the example.

4.2 Reusage of Code

Dymola includes Modelica-libraries, which enables the usage of pre-defined components in the modelling process. This enables the reuse of code which saves the user a lot of time and effort. New components are easily created to fit the need of the programmer, by adding and connecting already existing ones. Modelica invites to design models with a nested structure, i.e. using components within components, which leads to models with a hierarchical structure.

4.2.1 Connectors

To connect components Modelica uses connectors. Variables that are specified as inputs/outputs from a component can be connected if the types, e.g. `Real` or `Boolean`, match. In the Modelica text view the command

```
equation
 connect(a,b);
```

is used, which corresponds to $a = b$.

4.2.2 Inheritance

Modelica offers the possibility of inheritance, by defining a model as a partial model. The model inheriting the properties of another model is declared with

```
model child
 extends parent;
end child;
```

4.3 Extracting Models from Dymola

To be able to translate the model from Modelica to Matlab, it has to be extracted from the Dymola environment first. To facilitate the extraction, the source code

of the Modelica-model can be represented by ModelicaXML[14]. In this thesis the ModelicaXML refers to the eXtensible Markup Language (XML) standard of Dymola 2017 FD01.

4.4 Why XML?

After the model has been extracted to an XML-file it can be used to translate the model into other languages. The software program introduced in Chapter 5 enables this. One of the main advantages with this method is that the entire model is summarised in one file, including possible inheritances. The output of the parser is an m-file describing the model according to the FDT-standard in section 3.2 on which diagnosis can be performed. This work flow is depicted in Figure 4.1.

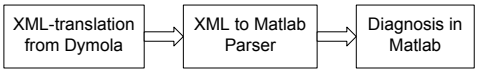

Figure 4.1: *Work flow of the implemented method, where the second step is described in Chapter 5 and the third step in Chapter 6.*

4.4.1 XML and ModelicaXML

XML is a standardised way for handling data. One of the main characteristics of XML is that it distinguishes between the context and structure of the data. The structure is defined by token elements, which appear as `<x> content </x>`, where `x` is the name of the tag and *content* the content contained in the token element. An empty element tag, or a token element without content, may be written `<x/>` and can contain attributes, which are incorporated by writing `<x attribute="value"/>`. For example, the equation $y = x + 1$ would be expressed as:

```
<SimpleEquation>
  <LHS>
    <ComponentReference>
      <Reference
        instanceName="y"/>
    </ComponentReference>
  </LHS>
  <RHS>
    <Binary Operator="+">
      <Left>
        <ComponentReference>
          <Reference
            instanceName="x"/>
        </ComponentReference>
      </Left>
      <Right>
```

```
        <Literal Value="1"/>
      </Right>
    </Binary>
  </RHS>
</SimpleEquation>
```

in ModelicaXML. The token names appearing in this example are elaborated in Chapter 5.

Depending on the application the syntax varies, but regardless two structural requirements must be maintained. Firstly every document must have a root element and secondly all tags must be properly nested. The requirements results in a tree structure, which means that the model structure in Modelica is kept after the transformation to ModelicaXML, which is written in XML. For the interested reader, more information on how the ModelicaXML-file is created can be found in [15]. The content and structure that an XML-file can have is defined by the respective XML Schema Definition (XSD), which can also be used to visualise the structure that may appear in a certain XML-file.

5

Parsing of Modelica Models

There are advantages of extracting models from Dymola to other platforms, such as Matlab. Instead of manually translating a model from Modelica to Matlab, which depending on the size of the model might be a tremendous task, it could be translated automatically. This thesis will explore one option on how the automatic translation can be achieved.

5.1 XML to Matlab Parser

In order to extract and translate the information contained in the ModelicaXML file software had to be developed. Therefore a program called XML to Matlab Parser (XMP) was implemented. The choice of programming language fell upon Python, which is suitable for a number of reasons. Python is foremost easy to use and a widely used language. The availability of the open-source XML-parser `xml.etree.ElementTree` (ET)[1] contributed as well, since it simplified the coding significantly. The XMP uses the ET to extract the tree structure as a Python object, on which operations is performed, to extract the information contained within each element.

5.1.1 Limitations

In the scope of this thesis only models which are translatable to valid ModelicaXML-code were considered. ModelicaXML-formatting is not standardised within the Modelica community and different programs built upon the Modelica language will have different standards. Therefore the structure will differ between different software applications of Modelica and for some might not be accessible at all. That is, only models translated by Dymola were taken into consideration. Since the time was limited not all cases that might occur in the Dymola-specific Mod-

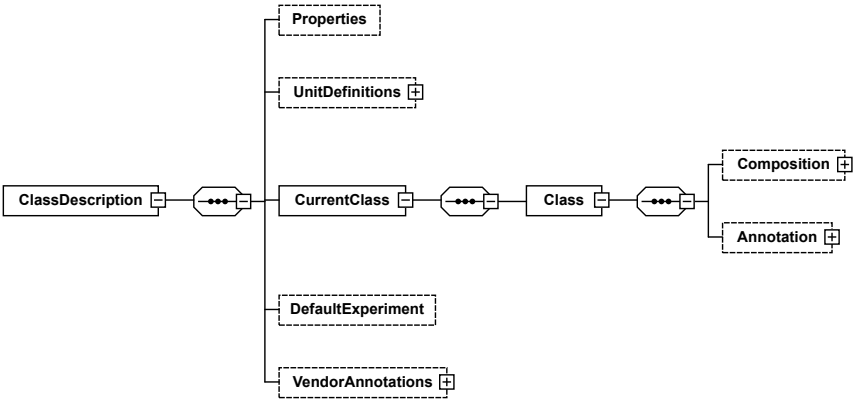

Figure 5.1: *The root element of ModelicaXML.*

elicaXML standard were taken into consideration. Instead the XMP was based on the SECS-model as an initial guideline and further elaborated with a model of the cabin control pressure in Saab 39 Gripen. The limitations of the XMP will be mentioned throughout this chapter.

5.1.2 Structure and Content of Dymola-based ModelicaXML

The ET-parser creates a Python object called `ElementTree`. The tree consists of elements and can be visualised, as in Figure 5.1, utilising the XSD file described in Section 4.4.1. An element is depicted as a rectangle containing its tag. An optional element, i.e an element not necessarily needed to fulfil the ModelicaXML-standard, is visualised as a rectangle with a dashed line. A plus sign inside a square attached to the element means that the tree can be expanded further, i.e that the element can have child-elements. Throughout this chapter the element tag is written in **bold** and optional text strings and attributes shown in *italic*.

In Figure 5.1 the root element, called **ClassDescription**, is depicted. Most of the information extracted by the XMP is contained in **Composition** which along with its children is depicted in Figure 5.2. **Declaration** contains all symbols and components declared in the current component, i.e the **Class** element currently regarded, as shown in the bottom part of Figure 5.3. In each of these sub-components the structure of the **Class** element is repeated, recreating the same nested structure as Modelica. As for the equations declared in the current component they are found in the element **Body**.

When defining models in the FDT the tree-structure is completely flattened, but the component hierarchy is kept in the names of variables to ensure unique variable-names and to enable the conversion from Matlab back to Modelica. This is achieved by always keeping track of which components have been entered and storing their names in a list, which is flattened before printing.

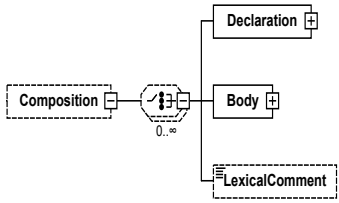

Figure 5.2: The element ***Composition*** *with its two child elements.*

5.1.3 Symbols

In ModelicaXML variables, parameters and constants are all declared with **Variable** as seen in Figure 5.3. What Dymola regards as variables will be referred to as symbols to avoid confusion. Parameters, constants, continuous and discrete variables are the type of symbols that may appear in a ModelicaXML file.

Variables, Parameters and Constants

To distinguish between parameters and variables, the variable key *variability* is used. This key is not present if the variable is continuous. For parameters the key equals *parameter*, for discrete variables *discrete* and for constants *constant*. However the latter is ignored since constants are inserted into the equations before the translation to ModelicaXML takes place. If the symbol is a parameter it will have an assigned expression which is found in **BindingEquation** and/or **StartExpression**. If the symbol is non-scalar the dimension is to be found in **Dimension**. Variables, independent of their causality are considered unknown and will be added to the set of unknown variables $\boldsymbol{X}$, unless they already are present in the set of known variables $\boldsymbol{Z}$.

Known Variables

In the field of fault diagnosis it is important to know, or to decide, which variables are considered known. Typically these are sensor measurements or states which are potentially measurable, and inputs. Which the known variables are has to be determined manually, but is often a straight forward task. These are contained in the set of known variables $\boldsymbol{Z}$.

Fault Variables

Fault variables need to be distinguished from known and unknown variables. If faults are implemented in the Modelica model, this can be achieved by using a name convention. If no name convention is used, or if it is inconsistently used, the faults can be sorted out from the rest manually afterwards and placed in the set of fault variables $\boldsymbol{F}$. If no faults are implemented in Modelica or if the user chooses to add more faults this has to be done manually.

The FDT requires that a fault variable only occurs in one equation, this based on the assumptions made in [7, page 5]. If faults are added manually, after the

parsing step, the user has to take this into consideration. But if fault variables are introduced directly into the model in Modelica, the XMP will handle this by considering the original fault f_{org} as an unknown variable. To avoid information loss a new equation $f_{org} = f_{new}$ is added, where f_{new} will be added to the set of fault variables instead.

5.1.4 Equations

Equations are contained in the element **Body**, containing the attribute *bodykind*, which states if the children are of type *equation*, *initial* or *algorithm*. In Modelica these are declared under the equation, initial equation or algorithm section respectively. In the FDT no differentiation is made between equations and algorithms and therefore both will be referred to as equations. In the case of initial equations these are valid only at the initialisation stage, and do not hold true at other time instances. Therefore the initial equations are disregarded, otherwise they might add false information. The element **Body** can contain different types of equations, which are displayed in Figure 5.4. All body types are implemented except `While`. However the type **MultiReturningFunction** has been deemed irrelevant, for the current application of the XMP and is disregarded. The reason is that they do not contribute with any structural information. In the models regarded only `assert` statements are contained in **MultiReturningFunction**s, which are used to verify that the specified conditions are met at the simulation stage. These are typically used to ensure that a model operates within its limits of validity, e.g. that an absolute pressure never assumes negative values. Common equations are contained in a **SimpleEquation** element and consist of a left hand and right hand side marked with **LHS** and **RHS** respectively, as seen in Figure 5.4. Each side can contain any **ExpressionType** elaborated below in Section 5.1.5. Although often the left hand side only contains one variable, i.e a **ComponentReference**.

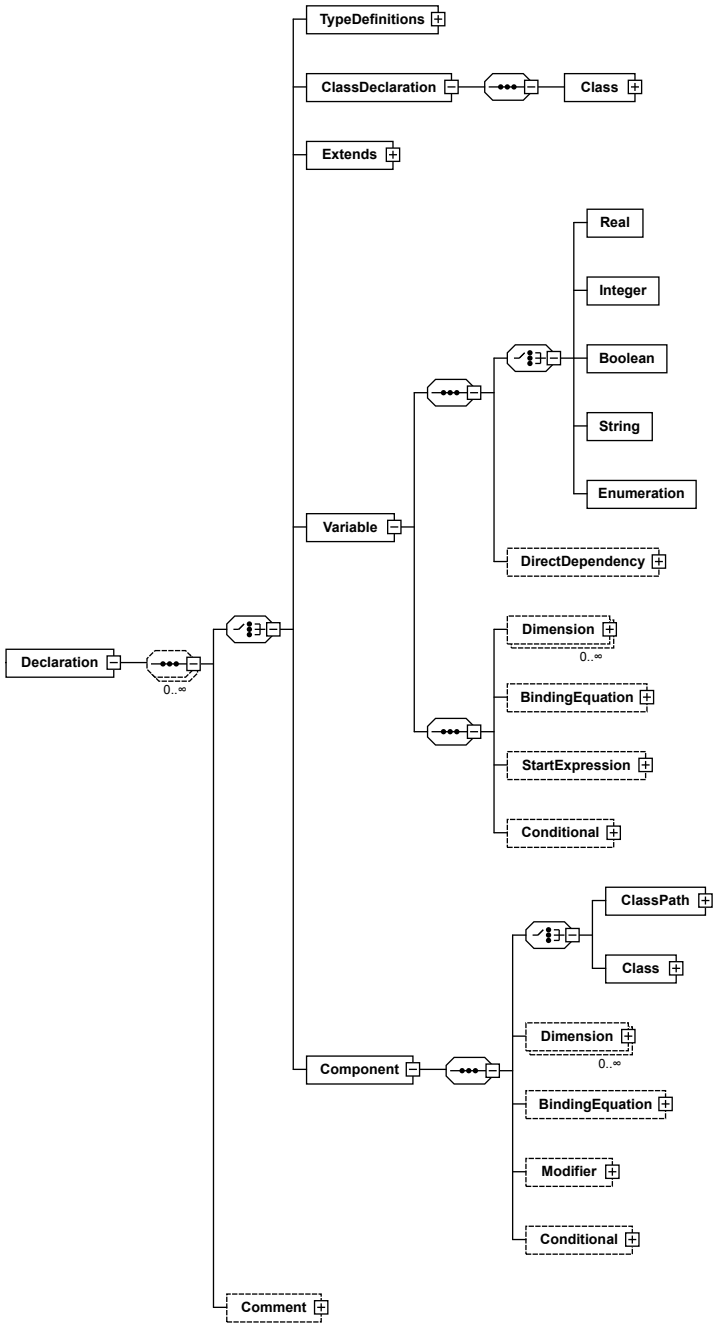

Figure 5.3: *The element* ***Declaration*** *and its children with the child elements* ***Variable*** *and* ***Component*** *expanded.*

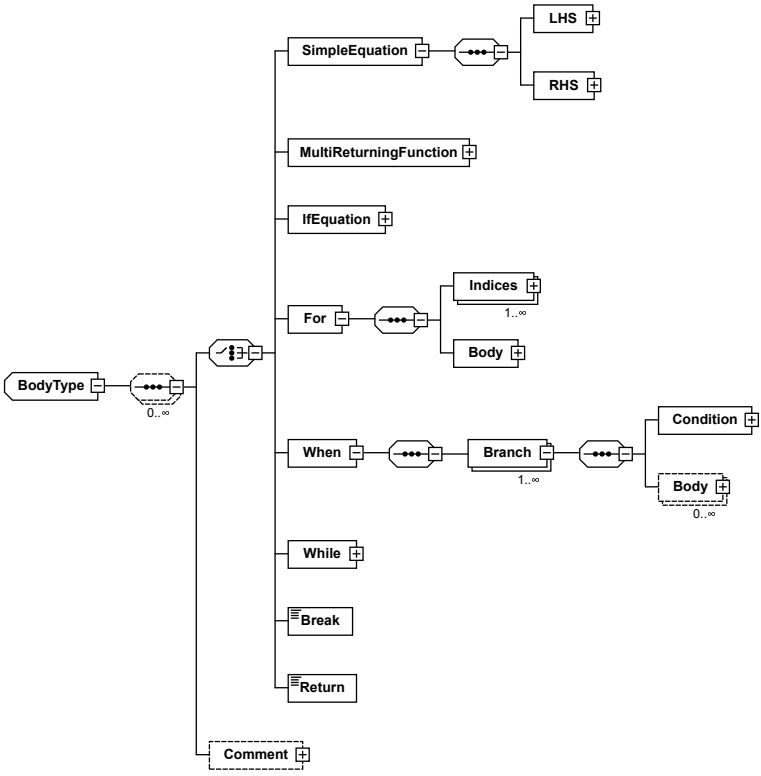

Figure 5.4: *All Body types that can be present in ModelicaXML.*

5.1.5 Expressions

Expressions appear when parameters are assigned values and in equations. The expression types

- `Literal`
- `ComponentReference`
- `Unary`
- `Binary`
- `IfThenElse`
- `Range`
- `FunctionCall`
- `ForExpression`

are the types that can occur in ModelicaXML, which are all handled by the XMP.

Operators

An operator is referred to as **Binary** or **Unary**. The operands, or operand if it is a unary, are below the operator in hierarchy and can be any of the above mentioned expressions.

Function

Functions are tagged with **FunctionCall** and are handled differently depending on the function. The name of the function is to be found in the *ClassName* attribute of the child-element **FunctionPath**. The input arguments are contained in the element **Arguments** which has one or more children named **Argument** containing the expression.

Functions that are called in the same manner in Modelica as Matlab, e.g. trigonometric functions, are extracted as they are. Other functions, that do not appear in the same manner in Matlab, must be treated differently. Derivatives holds a special position amongst these since they are regarded as separate variables in the structural analysis. If a variable is differentiated in an equation the derivative is added to the list of variables and a new equation is added in the printing process. Other functions that do not have a Matlab equal are regarded as external functions and have to be defined in a separate m-file. An example is the `spliceFunction`, which performs a spline interpolation of two functions. This has to be done manually, but the functions found in the standard library are often well-documented within Dymola. Some Modelica-functions, for example `smooth` or `noEvent` do not describe the model but are used to facilitate the modelling process and are disregarded by the XMP.

If- and When-Statements

Besides functions if- and when-statements are also defined by external functions. If-statements are tagged with **IfThenElse**. They consist of a condition, a then-branch and an else-branch. If the condition is true, the if-branch is executed, otherwise the else-branch is executed. If-statements are predefined by a shell function in Matlab which takes the condition and the then- and else-branches as arguments. Note that the **ExpressionType** described here should not be confused with the **ifEquation** from Figure 5.4. Although they serve more or less the same purpose they are expressed differently in ModelicaXML.

When-statements have the tag **When** and are of the type **Body**. They are therefore extracted in another fashion than if-statements, but are implemented in Matlab in a similar manner and will therefore be mentioned in this section. Each when-statement is declared with its own condition and only one branch which is executed if the conditions holds. Currently only when-statements where a single variable is assigned inside the statement is supported. The when-statements are bundled together as ifelse-statements without an else-statement in Matlab. The condition of each when-statement constitute the condition of a ifelse-statement except the first which is translated to the if-statement in the ifelse block. The branches inside the when-statements are translated to the branches of the ifelse-statement of the respective conditions.

References to Symbols

Symbols are declared inside **ComponentReference** which in turn contains one or more **Reference** instances depending on if the symbol is declared locally or outside the current component. As mentioned before the hierarchy is kept track of and is compared to the references to avoid printing component names twice or printing a false hierarchy in the final symbol string. The latter can occur if a symbol is referenced to inside a component but declared outside of it. Components, which are referenced to deeper down in the hierarchy, can be defined as inner/outer. They are declared both at the higher level as `inner`, and at the lower level as `outer`. An example for when the component typically is declared as inner/outer is a component that contributes with information of the environmental temperature and pressure of a system, or a component describing physical fields, e.g. an electrical field. If this is the case the symbols contained in these types of components will have different hierarchical structures depending on where they are declared, but they all refer to the same component. Such types of components do not have an attribute labelling them as inner/outer and the user therefore has to specify if such components are present to achieve the correct hierarchy.

5.2 Conclusion and Discussion

One of the main questions this thesis aims to answer is how Modelica models are portrayed in ModelicaXML. In this chapter the structure and content of ModelicaXML is presented. Implementation-wise the extraction has been solved mainly

with for-loops iterating over the tree-structure. Since the structure is repeated within components, recursive methods are used as well. Regarding the question on how Modelica should be translated to Matlab, the required syntax of the FDT is used as a base. Most equations in Modelica can be translated as is, since the languages resemble each other, but some paraphrases where made to fit the FDT, e.g. are derivatives handled differently.

Overall the XMP covers much of the Modelica language, but is not complete. To translate some of the remaining features, for example while-loops and universal when-statements, primarily more time is required. Some other parts of Modelica may offer greater challenges, e.g. translating all external functions, that can appear in Modelica, into Matlab. Depending on the desired level of autonomy, the implementation increases or decreases in complexity. Some limits of automation have already been detected, as the user for example has to define known variables, and inner/outer components. It is likely that more cases may appear. Furthermore, some solutions might work for the models regarded but perhaps not for other models, which were not explored. Additionally, the Modelica Association, responsible for setting standards in the Modelica community, needs to standardise the ModelicaXML-format, in order for further development to be justified. Still, the work presented in this thesis shows that it is indeed possible to extract and translate Modelica models into other languages.

6

Model Validation Method

One of the main aims of this master's thesis is to see if it is possible to validate models against data originating either from another model, or from physical sensors, utilising fault diagnosis methods. Model validation as such is not a novel subject, however a new method to accomplish this is presented here. Compared to more traditional model validation methods, the main aim of this method is to determine specifically for which equations the estimation error is big. For a model to be suitable for this method it has to be equation based.

6.1 Introduction of an Example

For the sake of understanding, the theory will be illustrated using an example, namely a coupled two-tank system illustrated in Figure 6.1. To illustrate model inaccuracies two models of the system are presented. The first, more detailed model, which hereafter will be referred to as the HFM, is considered to more accurately describe the real two tank system than a reduced LFM.

6.1.1 High Fidelity Model

The HFM is based on the physical relation of Bernoulli's principle, which relates the speed of flow of the water flowing out of the tank to the water level. Bernoulli's principle states that

$$v(t) = \sqrt{2gh(t)}, \tag{6.1}$$

where h denotes the water level, v the water flow speed and g gravity. Furthermore the relation between the outflow $q(t)$ and the water flow speed is given by

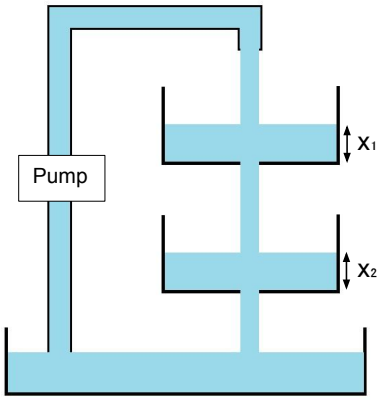

Figure 6.1: *Schematics of the coupled two-tank system .*

$$q(t) = av(t), \tag{6.2}$$

where a denotes the area of the hole through which the water flows out of the tank. The volume of the water contained in the tank is expressed by $Ah(t)$ where A is the cross sectional area of the tank. The rate of which the volume changes over time, i.e the time derivative, is

$$A\dot{h}(t) = u(t) - q(t), \tag{6.3}$$

i.e the flow into the tank ($u(t)$) subtracted by the flow out of the tank. Equation 6.1-6.4 gives an explicit expression of the water level in the tank:

$$\dot{h}(t) = \frac{1}{A}u(t) - \frac{a\sqrt{2g}}{A}\sqrt{h(t)}. \tag{6.4}$$

To model the two-tank system in Figure 6.1 two states are needed. The water level of tank one and two are described by x_1 and x_2 respectively. In turn the derivative of the states and thereby the model of the entire system is given by

$$\dot{x}_1(t) = \frac{1}{A_1}u(t) - \frac{a_1\sqrt{2g}}{A_1}\sqrt{x_1(t)} \tag{6.5}$$

$$\dot{x}_2(t) = \frac{a_1\sqrt{2g}}{A_1}\sqrt{x_1(t)} - \frac{a_2\sqrt{2g}}{A_2}\sqrt{x_2(t)}. \tag{6.6}$$

Equation 6.5 and 6.6 are the sub-models describing the dynamics of tank one and two respectively. For diagnosis purposes sensors have to be modelled as

well. Since the HFM is used to collect data every variable is measurable, but to imitate a reasonable system, four sensors are chosen. The four sensors measure the following:

Table 6.1: *Initial sensors in the two-tank system.*

y_1	Water level in tank 1.
y_2	Water level in tank 2.
y_3	Water flow between tank 1 and 2.
y_4	Water flow out of tank 2.

6.1.2 Low Fidelity Model

An implementation in Modelica of the model above will provide data which can be used for validating the LFM, hence called validation data from here on. In order to test out the thesis mentioned in the beginning of this chapter a LFM of the same system is needed. The simplification of the HFM can be done in a number of ways. As mentioned earlier Equation 6.5 and 6.6 can be regarded as two sub-models of the system. By replacing one of these, for example Equation 6.6, with a first order linear equation, which is chosen arbitrary, a simplified model is obtained. The other equations remain the same and Equation 6.6 is replaced by

$$\dot{x}_2(t) = \frac{kx_1(t) - x_2(t)}{T}, \tag{6.7}$$

where k and T are tuning parameters. A simulated step response, depicted in Figure 6.2, shows that the LFM, with some tuning, is a very good approximation of the HFM, at least at the given operating point. In Section 6.5, the LFM used is not as well tuned, since a more pronounced deviation from the HFM is desirable, for the purpose of simulating model errors.

6.2 Introducing Faults

The faults introduced for model validation purposes do not describe physical faults that can arise in the system as per usual in fault diagnostics, but rather form primary candidates for model uncertainty. The faults are implemented as additive and will describe different kinds of faults that can possibly give rise to model uncertainties, for example inaccurate parametrisation or badly chosen model equations. To determine which model equations are inaccurate, faults are added to some of the equations. They should preferably not be added to every equation, but should only be added to equations that are suspected of contributing to model uncertainties. Adding faults to every equation may lead to unnecessary calculative complexity and decreased fault isolation properties. In the case of the LFM it is assumed that equation e_1 (6.8) and e_2 (6.9) describing the dynamics of the system are more likely to contribute with model uncertainties. Suppose

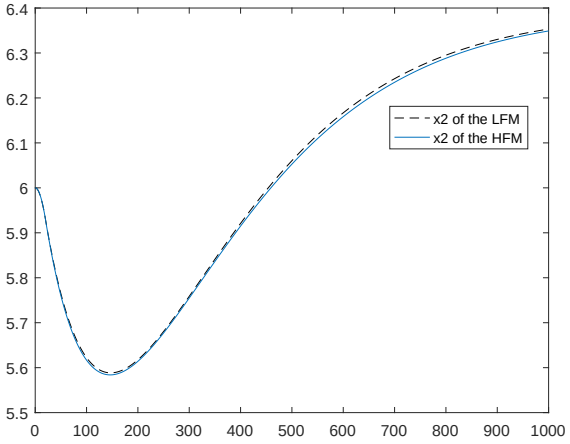

Figure 6.2: *The step response of x_2 of the HFM depicted in a solid line, and x_2 from the well tuned LFM presented in a dashed line.*

that equation e_5 (6.13) also is uncertain. With faults added to the mentioned equations the example model is thus described by

$$e_1 : \dot{x}_1 = \frac{1}{A_1} u - \frac{a_1\sqrt{2g}}{A_1}\sqrt{x_1} + f_1 \tag{6.8}$$

$$e_2 : \dot{x}_2 = \frac{kx_1 - x_2}{T} + f_2 \tag{6.9}$$

$$e_3 : y_1 = x_1 \tag{6.10}$$

$$e_4 : y_2 = x_2 \tag{6.11}$$

$$e_5 : y_3 = a_1\sqrt{2gx_1} + f_3 \tag{6.12}$$

$$\tag{6.13}$$

$$e_6 : y_4 = a_2\sqrt{2gx_2} \tag{6.14}$$

$$e_7 : \dot{x}_1 = \frac{d}{dt} x_1 \tag{6.15}$$

$$e_8 : \dot{x}_2 = \frac{d}{dt} x_2, \tag{6.16}$$

where the time dependency has been omitted for a shorter notation. Equations 6.15-6.16 might seem redundant, but they are necessary for the FDT to determine the connection between the derivative and its primitive.

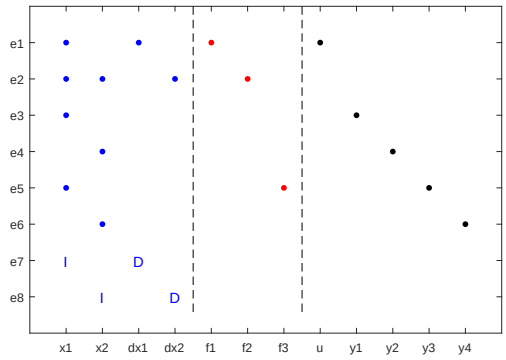

Figure 6.3: *Structural information of the example model, with unknown variables to the left, faults in the middle and known variables to the right.*

6.3 Residual Generation and Selection

When the model has been defined according to the FDT the model structure, and diagnostic properties, which will determine how the residuals shall be formed, can be examined. Based on the structural information MSO-sets can be deducted and used for residual generation.

6.3.1 Diagnostic Properties

The model structure will give a good indication on how the diagnosis system can be designed. For the example used in this chapter the model structure is depicted in Figure 6.3.

Besides examining the structural information, the Dulmage-Mendelsohn decomposition, plotted in Figure 6.4, gives an informative picture of the diagnostic properties. The whole model is contained in the over-determined part of the decomposition, which means that all equations can be used as a base for MSO-sets.

The fault detection and isolation properties, closer described in Section 3.1.3 are also relevant when designing residuals. All three faults appear in different MSO sets each, making them isolable from one another, which might not always be the case. To achieve higher isolability the toolbox supports the placements of new sensors, a subject not examined in this thesis.

6.3.2 Set of MSOs

There are ten possible MSO-sets suggested by the FDT, of which three are needed to achieve full isolability. These three are picked out using the test selection method of the FDT. The first residual, r_1, contains no dynamics but is purely algebraic and is made up of the set $\{e_3,e_5\}$, where e_5 contains the fault variable

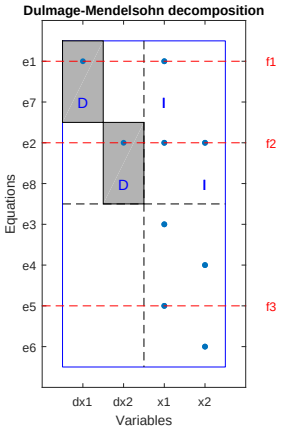

Figure 6.4: *Dulmage-Mendelsohn decomposition of the example model.*

f_3. Thus r_1 is sensitive to the fault f_3. The sequential residual generation method results in

$$\hat{x}_1 = y_1$$
$$r_1 = y_3 - a_1\sqrt{2g\hat{x}_1},$$

where the first equation is used to estimate the state x_1 and the second is used for the residual calculation. The two remaining MSO-sets have dynamic properties and the observer method is therefore used. One of these MSO-sets is made up of the equations $\{e_2,e_3,e_6,e_8\}$, which will be sensitive to f_2, and the observer based thereon is described by

$$\hat{x}_1 = y_1 \tag{6.17}$$
$$\dot{\hat{x}}_2 = \frac{d}{dt}\hat{x}_2 \tag{6.18}$$
$$r_2 = y_4 - a_2\sqrt{2g\hat{x}_2} \tag{6.19}$$
$$\dot{\hat{x}}_2 = \frac{k\hat{x}_1 - \hat{x}_2}{T} + Kr_2, \tag{6.20}$$

where K is the gain controlling the feedback in the observer. The third residual, r_3, is based on the MSO set $\{e_1,e_3,e_7\}$, and is sensitive to fault f_1. The fault sensitivity of the three residuals is depicted by the following FSM:

$$\text{FSM}_{\text{LFM}} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

6.4 Model Validation Measures

The next step is to determine which residuals are significantly larger than zero and how much larger than zero they are. To isolate which faults give rise to a deviation from zero in the residuals, the FSM_{LFM} is useful.

To determine which faults are significantly larger than others the magnitude of the error has to be determined. For this purpose the prediction error, given by

$$V = \frac{1}{N}\sum_{n=1}^{N} \|y - \hat{y}\| \tag{6.21}$$

is a good measure[13]. If the residuals are sensitive to one fault each the prediction error of the corresponding residual will provide the measure of the fault, but if one or more residuals are sensitive to more than one fault the measure has to be weighted. The solution proposed here is to take the product of all prediction errors of the residuals sensitive to each fault, i.e

$$M_{f_j} = \prod_{V \in V_{FS}} V_{r_i}, \tag{6.22}$$

where V_{FS} is the set of prediction error of the residuals with the corresponding fault sensitivity to fault f_j. In practice this can be achieved by multiplying each row of the FSM, corresponding to the residuals, with the respective prediction error V_{r_i} and then multiplying each non-zero element of each column.

6.5 Model Validation Results of Example Model

When the MSO sets have been selected and the residuals created these need to be evaluated. By evaluating the residuals against data originating from the model itself the behaviour in the fault-free case can be studied. If the residuals are far from zero, either the states are hard to estimate, or the model is not translated well enough. This is referred to as the estimation phase, where the gain K of the observers should be determined as well. The validation phase follows after the estimation phase, where the model is validated against another model or a real life system.

6.5.1 Estimation Phase

To determine how well the residuals perform in the fault free case the model is validated against simulation data originating from the model itself. In the plots depicted in Figure 6.5, the noise free data originates from the Modelica-model of the system, where a step occurring at $20s$ is used as input. For the observers the gain is set to $K = 0$ initially, which means that there is no feedback. This will reflect the residuals ability to estimate the states blindly, i.e. without the use of feedback from the measurements. The estimation error V_{r_1} of the first residual r_1 is very small compared to the others. This follows from the fact that

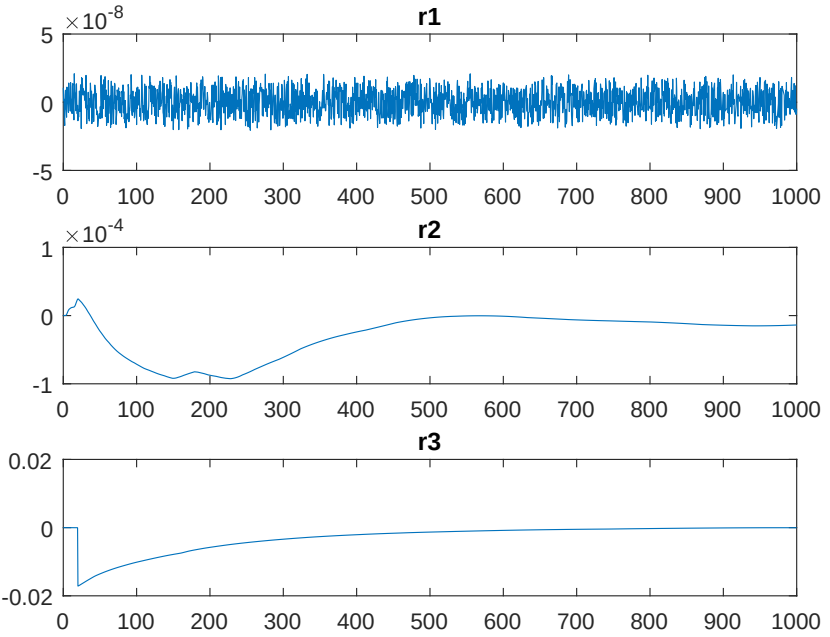

Figure 6.5: *Residuals of the LFM example model validated against data from the same model, with zero gain in the observers r_2 and r_3.*

r_1 is purely algebraic. Therefore the estimation error of r_1 is the error originating from computational errors in Matlab. In the observer-based residuals, especially in r_3, but also noticeable in r_2 for a higher value of K, a peak appears at the time of the step.

In Figure 6.6 the states x_1 and x_2 are plotted in a solid line and their respective estimate in a dashed line. The state estimate is taken from the observer described by Equation 6.17-6.20. As seen in Figure 6.6, the state estimation is very good and the difference, which displayed in a close-up of the plot in Figure 6.7, is barely noticeable. The peaks in the residuals arise at the time of the step since x_1 becomes harder to estimate due to the fast change, although the estimation error is small enough that the estimation and measurement seem to overlap perfectly. This problem can be partly remedied by raising the value of K, making the observer adapt faster to the changes of the states, but is ultimately a limitation to the precision that can be achieved. It may be noted though, that extreme sudden changes, like that inflicted by an ideal step, do not occur naturally.

This step is useful for validating the parsing itself if the model is translated from another language, but also for determining the observer gain K. In this example the gain has been set with the goal to level out estimation errors occurring in the fault free case. The maximum value of the residuals, after a simulation

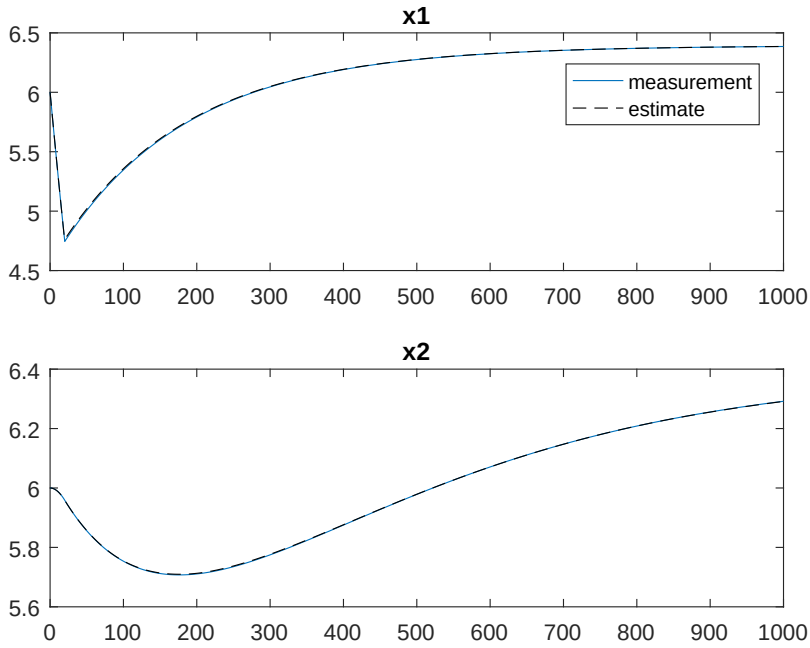

Figure 6.6: *The step response of the example model and its estimate in the fault free case.*

with $K = 1$, is used as a guideline for an appropriate value of the gain. For example would the gain in r_2 be set lower than that of r_3. This process often needs some tuning and the aim should be to obtain approximately the same order in the residuals.

6.5.2 Validation Phase

In the validation phase the data which the LFM is validated against is taken from the HFM. This data may be measurements from a real life system or in this case simulation data originating from another model. To illustrate the difference from the estimation phase, the same input is used, i.e a step, which begins at time 20 s and the gain is set to $K = 0$. In Figure 6.8 the step response of the HFM is plotted in a solid line and the step response of the LFM in Matlab, i.e. the estimate is plotted in a dashed line. The state x_1 is estimated correctly which is expected, since the equation is the same as in the HFM. The estimate of x_2 is not well fitted to the validation data, also as expected.

As a result r_2 reacts by increasing approximately with a power of ten, while the other residuals remain the same, as seen in Figure 6.9, when the gain is set to the determined value in the estimation phase. By calculating the model valida-

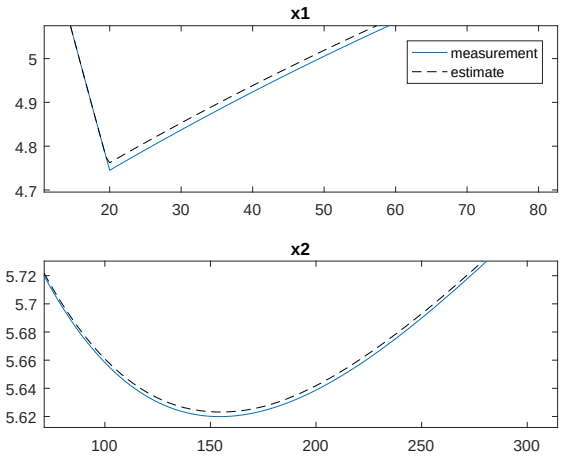

Figure 6.7: *A close-up of the plot in Figure 6.6 around the time of the biggest estimation errors of each residual.*

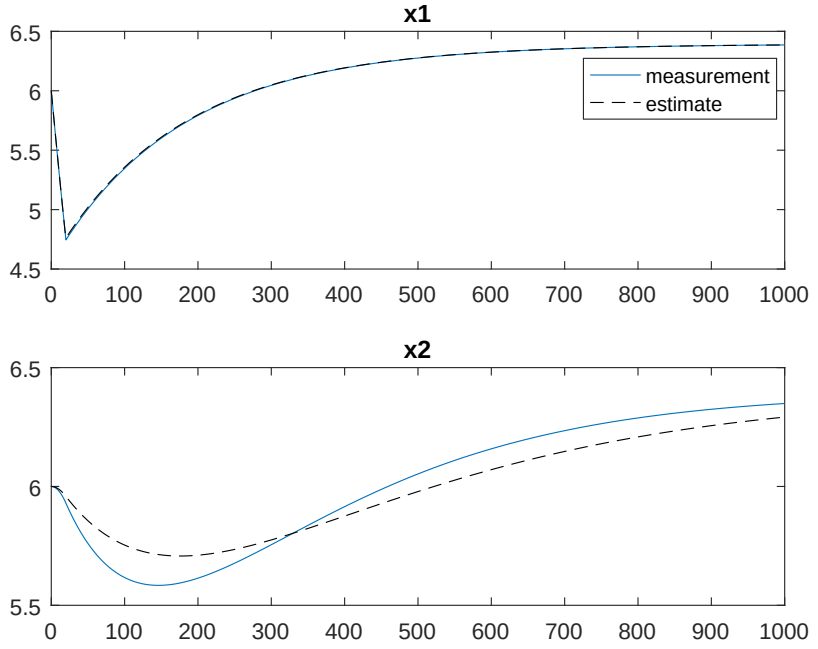

Figure 6.8: *The step response of the example model and its estimate when validating the LFM against the HFM.*

Table 6.2: *Model validation measure M_{f_i} of the validation data presented in descending order.*

M_{f_2}	7.2134e-03
M_{f_1}	3.0166e-05
M_{f_3}	1.7411e-13

tion measure M_{f_i}, according to 6.22, the result presented in Table 6.2 is obtained, pinpointing f_2 as the most probable source of model uncertainty.

6.6 Conclusion And Discussion

The method developed has proven to work for the purposes intended on this simple example. Although still in its cradle it appears to be a useful tool for narrowing down which equations might be less trustworthy. There are a few aspects the method should take into consideration, which it, due to the limited time available, does not. Different faults may not be as strongly detectable as others, which will lead to the residual reacting less if a fault is weekly detectable. Therefore the model validation measure would benefit from a weight depending on how strongly detectable a fault is, which is not supported by the FDT.

As shown in Section 6.5.1 the residual can react if the state estimation is poor, even though the model may be well fitted to the estimation data. If the estimate can be improved, e.g. by choosing a more sophisticated method for state estimation, the result may be more trustworthy. Another aspect to consider is the fact that a higher gain pushes the estimate towards the real value, meaning that the estimation error will appear lower than it really is. This is a downside of using the observer-gain to normalise residuals. Furthermore, sequential residuals are not incorporated in this method.

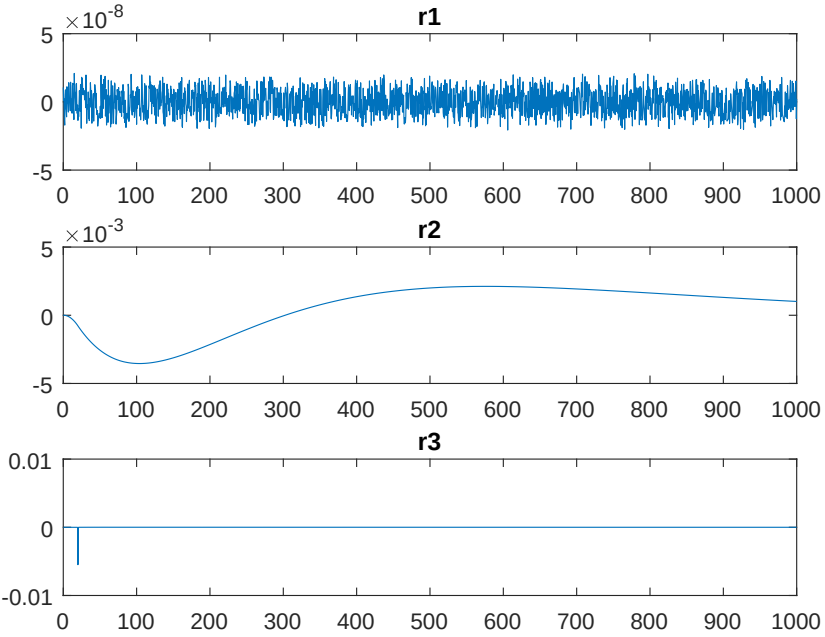

Figure 6.9: *Residuals of the LFM example model validated against data from the HFM, with non zero gain in the observers r_2 and r_3.*

7

Properties of the SECS Model

After the SECS LFM has been parsed from Modelica to Matlab structural analysis can be performed on the model. Due to the complexity of the SECS model and the limited time available, no results of the model validation are presented in this thesis. Although the creation of MSO sets and residuals was successful after limiting the number of known variables, the simulation of these were not. According to the structural analysis, the residuals are low-index problems, but under certain circumstances, namely if some equations are non-injective, the DAE may not be solvable by only deriving or integrating once. This thesis is limited to only examining low-index problems, since the solvers used cannot handle high-index problems. This problem can be solved by translating the residuals back to Modelica and letting Dymola solve the the DAE, which is examined in [12].

7.1 Model Information

The LFM describing the SECS system, depicted in 2.1, is implemented in Dymola and consists of roughly 800 equations and the same amount of unknown variables. The SECS LFM is mainly made up of equations encoded in the text layer of Modelica, while the HFM is modelled using mostly predefined components, and consequently contains a lot more equations. It is equipped with 15 sensors, that from a diagnostic point of view are regarded as known variables. The model equations describe the flow of air from the engine through the system, specifically the change of temperature and pressure throughout the system. After parsing the model from Modelica into Matlab, 22 fault variables were added to equations, that were regarded as very likely candidates for contributing with model uncertainties.

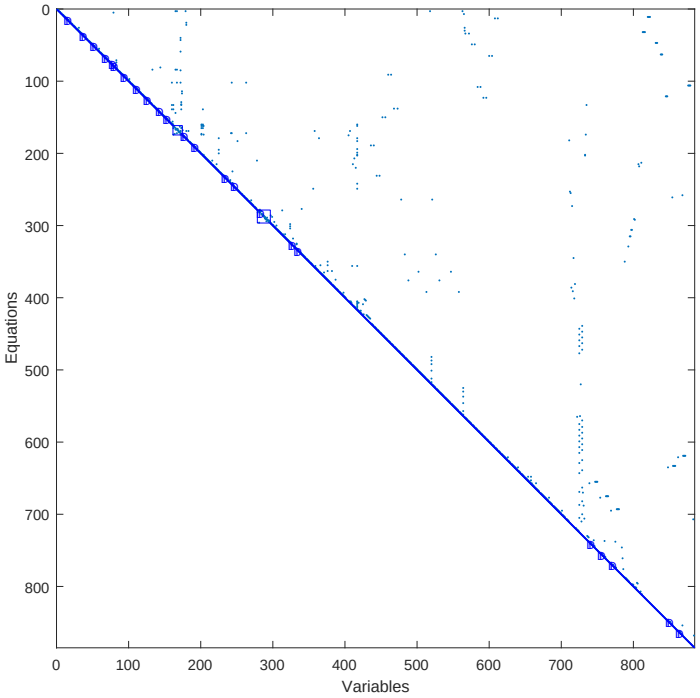

Figure 7.1: *The Dulmage-Mendelsohn decomposition of the SECS-model without any sensors.*

7.2 Structural Information

If no sensors are added to the known variables the model is exactly determined and has no redundancy, which is an indication that the parsing is done correctly. This follows from the fact that Modelica can only build a model if the model is exactly determined. The Dulmage-Mendelsohn decomposition of the model, with the sensor variables placed in $\boldsymbol{X}$, is depicted in Figure 7.1.

As described in Chapter 2 the SECS is equipped with ten temperature sensors, four pressure sensors and one accelerometer. In Figure 7.2 the Dulmage-Mendelsohn decomposition of the SECS-model, with the sensor variables placed in $\boldsymbol{Z}$, is depicted. It contains no under-determined parts but an over-determined part, seen in the bottom right corner, which is a necessity for the fault diagnosis. The 15 sensors yield a redundancy of 15, as expected.

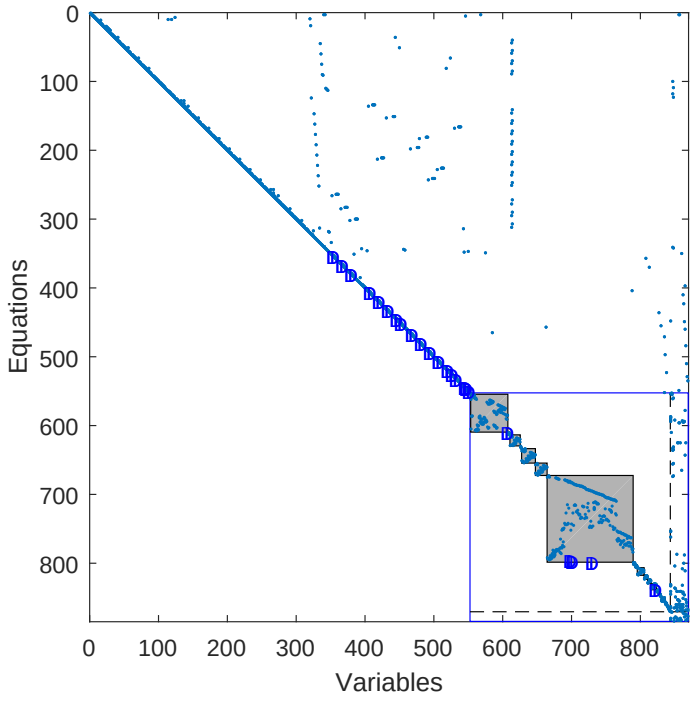

Figure 7.2: *The Dulmage-Mendelsohn decomposition of the* SECS*-model with sensors. The over-determined part seen in the bottom right corner contains the known variables.*

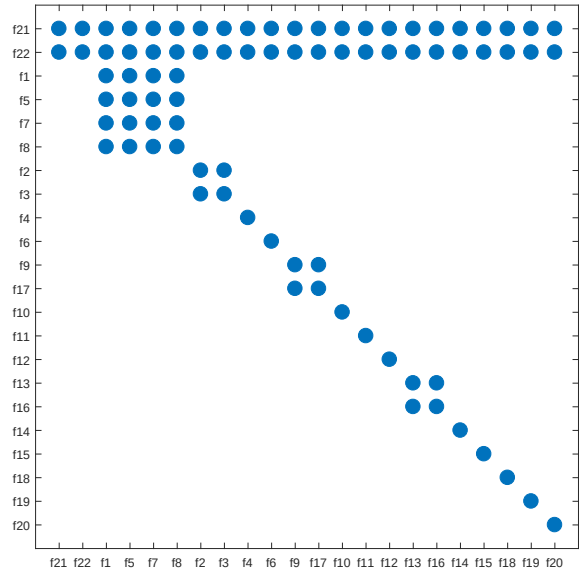

Figure 7.3: *The fault isolability matrix of the SECS model with the original set of sensors.*

7.3 Fault Diagnostic Properties

Besides the structural information the fault isolation properties of the model can be determined without having to simulate residuals. With the 15 original sensors available, a fairly high isolability can be achieved, which is depicted in Figure 7.3. As Figure 7.3 suggests fault f_{21} and f_{22} are not isolable from any other fault. This is clarified by Figure 7.4, where these two faults are not included in the over-determined part, making them not isolable from the rest. To achieve higher isolability more and/or different variables must be measured. It is also noticeable that faults appearing in a cluster, for example the faults f_1,f_5,f_7 and f_8, are not isolable from another either. The reason for this being that the faults appear within the same MSO set, as depicted in Figure 7.4. All residuals based on the MSO set will be sensitive to all faults appearing in it and hence can not be isolable from another.

Since the redundancy of the model, with all the sensors included, was to high for the FDT to be able to produce MSO sets, the number of known variables has to be reduced. As a consequence the isolation properties will be deteriorated, but this problem can be solved by removing and adding sensors in turn, depending on which faults need to be isolated. E.g. the sensors appearing in the first part of the SECS can be moved from $\boldsymbol{Z}$ to $\boldsymbol{X}$ and analysed first. Thereafter, which sensors that are to be regarded as known variables can be replaced to achieve different fault isolation properties.

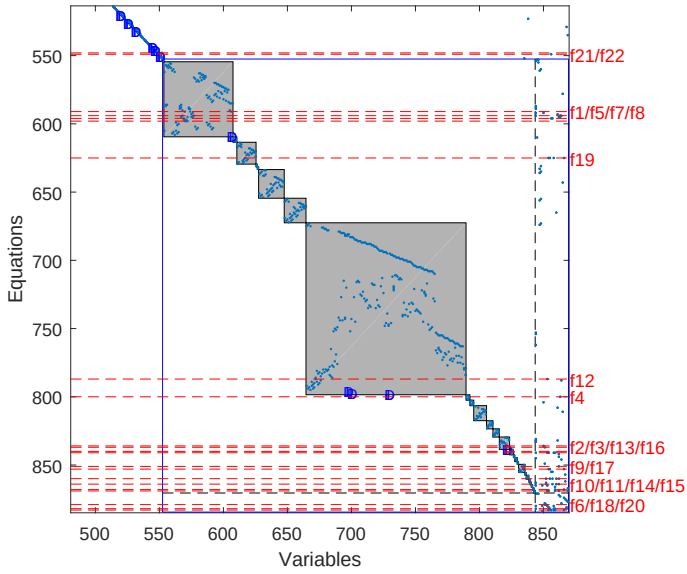

Figure 7.4: *The over-determined part of the Dulmage-Mendelsohn decomposition of the SECS LFM with faults included.*

8

Summary and Future Work

In this master's thesis the possibility to extract and translate models from Modelica in Dymola to the FDT in Matlab has been examined and successfully implemented. The structure of the Dymola-based ModelicaXML-file was of the greatest importance and was therefore examined closely. The result of the parsing was then used as a base for testing the developed model validation method. Although the method developed needs a lot more testing and further development to be useful in practice, the method has been found to work.

8.1 Future Work

This thesis may have reached its end, but the work on the subjects investigated have not. Therefore some proposals to possible ways to improve and continue the progress on the subjects examined are presented in this section.

8.1.1 Parsing of Modelica Models

In this thesis the parsing of Modelica models to Matlab has been shown to be possible. To develop the XMP further, all cases occurring in the Modelica language need to be covered and tested on more models. Furthermore, the option to translate the model to other formats and languages than the FDT requires, should be implemented to widen the area of use.

8.1.2 Model Validation

Introducing more specific faults, e.g. parameter faults, could be used to pinpoint bad parametrisation. As already mentioned the downfall with introducing more

faults is that it contributes to a rise in complexity and unsatisfactory isolation properties.

One of the greater obstacles presented is getting an adequate estimate of the states when dynamic is present in the model. Therefore a more sophisticated method of estimation could be used to further reduce inaccuracies due to calculative limitations in Matlab. An extended or unscented Kalman filter could possibly offer a more accurate state estimation. Another improvement, preferably used with a more refined state estimation method, is to normalise the residuals after the estimation phase. This was partly investigated, though not thoroughly enough to yield usable results in this thesis.

An entirely different take on residual evaluation than the one presented here could be to use machine learning elements to design a diagnosis system. By using simulation data from the model itself as correct data to train on, the system could be taught to recognise when the residuals deviate from zero due to calculative errors and when an actual fault is present.

Bibliography

[1] Elementtree overview. URL http://effbot.org/zone/element-index.htm.

[2] Andrew L Dulmage and Nathan S Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10(4):516–534, 1958.

[3] H Elmqvist, F Boudaud, J Broenink, D Brück, T Ernst, P Fritzson, A Jeandel, K Juslin, M Klose, SE Mattsson, et al. Modelicatm-a unified object-oriented language for physical systems modeling. *Tutorial and Rationale, versión*, 1, 1999.

[4] Hilding Elmqvist, Sven Erik Mattsson, and Martin Otter. Modelica-a language for physical system modeling, visualization and interaction. In *Computer Aided Control System Design, 1999. Proceedings of the 1999 IEEE International Symposium on*, pages 630–639. IEEE, 1999.

[5] Erik Frisk. Fault diagnosis toolbox. *Department of Electrical Engineering Linköping University, Sweden*, 2016.

[6] Erik Frisk, Mattias Krysander, and Daniel Jung. A toolbox for analysis and design of model based diagnosis systems for large scale models. *Department of Electrical Engineering, Linköping University, Sweden*.

[7] Erik Frisk, Anibal Bregon, Jan Aslund, Mattias Krysander, Belarmino Pulido, and Gautam Biswas. Diagnosability analysis considering causal interpretations for differential constraints. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 42(5):1216–1229, 2012.

[8] Peter Fritzson. *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons, 2010.

[9] Torkel Glad and Lennart Ljung. Modellbygge och simulering. *Linköping University. Second edition. Studentlitteratur*, 2004.

[10] Mattias Krysander, Jan Åslund, and Mattias Nyberg. An efficient algorithm for finding minimal overconstrained subsystems for model-based diagnosis.

IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 38(1):197–206, 2008.

[11] Mattias Krysander, Jan Åslund, and Erik Frisk. A structural algorithm for finding testable sub-models and multiple fault isolability analysis. In *21st International Workshop on Principles of Diagnosis (DX-10), Portland, Oregon, USA*, pages 17–18, 2010.

[12] Petter Lannerhed. Structural diagnosis implementation of dymola models using matlab fault diagnosis toolbox. Master's thesis, Linköpings universitet, 2017.

[13] Mattias Nyberg and Erik Frisk. Model based diagnosis of technical processes. 2008.

[14] Adrian Pop and Peter Fritzson. Modelicaxml: A modelica xml representation with applications. In *3rd Modelica conference*, 2003.

[15] Ulf Reisenbichler, Hansjörg Kapeller, Anton Haumer, Christian Kral, Franz Pirker, and Gert Pascoli. If we only had used xml. In *5th Modelica conference, September*. Citeseer, 2006.