Un point de vue unifié sur la diagnosticabilité

Xavier Pucel

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Institut National des Sciences Appliquées de Toulouse
Spécialité Réseaux Télécommunications Systèmes et Architectures

---

Présentée et soutenue par **Xavier Pucel**
Le 11 décembre 2008

Un point de vue unifié sur la diagnosticabilité
A unified point of view on diagnosability

Jury

Louise Travé-Massuyès	(directrice de thèse)
Philippe Dague	(rapporteur)
Luca Console	(rapporteur)
Jean-Jacques Lesage	(examinateur)
Christophe Chassot	(examinateur)
Jose Aguilar	(examinateur)
Yannick Pencolé	(invité)

École Doctorale : École Doctorale Mathématiques Informatique Télécommunications de Toulouse
Unité de recherche : LAAS-CNRS, Université de Toulouse
7 av. Colonel Roche, 31077 Toulouse
Directrice de thèse : Louise Travé-Massuyès

Remerciements

Cette thèse est le fruit de trois années de travail, non seulement de moi-même, mais de plusieurs personnes sans qui elle ne serait pas. Bien qu'ils ne soient pas mentionnés comme auteurs, ils ont participé à la réalisation de cet ouvrage.

J'ai eu la chance de travailler avec Louise Travé-Massuyès, ma directrice de thèse, qui m'a transmis sa vocation de la recherche. Son rapport au travail est un exemple, tant par la réussite de sa carrière que par l'aisance et le naturel quotidiens avec lesquels elle exerce ce métier. J'ai également bénéficié de sa clairvoyance scientifique, qui m'a fourni un sujet de recherche à la fois motivant et valorisant. Je continue grâce à elle dans cette voie avec une sérennité et une volonté rares.

Les membres de mon jury de thèse : Louise Travé-Massuyès, Philippe Dague, Luca Console, Jean-Jacques Lesage, Christophe Chassot, Jose Aguilar et Yannick Pencolé ont accepté d'évaluer mes travaux. Le temps et l'énergie qu'ils ont investi à cette tâche sont des preuves de confiance très motivantes. Les corrections minutieuses et pertinentes de Philippe Dague, Luca Console, Louise Travé-Massuyès et Yannick Pencolé démontrent l'intérêt qu'ils ont pour mes travaux. Ils me rendent au moins une partie de l'estime que je leur porte, et j'en tire une immense satisfaction.

Mes collaborateurs directs, dans l'ordre chronologique Louise Travé-Massuyès, Marie-Odile Cordier, Stefano Bocconi, Claudia Picardi, Daniele Theseider Dupré et Yannick Pencolé ont tous sans exception été des interlocuteurs pertinents et avisés, et ont joué un rôle important dans chacune de mes contributions scientifiques. Mes travaux sont tous issus des échanges que j'ai eus avec eux, tant dans l'inspiration que dans la transpiration.

Mes collaborateurs indirects, principalement les membres du projet WS-DIAMOND ainsi que les membres du groupe diagnostic, supervision et conduite (DISCO) du LAAS-CNRS, ont participé à instaurer une atmosphère de saine émulation.

Mes collègues de travail, par ordre alphabétique : Vincent Albert l'indien dans la ville, François Armando dont l'autodérision et le cynisme sont aussi fiables et constants que la lumière du phare dans la nuit, Mehdi Bayoudh et

ses prises de position, Emmanuel Bénazéra le routard sans guide, Eva Marais celle qui vient du monde normal, Fabien Perrot le geek, Hervé Ressencourt le flegmatique, Pauline Ribot et ses potins, Siegfried Soldani le super actif, Nicolas Van Wambeke et sa classe internationale, ont su me détourner du boulot lorsque c'était possible voire nécessaire. Le café et les parties de xblast, open arena, poker ou uno ont été des moments ou de détente et de relâchement, notamment des règles du fair play, de l'humour ou du uno.

Perrine Laurent, ma concubine, m'a libéré par son amour et sa présence de beaucoup de doutes et de soucis qui auraient pu me détourner l'esprit de ma thèse.

Mes amis des troglodytes, auprès desquels j'ai fini de construire mon identité pendant mes études, m'ont donné de nombreuses occasions de festoyer, et m'ont permis de découvrir d'autres modes de vie à travers leurs emplois, leurs lieux de vie et leurs projets.

Mes amis du Louron et du ski m'ont supporté et ont partagé avec moi l'amour de la montagne et le goût du sport et de l'effort. Mon instituteur Michel Castillon a parié sur l'informatique avant tout le monde, et a fait naître en moi une passion de l'informatique qui a donné lieu à une vocation vingt ans plus tard.

Mes parents, Paule par son soutien, et Jean-Yves par son exemple, et peut-être Philippe par son absence, m'ont toujours encouragé dans la voie des études. Eux et mon frère Matthieu m'ont apporté un soutien constant et infaillible depuis toujours. Ils m'ont soutenu dans mes choix d'orientation, et m'ont permis de me réaliser dans mon travail.

À toutes et à tous, et à celles et ceux que j'ai pu oublier,
Merci.

Contents

List of Figures

List of Definitions

Introduction

The development of information technologies and its applications in the industry as well as in services has introduced complex automated systems about everywhere in our lives. The number of systems used by people who have absolutely no idea of how they work is increasing if not exploding. In our own pockets, cellular phones are a good example, not to mention cars, planes, food industry, cosmetic industry, entertainment, communication, financial markets, etc. This is a fact, in modern societies, people rely on systems they do not fully understand.

As the systems around us become more complex, their failures become more and more difficult to predict, understand and repair. The need for tools for assisting the supervision of those systems has been motivating a growing effort for the last twenty to thirty years. The problem of fault diagnosis has been addressed many times, and is nowadays a mature research field. The techniques used for diagnosis have evolved from ad-hoc approaches to model-based approaches, considered more practical and more adaptive. An important issue identified during the numerous industrial applications of model-based diagnosis is that in many cases the diagnosis problem is addressed after the system is designed, making impossible the addition of sensors, or even the modification of the system in order to provide more information to the diagnosis engine.

Today, the ability of a system to be diagnosed has become a marketing argument and is part of the specifications of many kinds of systems. It is important for a system designer and manufacturer to ensure that a system is diagnosable, i.e. that the faults that may occur in it are identifiable relatively easily. This is a key to provide more reliable systems, in which maintenance costs are more predictable. This property of the system is called diagnosability, and needs to be addressed at design time. It is meant to be a factor to take into account during design, at the same level as production cost or other product qualities. The resulting need for tools that allow one to analyze a system's diagnosability and provide feedback to the designer is then straightforward.

The problem of diagnosability analysis is still recent in the research field. Various research communities have addressed the problem of model-based diagnosis in distinct and parallel tracks, and the resulting diagnosability analysis approaches inherited this heterogeneity. Although the formalisms used in these approaches are very different, the reasoning principle is very similar. Some work

has been done in order to compare and unify these approaches. This thesis contributes to this unification work and provides a general view on diagnosability that accounts for the existing diagnosis and diagnosability approaches. This thesis also addresses the problem of the integration of diagnosis in a general supervision assistance tool.

This document is divided into three parts. First, an overview on diagnosis and diagnosability analysis approaches is provided. In the exercise of model-based diagnosis, a hard and important part is building the system model. A state of the art of the existing formalisms and algorithms for diagnosability analysis reveals a significant diversity in the formalisms and in the approaches. An original investigation of implicit hypotheses and differences in each approach, and a resulting debated position for unified definitions for all the approaches are reported.

The second part includes the core of this thesis' contribution. The concept of fault signatures, imported from state-based approaches, is elected to be an universal concept for defining diagnosability. It is extended to event-based approaches in such a way that a unique definition of diagnosability stands for all the existing modeling approaches. This concept of fault signatures is extended for providing an efficient diagnosability checking approach. It is further extended to more general properties like repair preconditions or quality of service in order to integrate more easily diagnosability in a general supervision module.

The last part depicts an application to web services. The technology of web services is based on languages that appeared recently. These languages derived from XML (eXtensible Markup Language) are very modular and very structured, and have benefited from an important industrial interest since their introduction a little before year 2000. Their ability to implement distributed programs that deliver services, and to compose those services into more complex services, inside as well as across companies, has designed a particular need for reliability of these systems. The great modularity and flexibility of the systems created with these technologies makes model-based approaches very suited for their supervision. The diagnosability approach proposed to analyze such systems reuses an existing diagnosis algorithm. This algorithm uses a constraint network model and is based on constraint propagation. By modifying the execution context of the diagnosis algorithm, an original diagnosability analysis algorithm is obtained. This algorithm has several advantages, in particular a modular architecture, low communication rate, and a compact representation of the results to the designer.

The results obtained in this thesis provide grounds for a sound and efficient development of diagnosability analysis. The unified definitions allow better communication between the communities that use different formalisms. The extensions allow better integration of diagnosis and diagnosability analysis with other supervision tasks. Finally, this thesis provides a high level, lowly technical point of view on diagnosability which can make it easier to import into the industry.

Part I

Models and approaches for Diagnosability

The problem of fault diagnosis is faced by almost every one that designs complex automated systems. The solutions to this problem evolved from associative to model-based approaches. The problem of model-based diagnosis was addressed separately by two distinct communities. The community of automatic control developed Fault Detection and Isolation (FDI) techniques for continuous and event based systems. The community of Artificial Intelligence developed approaches based on first-order logic and constraint networks. Both communities also used discrete-event formalisms to adress the diagnosis problem. These approaches were developed in three distinct and parallel tracks, and comparing them reveals great similarity in the overall reasoning, but great diversity in the modeling details. When these communities address the problem of diagnosability analysis, the same diversity is reproduced.

This chapter provides an overview of model-based diagnosability analysis approaches. First, considerations about how to model an automated system are presented, independently of the underlying formalism. Then the existing formalisms and approaches to diagnosability are described. Finally, generic notions are defined for all approaches, which provide grounds for abstract reasoning.

Chapter 1

Model-based diagnosis and diagnosability

Fault diagnosis is a problem that needs to be addressed when supervising an automated system. The system offers a predefined interaction with the user or with a controller: it receives inputs (commands, parameters, etc) and produces outputs (products, sensor values, etc). Some of the inputs and outputs are known to the user, and allow one to estimate in a relative way what is happening in the system. The system is not 100% reliable, and some faults may alter its behavior and prevent it from fulfilling its purpose, or even threaten the security of its users. The problem of fault diagnosis consists in monitoring the observations emitted by the system, and look for symptoms that would indicate that the system is under a fault.

1.1 Model-based diagnosis

Automated fault diagnosis has been receiving interest from the scientific community for over thirty years. The early approaches to diagnosis like expert systems are based on association of symptoms to faults, in a similar way as medical diagnosis is performed. These approaches rely on the reuse of past experience in order to assist the diagnosis of future faults. The main limitation of these approaches is that the experience may be very long to acquire, particularly for the cases in which several faults are combined, and sometimes longer than the system's life cycle. Moreover, these methods may not be applied to a new system, or to a newly modified system. Any modification of the system makes all or most of the expert knowledge outdated. Finally, in some systems, the number of possible faults is so large and these faults are so rare that these approaches are simply not accurate.

In response to the limitations of associative diagnosis approaches, model-

based diagnosis approaches were developed. They are based on a description of the system's behavior in a mathematical language called the system model. This model is generally provided by the system designer, and describes at least the normal behavior of the system. Better results are achieved when the model describes the behavior of the system under predefined faults, or when the model accounts for the system structure, i.e. the components that intervene in the system behavior.

Some preliminary work can be performed at design time, in order to facilitate the diagnosis process. In some approaches, preliminary work consists in selecting the consistency tests to rely on for diagnosis, this selection can be assisted by automated tools. In some other approaches, preliminary work consists in compiling the model into an automaton that organizes all the reachable diagnosis results, and that just needs to be fed with the runtime observations. In other approaches, no preliminary work is done, diagnosis is performed on the model "out of the box", fed on line by observations.

In some cases, several fault situations can explain the observations, the diagnosis is ambiguous. The precision of the diagnoses that are produced at runtime is an important aspect. It is not worth modeling a system and running a diagnosis engine if it always provides diagnoses that are so undetermined that no decision can be made upon them. This problem is called diagnosability analysis, and is a prerequisite for diagnosis.

1.2 Diagnosability analysis

Industries that want to add diagnosis capabilities to their systems often face a particular problem, that comes from the fact that the system designers are in most cases not competent in model-based diagnosis. Hence diagnosis is not taken into account at design time, but later on in the system development process. The problem of this protocol is that diagnosis cannot influence the design of the system, for example to add a sensor inside a solid block of the system. This can be avoided by providing to the designers tools that provide feedback about the diagnosis capabilities of the system.

Diagnosability is the property of a system and its instrumentation to exhibit different symptoms for a predefined set of anticipated faults. Diagnosability analysis totally relies on the existence of a model of the system, as illustrated in figure 1.1. It is often defined as a Boolean, yes/no property, but it is more complex than that. In most cases, when several faults are combined, the observations are so disturbed that they do not allow to make the distinction if an additional fault occurs on top of that, this would make the system not diagnosable. However, the more faults are considered, the less probable it is for them to be present at the same time, and some diagnosis approaches limit the analysis to single or double faults, and do not consider other faults. Other diagnosis approaches only provide so called minimal diagnoses, i.e. if a single fault and a double fault including the single one can explain the observations, the algorithm

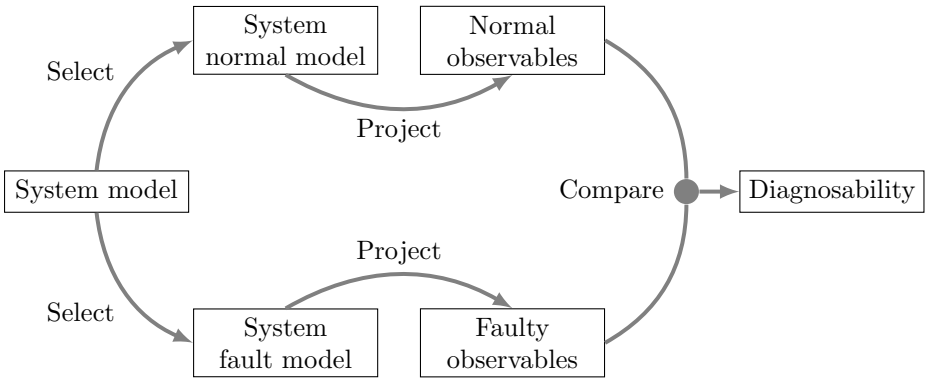


Figure 1.1: Overall approach for diagnosability

chooses the single fault. Some works aim at defining diagnosability as a degree, i.e. an integer or real number, but this is, as well as the Boolean representation, not enough to express diagnosability in all its complexity.

If a system is not diagnosable enough, this may result from two causes:

- The system does not contain enough sensors, or the sensors are badly placed. The observations returned by the sensors do not allow one to discriminate two different fault situations. This can be solved by adding sensors in the system, which is not always possible.
- The system produces sufficient information, but the model is not precise enough, it is too abstract (abstraction is defined and discussed in section 1.3). In this case, it is not necessary to modify the system, but only the model used for diagnosis. Some information needs to be added in order to provide better inputs to the diagnosis engine.

It seems natural to use the same model for diagnosability analysis and diagnosis, since the results of diagnosability analysis would not be relevant for a diagnosis approach that uses another model. Modelling a system for diagnosability analysis and for diagnosis is consequently the same task.

1.3 Knowledge representation and abstraction

When supervising a system, having an exact and precise diagnosis requires a correct and sufficiently complete model. An incorrect model may result in erroneous diagnoses, and a model not complete enough may result in too imprecise diagnoses. The issue of model correctness for diagnosis has been addressed in [Lopéz-Varela, 2007], and is not considered in this document.

The size of the model is also an important question. Model-based diagnosis and diagnosability analysis are complex algorithms, and the computation time increases with the size of the model. There are two ways to control the size of the model: the formal language used for modeling the system, and the level of abstraction at which the modeling is done. Moreover, some additional factors may influence the modeling, in particular the type of faults and the type of decision to be taken in reaction to a diagnosis. This chapter presents the aspects mentioned above to take into account when modeling a system for diagnosis.

The first step when designing a diagnosis module is to choose which information to take into account for diagnosis. This step has dramatic consequences on the final result since the information discarded at the modeling stage is then inaccessible during diagnosis. Hence if some crucial information is neglected, the resulting module will probably exhibit a poor diagnosability level. On the other hand, it is in general impossible to express and compute information about the whole system in its finest details, because this would make the diagnosis problem intractable for a computer. The choice of which information to make abstraction of and which information to take into account is consequently a difficult choice that must be done by a human designer.

A model is always an abstraction of the real world in the sense that there are always some aspects of the system that are not included in the model. In [Ressencourt, 2008, Lind, 2003], several types of abstraction are identified: functional abstraction, abstraction by aggregation, and qualitative abstraction.

1.3.1 Functional abstraction

Functional abstraction consists in describing what the system does instead of how it works [Chittaro *et al.*, 1993]. It relies on a classification of the knowledge about the system, and its organisation into a hierarchy. At the lowest level of abstraction stands knowledge about the physical composition of the system, while at its highest level is the knowledge about the system purpose. 4 classes are proposed:

Structural knowledge: this is the lowest abstraction level. It describes the components, their nature and how they are interconnected. It contains what the components are made of, and the interconnection rules. This is a pure physical knowledge totally independent of how it works and what for.

Behavioral knowledge: at this level, the behavior of components is represented. It describes how the components work and how they interact, their states and they operation.

Functional knowledge: this level is attached to the description of the processes that happen in the system, and the roles played by the components in the realization of these processes.

Teleological knowledge: This is the highest level of abstraction, that describes the goals of the system, and the conditions that must be fulfilled while achieving these goals. This is a pure specification level of knowledge, totally detached of how the system is implemented.

Most diagnosis approaches rely on structural and behavioral knowledge, and sometimes on functional knowledge. However, the link with teleological knowledge must not be neglected, since diagnosis is just a part of the supervision task, and in general repair or reconfiguration takes into account teleological knowledge.

1.3.2 Abstraction by aggregation

Abstraction by aggregation, also known as structural abstraction, consists in encapsulating knowledge into a "black box". The relations between pieces of knowledge inside and outside the black box are kept, but the relations between pieces of knowledge inside the black box are abandoned. This kind of abstraction has been used for model-based diagnosis [Chittaro and Ranon, 2004, Autio and R.Reuter, 1998].

The abstraction by aggregation is generally applied to all the categories of knowledge mentioned in the previous section: when several components are aggregated, so are the behaviors, the functions and the goals of the original components. The result is a large black box component, whose internal behavior functions and goals are hidden, but whose interactions with other components, whose external functions, preconditions and goals are described.

Such an abstraction can be dictated by the possible repair actions. If replacing a resistor can only be done by replacing a whole electronic card, aggregating the components that form this card is a good idea. In this case, it is not necessary to distinguish faults on two components inside the card, since the decision that follows these two diagnoses is the same. Hence, abstraction by aggregation can lead to the decrease of the number of modeled faults, which has a critical impact on diagnosis and diagnosability.

1.3.3 Qualitative abstraction

A qualitative abstraction consists in reducing the domain of the variables used to describe the model. Such an abstraction is very helpful to compact behavioral or functional knowledge, and makes it easier to analyze. For example, a real number can be abstracted into a value that can have the following three qualitative values $\{-, 0, +\}$. Sometimes more values are necessary, and can be related to intervals for the original variable. When using a qualitative model for diagnosis, the idea is to capture only the relevant aspects of the behavior.

This kind of abstraction is difficult to perform, since it can have a significant

and unexpected impact on the diagnosability of the system. However, its impact on the size of the model is such that it is very commonly used.

Example 1.1 (Abstraction and diagnosability) *Let us consider a cluster of electrical lights bulbs in a series circuit. If one bulb breaks into an open circuit, all the bulbs go off, and the observations (voltage on, light off) do not allow to tell which bulb is broken. This is an example of a non diagnosable system.*

At the aggregated level, the faults in all bulbs are assimilated to a unique fault, which makes the system diagnosable, since the absence of light indicates that some bulb has gone off.

In reality, the observations allow us to tell that the fault occurred in the light bulb cluster, but not in which bulb. According to what is considered as a fault (one fault per bulb, or one fault for the whole cluster) the system is considered as diagnosable or not.

Figure 1.2 shows how the knowledge about this system can be classified according to its abstraction level, and shows the best level of abstraction at which to consider the system for diagnosis. The model could be simplified further by a qualitative abstraction, by considering that light and voltage are Boolean variables, thus simplifying the behavioral model.

1.4 Modelling formalisms

The information to be considered in the diagnosis reasoning depends mainly on the nature of the system itself, and on the nature of the faults to be diagnosed. A hydraulic system supposed to achieve several precise continuous functions should not be modeled the same way as a telecommunication system characterized by complex communication protocols, since the most suited models are different. Although some common elements should naturally appear in both models, they must not take into account the same aspects of the system: the precise relation between variable values and their derivatives in the case of a hydraulic system, or the high importance of sequential aspects in the telecommunication system. The nature of the observation instrumentation also influences the information to be considered, since it generally offers an abstraction of what is really going on in the system.

Let us illustrate these considerations with examples. In an electronic signal processing system, the physical variables are related by differential equations, since the physical values (voltage, current) and their derivatives are linked by constraints that result of the electronic components in the system (capacitors, inductors, ...).

In a telecommunication network, although a lot of signal processing is done for modulation and demodulation, the supervisor is not interested in the details of these operations. The important aspects of the system consist in monitoring

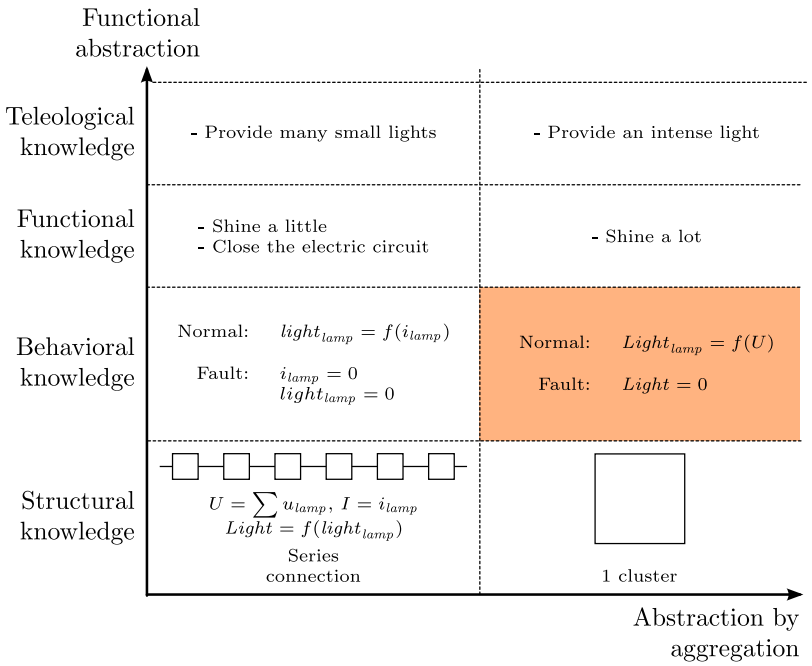
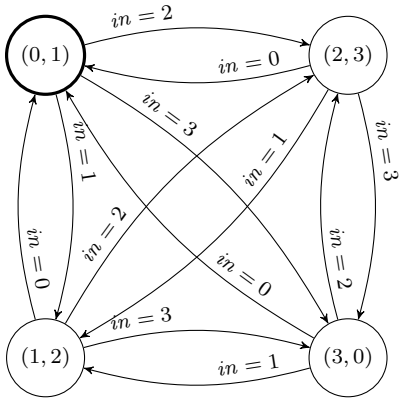


Figure 1.2: Model abstraction levels. Classification of the knowledge used in the diagnosis model for the system described in example 1.1. The knowledge is classified by levels of functional and abstraction by aggregation. The highlighted knowledge is sufficient to perform diagnosis, other knowledge do not need to be included in the model.



$$out = (in + 1)\%4$$

Figure 1.3: Adapted and unadapted formalisms for modeling systems.
On the left, a automaton modeling a circular adder for 2-bits coded numbers. In each state, a couple indicates the values for the input and the output. The initial state has a thick border.
On the right, a constraint model for the same system, much simpler, smaller and easy to maintain.

the resource management protocols, that allow one to detect overloads or physical failures in some nodes of the network. The model focuses on the sequences of messages, and maybe their content or delays, since the type of messages and the order of emission is more important than their internal parameters or the exact date of emission.

In a biological process, some physical variables can be relevant (temperature, pressure, ...), but some important criteria are difficult to describe with numbers, or even difficultly measurable, like the growth of some bacteria colony, or the hardness of the crust of a bread bun. In those case it is generally possible to abstract numeral aspects like the temperature into a small set of qualitative values "too cold, pretty cold, normal, pretty hot, too hot". This kind of abstraction does not prevent to reason on the system, and can even facilitate it. It is also possible to abstract the derivatives of such variables into "growing, decreasing, stable". In such processes, these qualitative aspects are more important than the real precise temperature or the exact number of living bacterias, or even than the order in which two bacteria colonies appear.

Figure 1.3 illustrates how the choice of the modeling formalism is important and has critical influence on the whole diagnosis and diagnosability analysis tasks.

1.5 Diagnosis context

The context in which diagnosis is performed can also influence the modeling phase. We distinguish two types of contexts:

- Online versus offline
- Instance versus class (for functional model mainly)

Online diagnosis is performed while the system is running, whereas *offline* diagnosis is performed when the system is in a test bench. Diagnosis in each context is a very different task, since online the observations result from the actual use of the system, while offline it is necessary to perform tests under various operating modes. Each context offers advantages in terms of diagnosis. For example, online diagnosis allows one to measure how a plane wing behaves inside turbulence, which is not possible offline, on the ground. On the opposite, offline it is possible to test only some components or functions of the system independently of the rest of the system. It is possible to test the command of plane reactor separately, which would not be possible online, during a flight. Moreover, online diagnosis applications are constrained by time, and computations must be done in adequation with the system's dynamics. A compromise between accuracy and performance often needs to be found.

Another type of context can arise when diagnosis is based mostly on functional knowledge. This type of situation, illustrated in part III, occurs when the system is flexible and modular, and its structure may change, while its functional specifications are constant. Faults, observations and decisions that result from diagnosis are related to functional knowledge. In this type of context, the distinction between *instance* and *class* contexts is important.

In the functional instance context, diagnosis and in general supervision focuses on one execution, or instance, of one function. Diagnosis aims at detecting faults that may prevent this function execution to complete successfully. The decisions that result from diagnosis are related to this one execution of the function. Information is not shared between several instances of the function.

In the functional class context, diagnosis and supervision are attached to the aspects that are shared by all the executions of a function. Such aspects contain average values, like response time, or success rate, as well as data and components that are shared by several function instances. This context is a necessary intermediate between functional instance supervision and structural and behavioral supervision.

Chapter 2

Different formalisms for diagnosability

The model-based approaches for diagnosis and diagnosability analysis that were developed in the domains of automatic control and artificial intelligence can be classified into two categories: state-based approaches, and event-based approaches.

State-based approaches consider state-based systems (SBS), i.e. systems in which the characteristic aspects, in particular faults and observations depend on the system state. Faults and observations are represented by variables, and are associated to a set of states.

Event-based approaches consider event-based systems (EBS), systems in which the essential aspects of the system can be modeled by discrete events. These approaches generally model faults and observations as discrete events between unqualified states.

Some diagnosis and diagnosability analysis approaches address hybrid systems, and benefit from the modeling power of the two previous approaches.

2.1 State-based approaches

State-based approaches to diagnosability analysis generally rely on formalisms using a set of variables and a set of constraints over these variables in order to describe the system behavior.

The behavior model of a SBS $\Sigma = (R, V)$ is generally described by a set of n relations R, which relate a set of m variables V. If the values domain of each variable v_i of V is denoted D_{v_i}, then the relations r_i of R constrain the

set of admissible tuples of values of the variables to a subset of $D_{v_1} \times \cdots \times D_{v_1}$. In a component-oriented model, the relations r_i are associated to the system's physical components, including the sensors. The set R can be partitioned into behavioral relations, which correspond to the internal components behaviors, and observation relations which correspond to the sensors behaviors. The set of variables V is partitioned into the set of observable variables V_{OBS}, whose corresponding value tuples are called *observations*, and the set of unobservable variables noted V_{UNOBS}.

Observation values, possibly processed into fault indicators by a series of abstractions, provide a means to characterize the system at a given time. Faults may also be characterized by variables.

Without loss of generality, we equate fault indicators and system variables and always speak of the *observation tuples*. The set OBS is defined as the set of all the possible observation tuples, i.e., $OBS = \{(o_1, o_2, \ldots, o_k)\}$. The observation value pattern is referred to as the *observed signature* whereas the expected value patterns for a given fault, obtained from the behavioral model, provide the *fault signature*. Note that several value patterns may correspond to the same fault, for example when the system undergoes several operating modes. The fault signature is hence defined as the set of all possible observation tuples under the fault. When fault signatures are explicitly known for all anticipated faults, the diagnosis process comes down to comparing the observed signature with fault signatures. The "no fault" situation being considered as a special fault case with its associated signature, fault signatures also allow one to test fault detectability.

2.1.1 FDI approaches

The FDI acronym stands for Fault Detection and Isolation, and denotes the diagnosis approaches that are developed in the automatic and control community. In the FDI community dealing with continuous systems, a well-known method known as the parity-space method consists in using the system model to establish a set of Analytical Redundant Relations (ARR). This comes down to abstracting up the set of relations R into a reduced set of ARRs, which give rise to fault indicators known as residuals. The residual values are again abstracted into Boolean values that equal 0 when the corresponding ARR is satisfied and 1 when it is violated, indicating in this later case that there is a fault in the system. The observations hence result in a Boolean fault indicator tuple. In most applications, several residuals are used to monitor several faults. Each residual is sensitive to some faults and insensitive to some others, the Boolean pattern providing the fault signature. The signatures for all the system faults can be described in a *fault signature matrix*.

Criteria for fault detectability are given in [Nyberg, 2002]. The system is

modeled by a set of linear differential equations:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + B_u u(t) + B_d d(t) + B_f f(t) \\ y(t) &= Cx(t) + D_u u(t) + D_d d(t) + D_f f(t) \end{aligned} \tag{2.1}$$

- x is the state vector, it describes the current system state.
- y is the output vector, whose value is known by the supervisor.
- u is the known input vector, that describes the inputs that are measured or controlled by the supervisor.
- d is the unknown inputs vector, it describes the other system inputs, like disturbances, noise, etc.
- f is the fault signal, that is supposed to be constant.

This linear system model can also be expressed under the transfer function form, using the Laplace transform:

$$y = G(s)u + H(s)d + L(s)f \tag{2.2}$$

A residual generator is a function that takes as input the measured variables in the system, i.e. u and y. It is designed so as to be insensitive to the commands and disturbances, but sensitive to the faults. Formally, a residual generator r is defined as:

$r = Q(s)\begin{pmatrix} y \\ u \end{pmatrix}$ which gives by (2.2):

$$r = Q(s)\begin{bmatrix} G(s) & H(s) \\ I & 0 \end{bmatrix}\begin{bmatrix} u \\ d \end{bmatrix} + Q(s)\begin{bmatrix} L(s) \\ 0 \end{bmatrix} f$$

The matrix $Q(s)$ may be used for a residual generator if and only if r is insensitive to the inputs u and d, i.e. if $Q(s)\begin{bmatrix} G(s) & H(s) \\ I & 0 \end{bmatrix} = 0$, and sensitive to the fault with $Q(s)\begin{bmatrix} L(s) \\ 0 \end{bmatrix} \neq 0$.

Definition 2.1 (FDI Detectability) *A fault f is detectable if and only if there exists a residual generator that is sensitive to its signal.*

Various mathematical criteria are given in [Nyberg, 2002] for fault detectability in linear systems.

In [Isermann, 2005], some variations of this approach are described. Faults modeled by a signal that is added in the dynamic equation, as described above, are called *additive faults*. Another type of faults can be represented by modifying the value of the parameters A, B_u or B_d, in this case they are called *multiplicative faults*.

Another way to build residual generators is to perform *structural analysis* to exhibit the subsets of equations corresponding to Minimal Structurally

	f_1	f_2	f_3	f_4
r_1	0	0	1	0
r_2	0	1	0	1
r_3	1	1	0	0

Figure 2.1: An example of a fault signature matrix. Residuals are represented by rows, and faults in columns. A 1 (0) value indicates that the corresponding residual is (in)sensitive to the corresponding fault. All faults are isolable since their signatures are unique.

Overdetermined (MSO) subsystems [Krysander *et al.*, 2008]. The relations constituting a MSO can then be manipulated to perform algebraic substitutions and eliminate unmeasured variables. The resulting relations are parity relations, or *Analytical Redundancy Relations* (ARR). Such relations can only be derived if the model has more equations than variables, i.e. if the system is over constrained. [Krysander *et al.*, 2008]provides an algorithm that identifies the smallest over constrained subsystems, which allows to build ARRs efficiently by performing the substitutions only in such subsystems.

[Isermann, 2005] mentions another way to build residual generators, by means of state observers. While the parity space approach consists in projecting the model on observed variables, the state observer approach consists in estimating the value of unobserved variables. Both approaches take benefit of the redundancy in the model to generate consistency tests based on the residual generators. Overdetermined subsystems contain redundancy that can be used to generate residuals via the parity space or the observer approach.

Fault diagnosis in non-linear systems has been addressed with the observer approach in [Polycarpou *et al.*, 1997], and with the parity space approach in [Frisk, 2000].

The different faults and residuals are generally represented in a matrix called the *fault signature matrix*, as illustrated in figure 2.1.

Definition 2.2 (FDI Fault signature) *The* signature *of a fault is the set of all tuples containing the values obtained for the different residuals when the fault is present in the system.*

Fault signatures allow one to define fault isolability: if two faults do not stimulate the same sets of residuals, then a supervisor is able to discriminate them. Consequently, it is possible not only to detect these faults, but to isolate them.

Definition 2.3 (FDI Isolability) *A fault is* isolable *if and only if its signature does not contain any common tuple with any other fault.*

2.1.2 DX approaches

In the DX community, fault indicators are not usually built at design time. On the contrary, all the computation is made at runtime. The principle of abstracting the set of relations into a set of fault indicators is not applied, the abstraction is rather concerned with the variable values. Indeed, variable continuous domains are generally abstracted up into a finite number of qualitative values representing a given homogeneous qualitative behavior, for instance *ok* indicating that the variable behavior is consistent with its normal behavior, and *ab* indicating that it is not [Hamscher *et al.*, 1992, Dubuisson, 2001, Pucel *et al.*, 2007]. This kind of qualitative abstraction is common in these approches, but not necessary.

Conflict based diagnosis

The first original approaches to diagnosis reported in [Hamscher *et al.*, 1992] use first-order logic to model the system. Following approaches are based on propositional logic, or constraint networks, but inherit the same system model definition.

Definition 2.4 (DX System model) *The system is modeled by the following sets:*

- *COMPS is the set of system components.*
- *SD is the system description. It is a set of propositions written in first-order logic. Faulty components are modeled by a unary predicate AB(.), interpreted to mean "abnormal".*
- *OBS is the set of observations, written as first-order logic propositions.*

When $c \in COMPS$, the clauses $AB(c)$ and $\neg AB(c)$ are referred to as AB-literals*.*

Definition 2.5 (DX Diagnosis) *Let $C_1 \subseteq COMPS$, $C_2 \subseteq COMPS$ be two sets of components. We define $\mathcal{D}(C_1, C_2)$ as the following conjunction:*

$$\mathcal{D}(C_1, C_2) = \left(\bigwedge_{c \in C_1} AB(c) \right) \wedge \left(\bigwedge_{c \in C_2} \neg AB(c) \right)$$

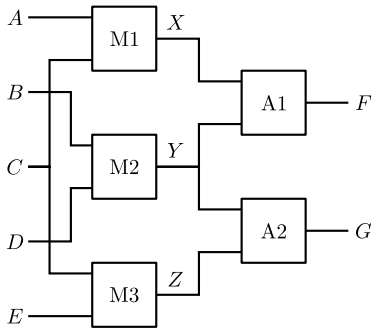
Let $\Delta \subseteq COMPS$ be a set of suspect components. The proposition $\mathcal{D}(\Delta, COMPS \setminus \Delta)$ is a diagnosis *for $(SD, COMPS, OBS)$ if and only if the following proposition is satisfiable:*

$$SD \cup OBS \cup \mathcal{D}(\Delta, COMPS \setminus \Delta)$$

A diagnosis $\mathcal{D}(\Delta, COMPS \setminus \Delta)$ is a minimal diagnosis *if and only if for every $\Delta' \subset \Delta$, $\mathcal{D}(\Delta', COMPS \setminus \Delta')$ is not a diagnosis.*

An advantage of these approaches is that the behavior of the system may be only specified in the normal case, as illustrated in figure 2.2. When a fault is present, the variables of the faulty component are unconstrained, and its behavior is unspecified. Diagnosis consists in identifying conflicts, which identify sets of components that cannot be all in a normal behavioral mode, and in establishing a diagnosis as a hitting set of all the conflicts.

Since faulty behavior is unspecified, it is always possible to suspect every component to be faulty. Consequently, the conjunction $\mathcal{D}(COMPS, \emptyset)$ is always a diagnosis in all situations. In general, one needs to expect an exponential amount of diagnoses with respect to the number of components. Under some reasonable hypotheses on the components behavior, the whole set of diagnosis is fully characterized by minimal diagnoses [Hamscher *et al.*, 1992, Dubuisson, 2001]. In this case, only minimal conflict sets and their minimal hitting sets are considered, in order to provide *minimal diagnoses*, which can be computed faster.



$$COMPS = \{M1, M2, M3, A1, A2\}$$

$$SD = \left\{ \begin{array}{c} \neg AB(M1) \Rightarrow X = A.C \\ \neg AB(M2) \Rightarrow Y = B.D \\ \neg AB(M3) \Rightarrow Z = C.E \\ \neg AB(A1) \Rightarrow F = X + Y \\ \neg AB(A2) \Rightarrow G = Y + Z \end{array} \right\}$$

Figure 2.2: A simple system, the polybox, is composed of 3 multipliers and 2 adders. Its model is described on the right side. Variables A, B, C, D, E, F, G, X, Y, and Z have finite domains.

Example 2.1 (Conflicts and diagnoses) *Let the system be the one described in figure 2.2, and let the observations be:*

$$OBS = \{A = 3 \wedge B = 2 \wedge C = 2 \wedge D = 3 \wedge E = 3 \wedge F = 10 \wedge G = 12\}$$

With the inputs provided, both F and G should be equal to 12. Two minimal conflicts can be computed:

$$AB(A1) \vee AB(M1) \vee AB(M2)$$
$$AB(A1) \vee AB(M1) \vee AB(M3) \vee AB(A2)$$

The meaning of these minimal conflict sets is that at least one component in the set $\{A1, M1, M2\}$ and one component in $\{A1, M1, M3, A2\}$ are faulty. Four minimal diagnoses can be derived from these minimal conflict sets:

$$\Delta_1 = \{A1\},\ \Delta_2 = \{M1\},\ \Delta_3 = \{M2, M3\},\ \Delta_4 = \{A2, M2\}$$

Minimal diagnoses are the most optimistic diagnoses, the diagnoses that suspect the smallest sets of components. Generally, it is assumed that components

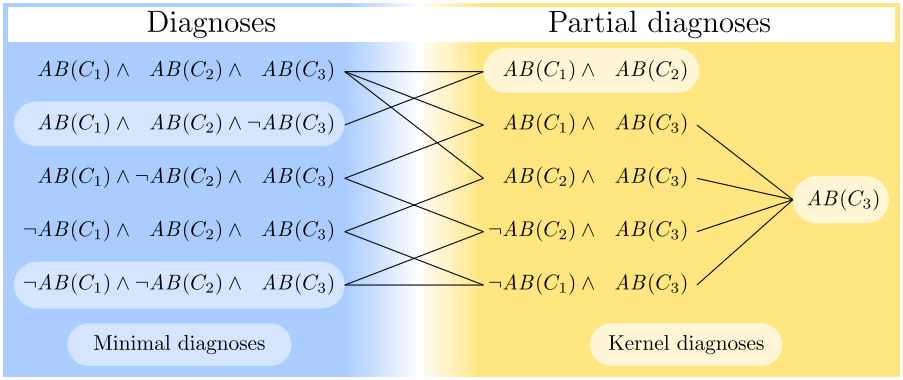


Figure 2.3: Minimal, partial, and kernel diagnoses for an illustrative set of 5 diagnoses involving 3 components C_1, C_2 and C_3.

are more likely to be normal than faulty, and minimal diagnoses are also the most probable. However, given the set of minimal diagnoses, one cannot deduce the whole set of diagnoses in the general case.

The concepts of partial and kernel diagnoses allow one to describe the set of diagnoses in a compact way, but still allow one to test whether a set of suspect components leads to a diagnosis or not. The principle is that if both $AB(C_1) \wedge \neg AB(C_2) \wedge AB(C_3)$ and $AB(C_1) \wedge \neg AB(C_2) \wedge \neg AB(C_3)$ are diagnoses, it can be interpreted as C_1 is faulty, C_2 is not faulty, and C_3 may or may not be faulty. This can be represented more compactly by $AB(C_1) \wedge \neg AB(C_2)$, which is a *partial diagnosis*. Partial diagnoses are organized in a hierarchy, as illustrated in figure 2.3. The partial diagnoses that are minimal, at the roots of the hierarchy, are called *kernel diagnoses*.

Definition 2.6 (DX Partial and Kernel diagnosis) *A* partial diagnosis *is a satisfiable conjunction P of AB-literals such that for every satisfiable conjunction Φ of AB-literals that extends P, $SD \cup COMPS \cup \{\Phi\}$ is satisfiable.*

A kernel diagnosis *is a partial diagnosis that does not extend any other partial diagnosis.*

In some situations, reasoning with only the model for the normal behavior of the system is an advantage. However, in this case it is not possible to discard a fault in a component, and in general for one observation there are many diagnoses. The diagnosis process can be much more precise and efficient when fault models are provided. In this case it is possible to assess not only which components are faulty and how, but also which components are normal.

Diagnosability analysis relies on the existence of fault models. The next section presents how the original diagnosis approaches can be extended in order to support diagnosability analysis.

Fault models

In the absence of fault models, only the nominal behavior of the system is specified, the faulty behavioral mode of the component is then called the *unknown mode*. With such a model it is possible to *detect* faults, and, with the information related to the structure of the system, to *locate* them.

The need for fault models has been addressed in the literature [Hamscher *et al.*, 1992], not necessarily for diagnosability analysis, but also for diagnosis. Some components may host several different faults that alter differently their behavior. For example, a damaged electrical component can behave like an open circuit or a short circuit according to the fault that occurred in it.

Fault models can be identified formally by a set of predicates or by a mode variable for each component. The behavior of the component for each fault is defined by a set of constraints associated to the predicate or mode variable value that corresponds to the fault. The (minimal) conflict set / hitting set approach is still valid with such models. The unknown mode can be defined as a particular behavioral mode associated to an empty set of constraints.

Specific fault models can also be defined using an exoneration model. If CM is a set of constraints representing the normal behavior of a component C, then $\neg AB(C) \Rightarrow CM$ is a normal model, and $\neg AB(C) \Leftrightarrow CM$ is an exonerated model. Exonerated models simply state that the exonerated component does not behave the same way when it is faulty and when it is normal.

Diagnosability

There exists few approaches to diagnosability in the DX community. In [Dressler and Struss, 2003], the problem of diagnosability analysis consists in identifying the operating conditions in which various faults are discriminable one from another.

In this approach, fault models are defined and modeled by mode variables. The concept of observable variables is introduced by means of a projection operator $PROJ_{obs}$ that projects a set of states on the corresponding set of observation tuples. Some variables, called *characterizing* variables, define the operating condition of the system. The problem addressed by this approach is, given two behavioral modes (fault/fault or fault/normal), to determine in which conditions they are not discriminable, necessarily discriminable or possibly discriminable. For a given set of characterizing variables, the set SIT_{NotD} (resp. SIT_{NecD}) is computed, it contains the situations in which the two modes are not discriminable (resp. necessarily discriminable). When the characterizing variable values belong to SIT_{NotD}, the observations do not allow the supervisor to tell in which mode the system is. On the other hand, when the characterizing variables belong to SIT_{NecD}, the supervisor knows from the observations in which mode the system is.

$$\begin{cases} ARR_1 : F - (A.C + B.D) = 0 \\ ARR_2 : G - (B.D + C.E) = 0 \\ ARR_3 : F - G - A.C + C.E = 0 \end{cases}$$

	F_{A1}	F_{A2}	F_{M1}	F_{M2}	F_{M3}
ARR_1	1	0	1	1	0
ARR_2	0	1	0	1	1
ARR_3	1	1	1	0	1

Figure 2.4: Analytical Redundancy Relations and single fault signature matrix for the Polybox example (see figure 2.2).

This comparison is to be performed for each combination of normal and faulty behavior for all the components in the system, which can be extremely time-consuming.

2.1.3 Unification of state-based approaches for diagnosis and diagnosability

The diagnosis approaches developed in the DX and FDI communities have been compared [Cordier *et al.*, 2004], and the resulting bridging concepts were applied for diagnosability analysis [Travé-Massuyès *et al.*, 2006b].

The comparison between the two approaches rely on the construction of an ARR-based reasoning built over a component oriented constraint based model. Only model of normal behavior is considered. ARRs are obtained by combining constraints from various component models. A component C is involved in an ARR if the ARR derives from a constraint of C, in this case the ARR may be sensitive to a fault that affects C's behavior. A fault signature matrix is built, matching the ARRs in lines versus the faults (single or multiple) in columns, as illustrated in figure 2.4. The ARR based diagnosis reasoning is compared to the conflict based diagnosis reasoning, revealing several hypotheses that are made on each side.

The paper [Cordier *et al.*, 2004] provides a definition for two types of exoneration assumption.

Definition 2.7 (SBS Exoneration assumptions) *The* component exoneration *assumption states that if the correct behavior model is satisfiable in a given context, then the component is assumed to be correct.*

The ARR exoneration *assumption states that if an ARR is satisfied, then all the components that support it are correct.*

The two assumptions are slightly different. The first assumption deals with a single component, and assesses that when faulty, the component cannot behave

the same way as when normal. The second assumption deals with a set of components, and also requires that the operating condition of fault condition of one component does not mask a fault in another component.

The ARR exoneration assumption is generally implicitly made in FDI approaches. It can be released by modifying the signature matrix. A 1 in the signature matrix means that a fault in the component necessarily results in the corresponding ARR being violated, this corresponds to the exoneration case of the ARR. The symbol X is introduced and means that a fault in the component may or may not result in the corresponding ARR being violated. This allows the designer to exonerate only some components, or some ARRs, as illustrated in figure 2.5.

A set of properties is given that ensures that the components incriminated by the ARR based approach are the same as the components incriminated by the conflict based approach. The hypotheses (System Representation Equivalence, ARR-d-completeness, ARR-i-completeness) that lead to this equivalence deal with equivalence of the system models and properties on the set of ARRs. The equivalence also relies on the relaxation of the exoneration assumption.

	F_{A1}	F_{A2}	F_{M1}	F_{M2}	F_{M3}
ARR_1	1	0	1	1	0
ARR_2	0	X	0	1	1
ARR_3	X	X	X	0	1

Figure 2.5: Fault signature matrix for the polybox with the exoneration assumption partially relaxed. Component exoneration is assumed for $M2$ and $M3$ only. ARR exoneration i sassumed for ARR_1 only.

2.1.4 Towards unified definitions

The comparison between state-based approaches to diagnosability described in [Travé-Massuyès *et al.*, 2006b] has led to various unified definitions related to diagnosability. These definitions are important, since they strongly influence the positions adopted in this thesis (defined in section 3).

Definition 2.8 (SBS Faults and observables) *The faults, single or multiple, are gathered into the set $\mathcal{F} = \{f_i\}$. Faults themselves are defined as the set of atomic malfunctions, hence, a single fault can be part of a multiple fault.*

The notation obs denotes the tuple value returned by the sensors. The set OBS_{f_i} contains all the possible tuples consisting of observed variable values under the fault f_i.

Let us assume that the model only describes the normal behavior of the system.

In the case of FDI approaches, *obs* is generally the value tuple of the residuals, not the observable variables. For DX approaches, *obs* contains the tuple value of the variables that correspond to the sensor values.

Definition 2.9 (SBS Diagnosis candidate) *For a given observation obs, $f_i \in \mathcal{F}$ is a* diagnosis candidate *if and only if obs $\in OBS_{f_i}$.*

f_i is a minimal *diagnosis candidate if and only if for each $f_j \subset f_i$, f_j is not a diagnosis candidate.*

In the literature, the terms "diagnosis candidates" and "diagnosis" (as defined in definition 2.5) are used without distinction.

Definition 2.10 (SBS Discriminability) *Two faults f_i and f_j are said to be* strongly discriminable *if and only if for any obs, when f_i is among the diagnosis candidates, f_j never is, and conversely. In other words, $OBS_{f_i} \cap OBS_{f_j} = \emptyset$.*

Two faults f_i and f_j are said to be non discriminable *if and only if for any obs, when f_i is among the diagnosis candidates, then f_j also is, and conversely. In other words, $OBS_{f_i} = OBS_{f_j}$.*

A fault f_i is said to be weakly discriminable *from a fault f_j if and only if, there exists an obs such that f_i is a diagnosis candidate, and f_j is not. In other words, $OBS_{f_i} \setminus OBS_{f_j} \neq \emptyset$. A pair of faults (f_i, f_j) is said to be weakly discriminable if and only if f_i is weakly discriminable from f_j* or *f_j is weakly discriminable from f_i, i.e. if and only if (f_i, f_j) is not non discriminable.*

Discriminability is a very important concept, that will be used throughout this thesis. It is the underlying property of detectability and diagnosability. Actually, detectability can be defined as discriminability from the normal behavior mode. Diagnosability is defined as follows.

Definition 2.11 (SBS Diagnosability) *A system is* strongly diagnosable *if and only if every pair of faults is strongly discriminable:*

$$\big(\forall f_i, f_j \in \mathcal{F}, f_i \neq f_j\big), OBS_{f_i} \cap OBS_{f_j} = \emptyset$$

A system is weakly diagnosable *if and only if every pair of faults is weakly discriminable:*

$$\big(\forall f_i, f_j \in \mathcal{F}, f_i \neq f_j\big), OBS_{f_i} \setminus OBS_{f_j} \neq \emptyset \vee OBS_{f_j} \setminus OBS_{f_i} \neq \emptyset$$

The definitions proposed in this article are very important, however the distinction between the individual component malfunction, and the abnormal behavioral mode of the system is not done, both are called faults. This impacts the ability to define formally the link between faults and components, or the set of observations for the normal mode.

This problem is addressed in this thesis, and the definitions proposed in section 3 introduce the notion of fault mode that allows one to make the distinction between the fault as an individual malfunction, and the fault as the current behavior mode of the system that results from a possibly empty set of malfunctions.

2.2 Event-based approaches

Most event-based approaches use discrete event formalisms like finite state automata and Petri nets. In these approaches, diagnosis reasoning is based on the occurrence of discrete *events*, these events correspond to the occurrence of a fault, or the occurrence of any other change in a particular behavior. The occurrence of an event can modify the state of the system.

In classical event-based systems, the language used for describing system states is limited: in automata states are simply enumerated, and Petri nets use a marking, i.e. a finite set of natural integers, for characterizing the system states. These constraints force the model designer to adopt a high abstraction level. Some enrichments of these formalisms, like colored Petri nets, allow the designer to stay at a low level of abstraction by providing a more expressive language for describing system states.

2.2.1 Automata

This section presents the approaches for modeling the system that make use of automata, more precisely finite labelled transition systems. In this domain, the approach in [Sampath *et al.*, 1995] is still a reference, from which interesting enrichments or adaptations have been made, in order to deal with distributed systems, or in order to enrich the means to describe faults.

Definition 2.12 (Finite state automaton) *A finite state automaton is defined by a tuple* $G = (Q, E, T, q_0, A)$ *where:*

- Q *is a finite set of states.*
- E *is a finite set of events.*
- $T \subseteq Q \times E \times Q$ *is the transition relation. When* $(q_1, e, q_2) \in T$*, also noted* $q_1 \xrightarrow{e} q_2$*, this means that the system can evolve from the state* q_1 *to the state* q_2 *under the occurrence of the discrete event* e*.*
- $q_0 \in Q$ *is the initial state.*
- $A \subseteq Q$ *is the set of accepting or final states.*

G *is* deterministic *if and only if* T *is a (partial) function* $T : (Q \times E) \to Q$*.*

The formalism of languages is closely linked to finite state automata, an dit is necessary to introduce both since the following approaches rely on both formalisms. We call E^* the set of all finite length sequences of events of E, including the empty sequence ε. E^* is closed under the sequence concatenation operator. E^* is a *language* over the alphabet E, and so is any subset of E^*. An element of $s \in E^*$ is called a *sequence* or *word*. The notation $s(i)$ denotes the symbol of E at index i in the word s, beginning at index 0. If $e \in E$ and $s \in E^*$, $e \in s$ if and only if e appears at some index in s.

In particular, E is a subset of E^*. Consequently it is a language, even if all the sequences it contains are only one event long. In the following, the notation $e \in E$ can indifferently refer to the event e or to the singleton sequence of events that only contains e, and E indifferently refers to the set of symbols or the language.

Operations over languages include classical set operations: union, intersection, complement and difference, as well as concatenation, noted . and defined by $L.L' = \{ll', l \in L \wedge l' \in L'\}$. Self concatenation is noted $L^1 = L$ and $L^i = L.L^{i-1}$. L^* is the closure of L under self concatenation that also contains the empty word ε, also called Kleene closure.

For example, E^*_{uo} contains all the finite sequences of unobservable events. $E^*_{uo}.E_o$ contains all the finite sequences that result from the concatenation of a finite sequence of unobservable events and an observable event.

Back to automata, the notation $\longrightarrow$ is extended to sequences of events. In the automaton G, we have, for any *sequence* $s \in E^*$:

$$q \xrightarrow{s} q' \quad \Longleftrightarrow \quad q \xrightarrow{s(0)} q^1 \xrightarrow{s(1)} q^2 \xrightarrow{s(2)} \dots q^{n-1} \xrightarrow{s(n)} q'$$

Definition 2.13 (Regular language, acceptation) *A language $L \subseteq E^*$ is* regular *if and only if:*

- *$L = \emptyset$, or*
- *$L = \{\varepsilon\}$, or*
- *L contains a finite set of words of E^*, or*
- *L is the union, or intersection, of two regular languages L' and L'', or*
- *L is the Kleene closure, or complement in E^*, of a regular language L'.*

The automaton $G = (Q, E, T, q_0, A)$ accepts the language $L(G) \subseteq E^*$ defined by:

$$L(G) = \left\{ s \in E^* \;\middle|\; \exists q_A \in A, q_0 \xrightarrow{s} q_A \right\}$$

Property 2.1 *A language is regular if and only if there exists an automaton that accepts it.*

An automaton G is *live* if and only if $\forall q_i \in Q, \exists e \in E, \exists q_j \in Q, (q_i, e, q_j) \in T$, i.e. there is an outgoing transition from every state.

A language L is *live* if and only if every word of L is a prefix of at least one other word in L.

If all states are accepting states, i.e. $A = Q$, then G can be defined by the tuple (Q, E, T, q_0). In this case, $L(G)$ is prefix closed, i.e. every prefix of every word in $L(G)$ also belongs to $L(G)$.

The concept of *trajectory*, or *scenario* refers to a sequence of events that is accepted by an automaton. It is used to distinguish arbitrary event sequences (words) from sequences that actually denote a system behavior (trajectory).

The reference approach

The most well known approach for diagnosis and diagnosability analysis in event-based approaches is the so-called "diagnoser approach" described in [Sampath *et al.*, 1995, Sampath *et al.*, 1996]. The formalism and the attached semantics for describing faults and observations are very commonly used in the EBS diagnosis community. In this approach, the system is modeled as a finite state automaton G.

In the diagnosis approach, the automaton used for modeling the system is deterministic and accepts a prefix-closed, live language. In some approaches, the system model is described directly as a regular language L_{sys} instead of defining it through an automaton. The set of events is partitioned into the set of *observable events* noted E_o and the set of *unobservable events* noted E_{uo}. The set of *fault events*, noted E_f, is a subset of E_{uo}. Faults are assumed to be permanent. Moreover, the system is supposed to produce observations regularly, this assumption is expressed formally as the absence of cycles composed of unobservable events in the automaton G.

Diagnosability definition relies on an operation called projection on observable events that removes unobservable events from a word or trajectory.

Definition 2.14 ($\mathcal{P}_{OBS}$ and $\mathcal{P}_{OBS}^{-1}$) *Let e_i be an event, ε be the empty word, s, t be trajectories and ω be a trajectory containing only observable events. The projection on observable events $\mathcal{P}_{OBS}$ applies to languages:*

$$\mathcal{P}_{OBS} : E^* \rightarrow E_o^*$$

$$\forall e \in E \cup \{\varepsilon\}, \quad \mathcal{P}_{OBS}(e) = \begin{cases} \varepsilon & \text{if } e \notin E_o \\ e & \text{if } e \in E_o \end{cases}$$

$$\forall s, t \in E*, \quad \mathcal{P}_{OBS}(s.t) = \mathcal{P}_{OBS}(s).\mathcal{P}_{OBS}(t)$$

The inverse projection $\mathcal{P}_{OBS}^{-1}$ is defined as:

$$\mathcal{P}_{OBS}^{-1} : E_o^* \rightarrow L_{sys}$$

$$\mathcal{P}_{OBS}^{-1}(\omega) = \{s \in L_{sys}, \mathcal{P}_{OBS}(s) = \omega\}$$

This framework allows one to define diagnosability for discrete event systems. Informally, a system is diagnosable if and only if every fault event is necessarily associated to a bounded observable event sequence that could not have been generated in its absence. Formally, for any event $e \in E$, let $L_{\rightarrow e}$ be the language containing all the words of L_{sys} that end with e, $L_{\rightarrow e} = E^*.\{e\} \cap L_{sys}$.

Definition 2.15 (EBS Diagnosability) *An EBS is (strongly) diagnosable if and only if:*

$$\forall e_{f_i} \in E_f, \exists n_i \in \mathbb{N}, \forall s \in L_{\rightarrow e_{f_i}}, \forall t \in E^* \mid (st \in L_{sys}),$$
$$\|t\| \geq n_i \Rightarrow \left(\forall u \in \mathcal{P}_{OBS}^{-1}(\mathcal{P}_{OBS}(st)), e_{f_i} \in u\right)$$

In [Sampath *et al.*, 1995, Sampath *et al.*, 1996], diagnosis and diagnosability analysis rely on the construction of an automaton called the diagnoser, used to monitor the system and to analyze diagnosability. The diagnoser states contain a label indicating the current diagnosis, and the diagnoser events correspond to the system's observable events. When an observable event is received by the system supervisor, the corresponding transition is fired in the diagnoser, which brings it into a state containing the new current diagnosis, as illustrated in figure 2.6.

Definition 2.16 (Diagnoser) *The set of diagnoser* labels *L is defined as $L = 2^{E_f}$. The label N is commonly used instead of $\emptyset$.*

A diagnoser *for the system automaton $G = (Q, E, T, q_0)$ is the automaton $G_d = (Q_d, E_d, T_d, q_{d0})$ where:*

- *$Q_d \subseteq 2^{Q \times L}$, each state of the diagnoser contains a list of couples (system state, label).*
- *$E_d = E_o$, only observable events are considered in the diagnoser.*
- *T_d is the transition relation and uses the notation $\xrightarrow{e_d}_d$. We have $q_d \xrightarrow{e_d}_d q'_d$ if and only if:*

$$\Big(\forall (q_i, l) \in q_d\Big), \Bigg(\Big(\exists s \in E_{uo}^*.\{e_d\},\ \exists q_j \in Q, q_i \xrightarrow{s} q_j\Big) \Longleftrightarrow$$
$$\Big(\exists (q'_i, l') \in q'_d, q'_i = q_j \text{ and } l' = l \cup \big(E_f \cap \{s(0), s(1), \ldots s(n)\}\big)\Big)\Bigg)$$

- *$q_{d0} = \{(q_0, N)\}$ is the initial diagnoser state.*

A state of the diagnoser in which some labels mention a fault event e_f and other labels do not mention it is said to be e_f-uncertain. Intuitively, a cycle of e_f-uncertain states in the diagnoser would mean that after the occurrence of e_f, the system may enter a cycle and execute a sequence of events of any arbitrary length without exhibiting an observation that would characterize the fault. This intuitive property has proved to be sufficient for diagnosability, but not necessary: in some systems, even though the diagnoser contains a cycle

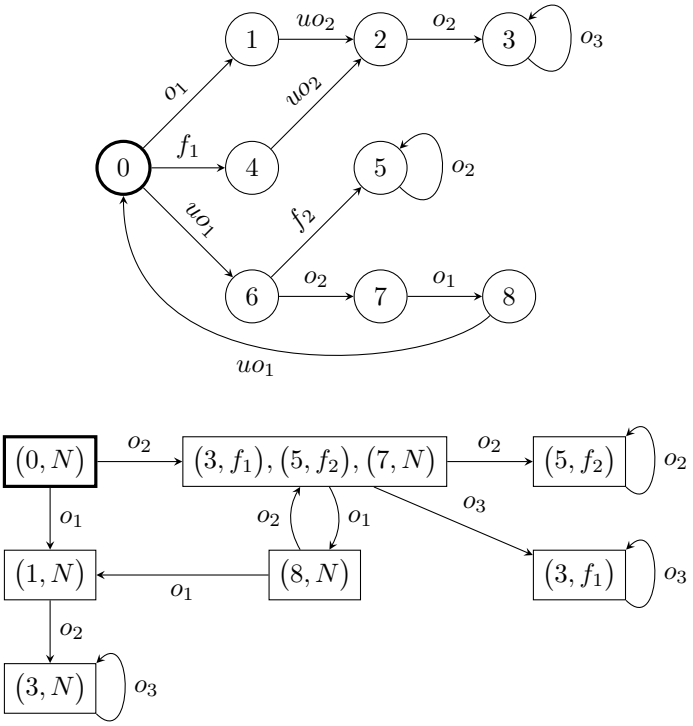
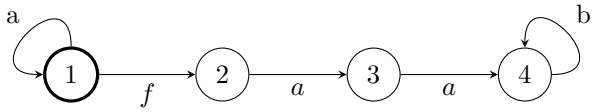


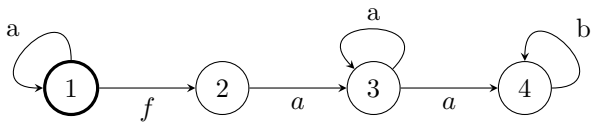
Figure 2.6: A diagnosable system and its diagnoser. Events f_i are fault events, events uo_i are unobservable normal events, and events o_i are observable.

of e_f-uncertain states, the system is still diagnosable [Sampath *et al.*, 1995]. The true equivalent property is more subtle, as illustrated in figure 2.7: an *indeterminate cycle* in the diagnoser is a cycle containing only e_f-uncertain states, that corresponds to both a normal cycle and a faulty cycle in the system. In some cases, a cycle of e_f-uncertain states corresponds to a non-cyclic sequence of events in the system, such cycle is not an indeterminate cycle. This restriction allows one to state that a system is diagnosable if and only if its diagnoser contains no indeterminate cycle.

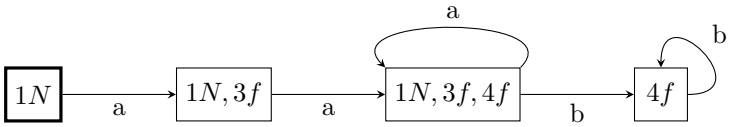
A particular interesting aspect of this approach is that the diagnosability of each fault can be evaluated separately. Some faults are diagnosable, and some are not, the system is diagnosable when all faults are diagnosable. However, it suffers from two major drawbacks: the diagnoser, even if it can be computed offline at design time, is extremely complex to build and requires tremendous amounts of memory. Moreover, diagnosability information is poor, since the designer is only aware of which faults are diagnosable and which are not, but does not give the reasons why, or the scenarios in which some faults are not diagnosable.



System n°1



System n°2



Common diagnoser for both systems

Figure 2.7: Diagnoser and diagnosability: a diagnosable system and an undiagnosable system that have the same diagnoser. Events a and b are observable, event f is a fault event. The diagnoser state $1N, 3f, 4f$ may form an indeterminate cycle if it corresponds to a normal cycle and a faulty cycle in the system.
In system n°1, the diagnoser cycle over the state labeled $1N, 3f, 4f$ only corresponds to a normal cycle over state 1. It is not an indeterminate cycle, and system n°1 is diagnosable.
On the contrary in system n°2, the diagnoser cycle corresponds to both a normal cycle over state 1 and a faulty cycle over state 3. Consequently, this is an indeterminate cycle, and system n°2 is not diagnosable.

Complexity and efficiency The major drawback of this approach is the computational complexity. The size of the diagnoser is exponential in the number of states of the system model, which makes it difficultly tractable in real applications. This problem was addressed in parallel in [Jiang *et al.*, 2001, Yoo and Lafortune, 2002] and polynomial time algorithms are provided. Those algorithms are based on the construction of automata dedicated only to diagnosability checking, az opposed to the diagnoser that also allows on-line supervision.

The problem of diagnosability analysis has been translated into a satisfiability problem in [Rintanen and Grastien, 2007]. This translation allows one to reuse powerful tools to perform efficient diagnosability analysis.

Distributed approach

The two drawbacks of Sampath's approach (diagnoser size and poor feedback to the designer) were addressed in an adaptation of the approach [Pencolé, 2005]. In this approach, the system is modeled by *several* communicating automata, each automaton representing a part of the system. The formalism is extended by defining a new kind of unobservable events called communication events. A communication event is always present in at least two subsystems, and is always fired synchronously by all the communicating automata that have it.

The choice of using communicating automata is particularly suited for distributed systems, since each communicating automaton represents the model of a local component. This approach also addresses the problem of representing parallelism, that requires a lot of resources in a classical automaton. Hence a system model is often much smaller when expressed with communicating automata, as illustrated in figure 2.8. This formalism is still as expressive as a classical automaton, and the full system model can be obtained by synchronizing the automata on the set of communication events.

Pencolé's approach to diagnosability analysis takes advantage of the decomposition of the system into subsystems and builds *interactive diagnosers*, each one is in charge of one subsystem, and all interactive diagnosers communicate one with another in order to provide a global diagnosis for the whole system. The size of the interactive diagnosers is much smaller than the size of the diagnoser for the whole system, and makes tractable their construction. Diagnosability is checked by synchronizing each interactive diagnoser with itself on observable events, in order to create a *twin diagnoser*, and by synchronizing twin diagnosers one with another, focusing on the *undiagnosable scenarios*. At the end of the process, these undiagnosable scenarios are returned to the designer in order to indicate precisely which behavior(s) of which component(s) to modify in order to make the system diagnosable.

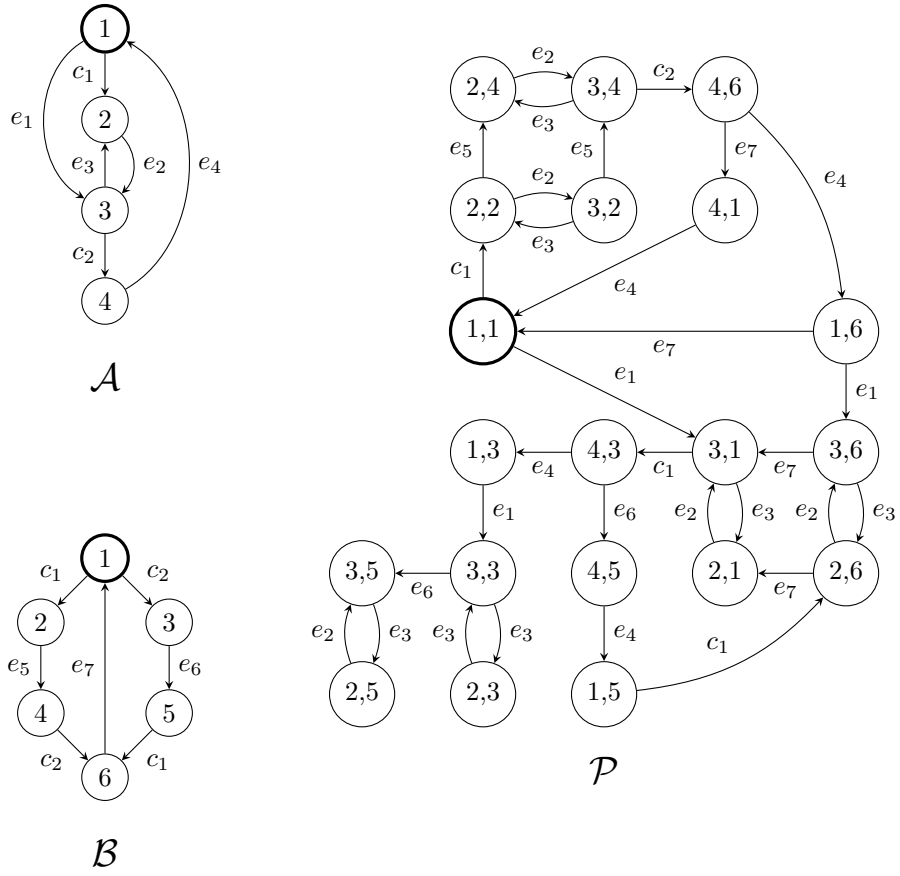


Figure 2.8: Synchronous product of two automata: the automaton $\mathcal{P}$ is the synchronous product of automata $\mathcal{A}$ and $\mathcal{B}$ over the set of communication events $\{c_1, c_2\}$. Initial states are indicated by a thick border.

Enriching fault specifications

In the two previous approaches, a fault is represented by a discrete event. Although in many cases this is sufficient, some undesirable behaviors in discrete event systems need a richer specification. The approach described in [Jéron *et al.*, 2006] uses supervision patterns in order to monitor the system. These patterns are described by a finite labelled transition system that uses the same events than the system model. The pattern's final states are reached when the undesired behavior has been recognized in the system. Such patterns can simply represent the occurrence of a fault, but they can also represent complex specifications, like "event e_1 must not occur n times in a row without the occurrence of event e_2". They simplify the modeling phase and allows to manipulate much smaller models, as illustrated in figure 2.9.

Observations are represented by observable events, which leads to the construction of one diagnoser per supervision pattern, the main step being the synchronization of the pattern with the model. Algorithms are provided to check the diagnosability of the pattern, and for efficient online diagnosis.

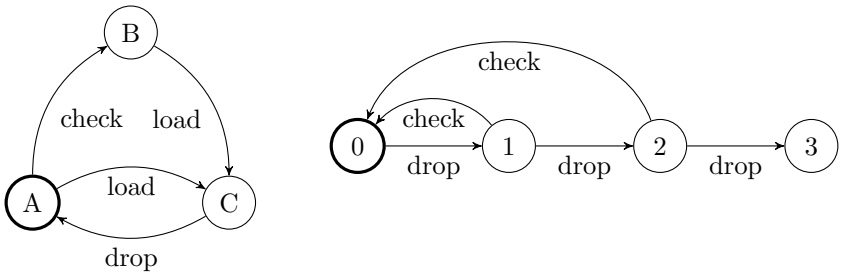
In terms of expressivity, the approach developed in [Jiang and Kumar, 2002] allows even richer fault specifications than the supervision patterns approach, since faulty behaviors are described by means of Linear Temporal Logic formula (LTL). The classical automaton described in 2.12 is enriched with a set of atomic propositions AP and a state labelling function $L : Q \rightarrow 2^{AP}$. Observations are represented by a mask function on the language generated by the automaton, which erases unobservable events from the language. This is equivalent to partitioning the set of events between observable and unobservable events.

Fault specifications are built by describing propositions that should be true in normal operation. These propositions are constructed by combining elements of AP using the usual logic operators $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$, and the following temporal operators:

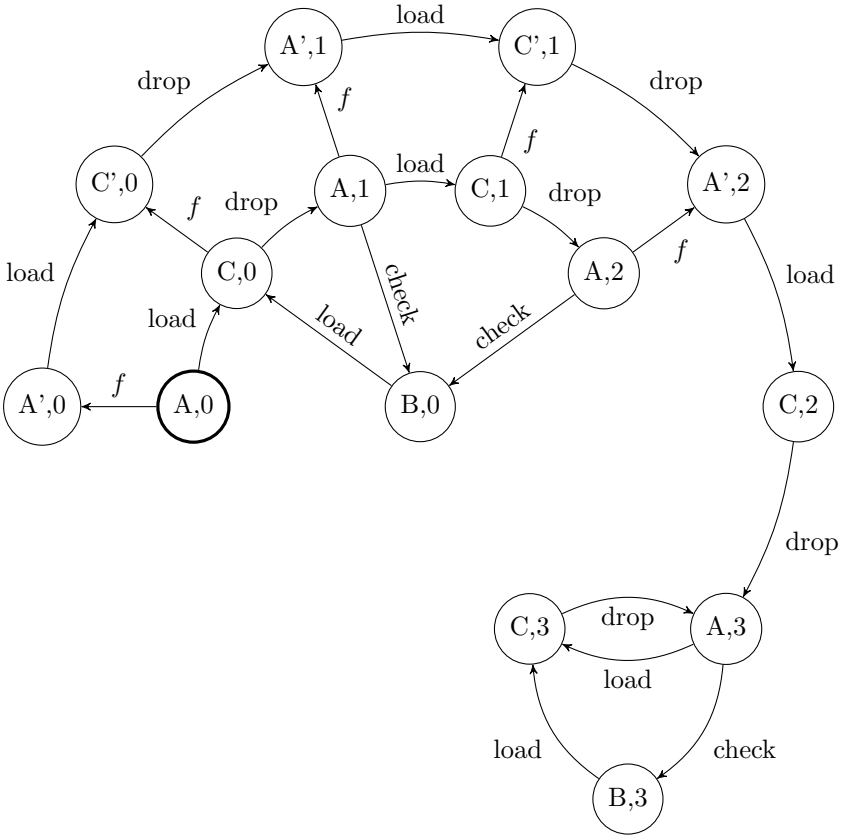
- Xp, next: p holds in next state.
- Gp, globally or always: p holds in all the future states.
- Fp, finally: p holds in some future state.
- $p_1 U p_2$, until: p_2 holds in some future state q, and p_1 holds in all states preceding q.

Linear Temporal Logic only refers to *one* execution of the system, i.e. one path or trajectory, possibly infinite. A proposition holds for a path if and only if it holds for the first state of the path.

For example, let AP contain the propositions p_1 ="the pipe is clogged" and p_2 ="the tank is clean". These propositions are used to specify the following specifications:



A simple system and its supervision pattern



A model for the diagnoser approach for the same system

Figure 2.9: Supervision patterns simplify the modeling. A robot meant to transport objects needs to perform a routine check every time it travels a predefined distance, that allows it to transport no more than 2 objects. The events *drop* and *check* are observable. A fault in the distance counter will make the robot ignore the routine check, and risk further damage. After over travelling, a routine check is not enough to restore the robot state.
Above, the model of this system, and a supervision pattern for the fault. Below, a model for the diagnoser approach, that appears to be much larger and harder to maintain.

- $f_1 = G\neg p_1$: the pipe never gets clogged (it is true in all states that not the pipe is clogged).
- $f_2 = GFp_2$: the tank is clean from time to time (it is true in all states that in a future state the pipe is clean), meaning the sequences of states in which the pipe is not clean are all finite.

f_1 is called a *safety* specification that requires that something bad (pipe clogged) never happens. On the opposite, f_2 is a *liveliness* specification requiring that something good (tank clean) keeps happening. It is claimed that levelness statements bring a significant enrichment to the fault specifications for diagnosis, in comparison to the approaches described in the previous sections. However, diagnosing that "the tank has ceased to be cleaned from time to time" is impossible from a finite observation, since the tank may always be cleaned in the future. The only way to reach such a diagnosis would be to prove that something bad prevents the tank from being cleaned, which falls back to a safety specification that can be expressed in the diagnoser approach. In this approach, diagnosability is defined in such a way that faults related to a likeness specification cannot be diagnosable. This seems consistent with the previous consideration, but then the utility of specifying faults with a aliveness statement seems very limited.

Although this approach does not enrich the scope of abnormal behaviors that can be diagnosed, the model allows the use of model checking methods that provide efficient algorithms for checking diagnosability, and building one diagnoser per fault specification. Moreover, this richer language may help making the system modeling easier.

2.2.2 Petri nets

The Petri net formalism is a very powerful way to model automated systems, in particular when parallelism and synchronization are important.

Definition 2.17 (Petri Net) *A Petri Net is defined by a tuple* (Q, T, A, M_0) *where:*

- Q *is the set of places.*
- T *is the set of transitions.*
- $A : (Q \times T) \cup (T \times Q) \to \mathbb{N}^+$ *describes the Petri net arcs.*
- $M : Q \to \mathbb{N}$ *is the initial marking.*

More details on Petri nets can be found in [Peterson, 1981]. At each moment of its execution, the state of the modeled system is indicated by the current marking. When adding labels to transitions, it is possible to generate the graph of reachable markings which is equivalent to the automata used in the previous section unless some infinite markings are reachable. However, asynchronous

communications and parallelism are much easier to model with Petri Nets than with automata, and in general the graph of reachable markings is much larger than the original Petri net.

Translation of the reference approach

The approach described in [Haar *et al.*, 2003] appears to be a translation of the reference approach using automata to the Petri net formalism. The purpose is to be applied to highly asynchronous distributed systems; this is the kind of systems in which Petri nets are highly more efficient than automata as a modeling language. A labelling function $\lambda : T \rightarrow \Sigma$ labels the transitions with symbols. The null symbol is accepted $\varepsilon \in \Sigma$, and the set of unobservable transitions is defined as $UO = \lambda^{-1}(\varepsilon)$, while the set of observable transitions as $O = T \setminus UO$. It is assumed that two observable transitions cannot be labelled by the same symbol. Each symbol (except ε) corresponds to a distinct observation. Faults are also represented by transitions, $\Phi \subseteq UO$ is the set of fault transitions.

The regular language generated by the automaton is translated into the set of configurations of the Petri net, this notion is similar but takes into account the asynchronous aspect. Notions of "length" and "last event" of a word are translated into "height" and "maximal event" of a configuration. Two words having the same observable projection are translated into two configurations being isomorphic with respect to the observable equivalence relation. The definition of diagnosability states that if a configuration κ has a fault event[1] as a maximal event, then any configuration that continues κ over a given finite height is isomorphic (w.r.t. the observable equivalence relation) with only configurations containing the same fault event.

Another translation

Another approach [Wen and Jeng, 2004] is derived from the standard automaton-based definition, but is not as close to the original definition. Faults are modeled by faulty transitions, as the set of transitions T is partitioned into normal transitions T_N and faulty transitions T_F. The modeling of observations is quite different from the automaton approaches, as the set of places P is partitioned into observable places P_o and unobservable places P_{uo}.

The approach relies on the construction of a diagnoser under the form of an automaton. The definition of the automaton, using label propagation and range functions, is a straightforward adaptation of the original definition of the diagnoser given in [Sampath *et al.*, 1995]. The definition of indeterminate states and cycles in the diagnoser also follows the same principles. Diagnosability is not stated as a property of the system model, but as a property of the diagnoser. The problem of indeterminate cycles mentioned in figure 2.7 has been taken care of in this approach.

[1] Configuration events are linked to the Petri net transitions by an homomorphism.

Interpreted Petri net-based approach

Another approach described in [Ramírez-Treviño *et al.*, 2004] makes use of Interpreted Petri nets for diagnosis and diagnosability analysis. An Interpreted Petri net is defined by the tuple $(N, \Sigma, \lambda, \varphi)$ where:

- N is a Petri net.
- $\Sigma = \{\alpha_1, \alpha_2, \ldots\}$ is the input alphabet of the net, α_i being an input symbol.
- $\lambda : T \rightarrow \Sigma \cup \{\varepsilon\}$ is the labelling function. ε labels an internal event. Two transitions with the same input places and same input weight may not have the same label (except ε).
- $\varphi : R(N, M_0) \rightarrow (\mathbb{Z}^+)^q$ is the output function, where $R(N, M_0)$ is the set of reachable markings, and q is the size of the output vector.

Interpreted Petri nets can be synchronized between one another, allowing one to model components separately, and to build the full model automatically.

Observations are characterized by sequences of input-output symbols, defined as sequences of pairs of input and output that are consistent with the system: $\omega = (\alpha_1, y_1)(\alpha_2, y_2)\ldots(\alpha_n, y_n)$, where $\alpha_i \in \Sigma \cup \{\varepsilon\}$ is the current input when the output changes from y_{i-1} to y_i. The set of all input-output sequences that can be generated by the Interpreted Petri net IPN from a marking M is denoted $\Lambda(IPN, M)$. $\Lambda^k(IPN, M)$ denotes the set of input-output sequences of length k, and $\Lambda_B(IPN, M)$ denotes blocking input-output sequences: after generating one of these sequences, the Interpreted Petri net is in a deadlock (no transitions or only self loop transitions are enabled).

Faults are characterized by faulty places. Moreover, there is no distinction between several types of faults: places may be normal or faulty. The problem of fault diagnosis is then a problem of detection, and if needed isolation of the sub model containing the marked faulty place. The set of reachable markings $R(N, M_0)$ is hence partitioned into faulty markings and normal markings. A marking is faulty when at least one faulty place is marked. The definition states that a system is diagnosable if and only if there exists a $k \in \mathbb{N}$ such that for every faulty marking M_f and every normal marking M_n, we have: $(\Lambda^k(IPN, M_f) \cup \Lambda_B(IPN, M_f)) \cap (\Lambda^k(IPN, M_n) \cup \Lambda_B(IPN, M_n)) = \emptyset$. An algorithm is provided that allows to check a sufficient condition for diagnosability.

This approach is limited by the fact that only one type of fault is considered, although it seems to be extensible to several types of faults. However, the inability to check necessary and sufficient criteria for diagnosability is disappointing. Moreover, using places especially to express the presence of faults and not representing the system state may seem strange for non Petri net users.

2.3 Hybrid systems

In an attempt to combine the expression power of both state-based and event-based approaches, diagnosability has been addressed for systems modeled with hybrid formalisms.

In [Biswas *et al.*, 2006], the formalism of Real Time Hybrid Systems (RTHS) is used. In this formalism the behavior of the system is described by the notion of trace, which is a sequence of transitions that capture both continuous and discrete changes. The condition for diagnosability is expressed as a reachability condition, and tested via an adaptation of the diagnoser defined in section 2.2.1 to RTHS.

In [Fourlas *et al.*, 2002], Hybrid Input/Output Automata are used. The hybrid behavior is described by an hybrid execution which is an alternating sequence of continuous evolutions called trajectories and actions. Observations are only achieved by the measurement of transition guards over continuous variables. Input actions are not considered for diagnosis.

Two other approaches rely on the abstraction of continuous dynamics and fault indicators into a set of discrete events [Bayoudh *et al.*, 2008, Daigle *et al.*, 2008]. Event-based diagnosability is then performed in a classical way, according to the diagnoser approach defined in section 2.2.1. In [Daigle *et al.*, 2008], the system is represented by means of hybrid bond graphs, and discrete events are created to represent the change of value of residuals. In [Bayoudh *et al.*, 2008], the system is modeled by a hybrid automaton, and discrete events model the changes of overall signature, which seems to limit the combinatorial explosion due to the number of generated events.

In [Bayoudh *et al.*, 2006, Bayoudh *et al.*, 2008], a system is modeled by means of a hybrid automaton. Diagnosability is defined in several steps:

1. Multimode systems are defined.

2. Diagnosability for multimode systems is defined as an enrichment of FDI approaches.

3. Hybrid systems are defined as multimode systems with constraints over the transitions between modes.

4. The continuous dynamics are abstracted under the form of events representing the modification of the values of the residuals. The system is then represented under the form of an automaton, on which the diagnoser approach is applied.

A hybrid system is modeled by a hybrid automaton $S = (V, Q, E, T, R, (v_0, q_0))$ where:

- (Q, E, T, q_0) is a finite state automaton with observable events E_o, unobservable events E_{uo} and fault events E_f. The states of the automaton are called *modes*.
- V is a set of continuous variables, partitioned into the observable variables O and unobservable variables X.
- R is a set of constraints on $V \cup Q$. Each constraint R_i is associated to a mode of the system.
- (v_0, q_0) is the initial condition.

In a multimode system, there is no constraint over the transitions between modes. It is a hybrid model in which $E = \emptyset$ and $T = Q \times Q$. Some of these system modes are normal, and some are faulty.

Applying the parity space approach to each mode of the multimode system generates one set of residuals per mode. When the system is in a mode q_i, the residuals generated for this mode are equal to 0. However when the system is in mode q_i, the residuals generated for other modes q_j may become equal to 1. The signature is applied to modes instead of only faults. Each mode q_i of the system is associated to its signature, that indicates the value tuple for all the residuals generated from all the modes.

By denoting OBS_{q_i} the set of all the observations that can be recovered in mode q_i, the strong diagnosability definition is similar to definition 2.11 page 43:

Definition 2.18 (Multimode system diagnosability) *In a multimode system, a mode q_i is diagnosable if and only if it is discriminable from any other mode q_j:*

$$\forall q_i, q_j \in Q, OBS_{q_i} \cap OBS_{q_j} = \emptyset$$

The translation to hybrid system is done by transforming the residual switches into events, thus creating "diagnosis automata" that describe the behavior of the FDI diagnosis system. These automata are synchronized with the mode automaton (Q, E, T, q_0) to create the behavior automaton.

Definition 2.19 (Hybrid system diagnosability) *A hybrid system S is diagnosable if and only if its behavior automaton is diagnosable according to definition 2.15.*

Hybrid system diagnosability analysis is still a recent research domain, and this thesis focuses mainly on state-based and event-based approaches, that can be considered as mature. However, the unification of diagnosability analysis approaches should undoubtedly apply to hybrid approaches with an adapted point of view.

Chapter 3

Unified definitions

The approaches for diagnosability analysis described in the previous chapter 2 are very different one from another. Still, all approaches reference some common concepts that can be used as a basis for a global, model-independent view on diagnosability analysis. This chapter summarizes the hypotheses and points of view adopted in each approach, and describes the position that is adopted in this thesis. The resulting definitions account for all the diagnosability approaches, and claim to provide a unifying point of view.

3.1 Faults and fault modes

Definition 3.1 (ISO Fault) *In document ISO/CD 10303-226 of the* International Organization for Standardization*, a fault is defined as an abnormal condition or defect at the component, equipment, or sub-system level which may lead to a failure.*

The fundamental concept in diagnosis and diagnosability, shared by all the approaches, is the concept of fault. In definition 3.1, a failure occurs when the system does not fulfill the purpose for which it has been designed and constructed. This informal definition has been interpreted in various ways according to the existing modeling approaches.

This section presents the different meanings in the existing model-based diagnosis approaches for the concept of fault. The position used in the following chapters of this thesis is presented and debated.

3.1.1 Different interpretations of a fault

The concept of fault bears the same name in most diagnosis approaches, although comparison shows that it is subject to significantly different interpretations according to the point of view.

In FDI approaches, the assumption of the single fault is common. In this case, it is assumed that two components in the system may not be in abnormal condition at the same time. Each fault is a different cause for the system not to fulfill its purpose. However, in many approaches this assumption is relaxed, and the system may be subject to "multiple faults". The relaxation of this assumption makes the diagnosis approach more realistic in some application domains, but complexifies the diagnosis reasoning and makes diagnosability much harder to achieve, especially since sensors may be faulty and provide erroneous observations to the supervisor. As a compromise between the validity of the model and diagnosability, the scope of considered faults may be extended to only double faults, or triple, etc. The system behavior is influenced by the current fault.

In DX approaches, the behavior of each component depends on the presence or absence of faults inside it. The presence of these faults is characterized by one or several discrete variables called *mode variables*, each fault combination corresponds to a value for the tuple of mode variables. An important aspect is that mode variables may be binary (absence/presence of the fault), but may have a larger finite domain if needed.

In event-based approaches, faults are events, and may have occurred or not. Since in classical approaches faults are permanent, there is no distinction between trajectories containing one occurrence of a fault event, and trajectories containing several occurrences of the same fault event. The system behavior results from the set of fault events that have occurred in it.

Example 3.1 *Consider a hydraulic system in which a valve can get stuck open or stuck closed, and a pipe can get clogged.*

- FDI representation: *there are **5** different faults: 3 single faults (pipe clogged, valve stuck closed, valve stuck open) and 2 multiple faults (pipe clogged & valve stuck closed, pipe clogged & valve stuck open). The nominal mode and each fault are associated to a set of constraints that define the system behavior in the corresponding condition.*

- DX representation: ***2** mode variables are associated to the two components that may be faulty. The valve status is described by the variable:*

$$valve \in \{normal, stuck\ open, stuck\ closed\}$$

The pipe status is described by the variable:

$$pipe \in \{normal, clogged\}$$

- DES representation: ***3** fault events are defined: "pipe gets clogged", "valve gets stuck open", and "valve gets stuck closed".*

This example illustrates the diversity of the interpretations of the notion of faults in the existing approaches. According to the approaches, 2 variables, 3 faults or 5 faults are used to represent the various behaviors that can be adopted by the system.

3.1.2 Unified denomination

Having a unified view on the different approaches for diagnosability analysis requires to define concepts that allow one to express the various interpretations of a fault developed in the different communities. This section provides the definition for *fault* and *fault mode* that will be used further on in this thesis, and positions them with respect to the interpretations described in the previous section. These definitions aim to be as intuitive as possible and result from exchanges with members of the different model-based diagnosis communities as well as other research communities.

Definition 3.2 (Unified Fault) *A* fault *is an elementary unexpected event that occurs in a system or component and may alter its behavior. A system or component in which a fault has occurred is said to be faulty, and the fault is said to be present. Several different faults may occur in various orders in the same system or component.*

In the following of this thesis, faults are assumed to be permanent. This definition matches the DES interpretation of a fault, in the sense that the fault is an event, and that it is elementary. The DX interpretation also corresponds to this definition since diagnosis consists in assessing which components are faulty. The FDI interpretation is not consistent with this definition, since in FDI, saying that a fault is present implicitly means a *single* fault (except if stated explicitly as a multiple fault) thus implying that other faults are absent. The FDI interpretation actually corresponds to what is defined here as a fault mode.

Definition 3.3 (Unified Fault mode) *The system's behavioral mode that results from the occurrence of a given combination of faults is a* fault mode. *A fault mode corresponds to the presence of some faults and to the absence of the other faults. The normal system behavior corresponds to the absence of all faults, it is a fault mode called the normal mode of the system. The occurrence of a fault modifies the fault mode of the system.*

A fault mode can represent the absence of faults (normal behavior), a single fault, or a multiple fault. In DX approaches, a fault mode is defined by assigning a value to all the mode variables in the system. In event based approaches, the fault mode contains all the fault events that have occurred in the system trajectory.

3.1.3 Models for faults and fault modes

A unified definition for faults and fault modes is not useful if it cannot be translated into the various formal languages used in the different model-based diagnosis approaches. In this section, formal representations of faults and fault modes are proposed, and their links with the different existing formalisms are clarified.

A straightforward way to model faults and fault modes is to represent fault modes as sets of faults. When all faults are independent, any combination of faults is possible in the system, and the set of fault modes is the power set of the set of faults. However, as illustrated in example 3.1 page 60, some faults may not be present at the same time in the system. In the general case, the set of fault modes is a subset of the different combinations of faults.

Definition 3.4 (Fault modes set representation) *The set of all the faults that may occur in the system is denoted F_{sys}. A fault mode f is represented by the set of faults that are present when the system is in f. The set of all reachable fault modes is denoted $\mathcal{F}_{sys}$.*

$$\mathcal{F}_{sys} \subseteq 2^{F_{sys}}$$

The normal mode is a fault mode containing no fault, and denoted $\emptyset$.

This definition is easily translated to the DES formalism, since faults are represented by events. The fault mode is represented by the set of faults events that belong to the system trajectory. The FDI approach also copes easily with this formalism, since one or several sets of constraints are associated to each fault mode, independently of the formalism used to represent them.

The integration of this formalism with DX approaches is slightly more difficult. Let us first consider the case of binary independent faults: each fault is represented by a mode variable that ranges over the domain $\{normal, faulty\}$, and all combinations of faults are possible. The occurrence of a fault is characterized by the modification of the corresponding mode variable value. A fault mode is characterized by the assignment of all mode variables to a value. Each fault mode can hence be associated to a value for the tuple of all mode variables.

Definition 3.5 (Fault modes variable representation) *The system is associated to a set of n discrete* mode variables *m_i that range over finite domains D_i and characterize the presence or absence of faults. A fault mode f is defined by the assignment of all the mode variables to a value. The set of all reachable fault modes $\mathcal{F}_{sys}$ is defined as:*

$$\mathcal{F}_{sys} = \left\{(v_1, v_2, \ldots v_n), \forall i \in \{1 \ldots n\}, v_i \in D_i\right\}$$

Relaxing the assumption that faults are binary and independent makes the mapping between the set representation and the mode variable representation used in DX approaches slightly more difficult and arbitrary. For example, a mode variable with 3 values, like $\{normal, fault_1, fault_2\}$, corresponds in the

set representation to 2 faults that cannot be present at the same time. This exclusion can result from the fact that they correspond to two different ways to alter the same component, in that case the faults are the events changing the mode variable value from *normal* to $fault_1$ and from *normal* to $fault_2$. The exclusion can also result from the faults representing two levels of alteration of the same component like two degrees of wear, in that case the faults correspond to the mode variable value changing from *normal* to $fault_1$ and from $fault_1$ to $fault_2$.

If a 3-valued mode variable may represent components that would be modeled in two different ways with other approaches, a 4-valued mode variable m may be used to represent even mode kinds of components:

- 1 type of alteration with 3 levels of alteration of the component: $m \in \{$*normal, low wear, medium wear, advanced wear*$\}$.
- 2 independent faults: $m \in \{$*normal,* $alteration_1$*,* $alteration_2$*,* $alteration_1$ *&* $alteration_2\}$.
- 2 exclusive types of alteration, one of those types has 2 levels of alteration: $m \in \{$*normal,* $alteration_1$*,* $alteration_2$ *low,* $alteration_2$ *high* $\}$.
- 3 exclusive types of alteration: $m \in \{$*normal,* $alteration_1$*,* $alteration_2$*,* $alteration_3\}$.

All these different components have different set representations for their faults.

As illustrated above, there is no a priori correspondence between mode variables and fault events. The presence of each fault is characterized by one mode variable, but one mode variable may describe the presence or absence of several faults. Still, when considering all the faults at once, two different reachable combinations of faults in the system are necessarily represented by two different values of the tuple of all mode variables. Conversely, two different values for the tuple of all mode variables necessarily represent a different combination of present and absent faults, otherwise a mode variable would have two values representing the same situation. In DX representations, the current fault mode is represented by the value of the tuple of all the mode variables in the system.

3.2 Diagnosability

The definitions mentioned in section 2.1.4 are now revisited. The weaknesses plotted in the previous approaches are overcome thanks to the considerations developed in the previous section about faults and fault modes.

Definition 3.6 (Unified Fault signature) *The fault signature is a function* $Sig : \mathcal{F}_{sys} \rightarrow OBS$*. For each fault mode* f_i*, it associates the observations that can be obtained when the system is under* f_i*.*

This definition is totally consistent with the unified state-based approach definition of section 2.1.3, the only difference is that it accounts for the difference between faults and fault modes.

Diagnosis provides a set of explanations, or candidates, for the current observations. Each diagnosis candidate assesses the presence of some faults and the absence of other faults. It is consequently a fault mode.

Definition 3.7 (Unified Diagnosis candidate, Diagnosis) *A fault mode f_i is a* diagnosis candidate *for the current observation obs if and only if it can explain the current observations:*

$$obs \in Sig(f_i)$$

A diagnosis *for obs is the set of all diagnosis candidates for obs.*

Definition 3.8 (Unified Discriminability, Detectability) *Two fault modes f_i and f_j are* discriminable *if and only if:*

$$Sig(f_i) \cap Sig(f_j) = \emptyset$$

A fault mode f_i is detectable *if and only if it is discriminable from the normal mode:*

$$Sig(f_i) \cap Sig(\emptyset) = \emptyset$$

The property of detectability is not relevant for the normal fault mode. According to the previous definition, the normal mode is not detectable, since it is not discriminable from itself. Such definition can be adapted to exclude the normal mode, although we do not find it necessary.

Definition 3.9 (Unified Diagnosability) *A fault mode is* diagnosable *if and only if it is discriminable from any other fault mode.*

A system is diagnosable if and only if every pair of fault modes is discriminable:

$$\big(\forall f_i, f_j \in \mathcal{F}_{sys}, f_i \neq f_j\big), Sig(f_i) \cap Sig(f_j) = \emptyset$$

These definitions of discriminability and diagnosability are candidates for unified definitions since they are related to fault modes, and can hence apply to any modeling approach described before. However, at this point of the thesis report, the attentive reader could notice that the set of observables *OBS* has not been defined for event-based approaches. This implies that the definitions of signature, and all the definitions above do not hold.

Defining the set of observables *OBS* for event-based approaches is not trivial. It is actually the topic of the next chapter, and needs the use of additional formalisms, in particular ω-languages and Büchi automata. It is however possible, and we will prove that the definitions given in this section provide a *unified point of view on diagnosability*, for all the mentioned diagnosability approaches.

3.2.1 State of the art conclusion

The various approaches to diagnosability analysis use very different formalisms, but they address the same problem with very similar approaches.

We have unified the definition and the models for faults and fault modes, that account for all the existing modeling approaches. In order to claim that the diagnosability definitions that result from this positioning are universal, we still lack a unified definition of the set of observables.

The unified definitions described above mention a *set* of observables. This formal representation is adapted for state-based approaches, since diagnosis is performed on one observation at a time, and that it does not rely on any relation between observations.

In event-based approaches, the fact that a finite observation can be continued in order to have more information is fundamental in the definition of diagnosability. Representing the observables as a set with no particular structure seems unadapted, however the next chapter proposes a solution to this problem.

Part II

Diagnosability through Fault Signatures

It has been shown in the previous part that the various model-based diagnosis approaches are classified into two categories: state-based and event-based approaches.

Chapter 3 has shown that the fundamental concepts of fault and fault mode can be expressed and modelled in a unified way. Unified definitions for these concepts are given in definitions 3.2 to 3.5 and are adopted in the rest of this thesis.

The other definitions 3.6 to 3.9, concern diagnosis and diagnosability. They rely on a *set of observables OBS*. These definitions unify the state-based approaches in which an observation is a value for the observable variable tuple. However, for event-based approaches, it is shown that the observables are organized into languages, and that diagnosis relies on this language structure. Hence the different nature calls for further developments before these "unified" definitions apply to event-based approaches.

This part contains the core of the thesis contribution. First, the set of observables *OBS* is defined for event-based approaches, and the resulting definitions of signature and diagnosability are given. It is then proved that the definitions given in chapter 3 unify *all* diagnosability approaches.

In chapter 5, the complexity problem induced by the great number of pairs of fault modes is addressed. The concept of partial fault mode, inspired from the concept of partial diagnosis defined in DX approaches (definition 2.6, page 39) is introduced. The application of fault signatures to partial fault modes provides an efficient algorithmic approach for diagnosability analysis. An implementation of this algorithm is described in section 5.3.

Finally, the concept of fault signature is extended, in the case of state-based systems, to any property that defines a set of system states. The purpose of this extension is to allow diagnosability analysis not only for faults, but also for properties such as repair preconditions, quality of service, etc.

Chapter 4

Signatures for Event-Based approaches

As described in section 2.1, the concept of fault signature is applicable for all state-based diagnosis and diagnosability analysis approaches. In this chapter, it is shown that it can be applied to event-based approaches as well.

In state-based approaches, observations are modelled by means of observable variables. In most cases, these observable variables are discrete: in FDI approaches, the continuous variables are involved in consistency checks, whose results (check passes or check fails) are generally discrete. Diagnosis reasoning takes as input those discrete variable values in order to provide an explanation in terms of faults or fault modes. An important aspect is that during the diagnosis reasoning, each discrete variable has a unique value, which means the system does not evolve from the point of view of the diagnosis process.

State-based approaches are often considered to be based on a snapshot of the system, i.e. on its state at one precise instant. This is oversimplified, since incremental approaches exist for diagnosis, and since FDI residuals are generally numeric filters based on a memory and several observations. Reasoning on a temporal observation window (i.e. on the last n observations, n being a finite integer) may allow to increase diagnosability for many systems, but it does not solve the problem in general. There is a part of the past history that is not taken into account for diagnosis, which may contain pertinent information. Moreover, there is no diagnosability analysis approach in state-based approach that accounts for incremental with observation windows, or with all the observations in the system history.

In event-based approaches, observations are modelled by observable events. The diagnosis process takes into account the sequence of all events that are observed during the system life cycle. The consideration of the system history is the fundamental difference between state-based and event-based approaches

to diagnosis and diagnosability analysis. It impacts strongly the diagnosis reasoning, since in event-based approaches, the diagnosis process is not restarted again each time a new observation is received: the process keeps track of the system history, and updates the diagnosis candidates when new observations are received. Diagnosability analysis also takes into account the possibility of evolution of the diagnosis, and only requires the delay between the occurrence of a fault and characteristic symptoms to be bounded.

4.1 A new point of view on EBS observables

The definition of a set of observables is strongly linked to the relative lifetimes of observations and diagnosis process. In state-based approach, since diagnosis is performed on a unique observation or on the last few observations, it cannot make use of possible relations between past observations and recent ones. The observables can consequently be gathered into a simple set. In event-based approach, diagnosis is based on a finite observation of the system, in which several observable events occur. Diagnosis is based on the observed sequences of events, but each time a new observable event is received, the sequence evolves. Considering the *set* of all observable sequences of events is not sufficient for diagnosability. It is necessary to consider the *language* of all observable sequences of events, since diagnosability is strongly linked with the notion of extension of a sequence of events.

The solution described here relies on an original point of view. The concepts of *observation* and *observable*, that are not clearly distinguished in state-based approach, are clarified.

In EBS, an observation is a finite sequence of events; yet, any observation can potentially be continued. Hence, the observable, i.e. the potential observations, are considered as the infinite continuations of the finite observable event sequences. When two infinite observables are different, they differ at some finite index i. A finite observation of length i or more is then sufficient to make the distinction between these two infinite observables. From now on, we distinguish the *set of observations* $\mathcal{P}_{OBS}(L_{sys})$ that contains finite observable event sequences and the *set of observables OBS* that contains infinite observable event sequences.

4.1.1 Infinite event sequences

In the event-based context, in order to account for the possibility to refine the diagnosis after the fault has occurred, we reason in terms of the infinite sequences events that may be generated by the system. The formalisms of ω-languages and Büchi automata allow us to define and reason about such objects.

An original definition of fault signatures for event based systems is now pro-

vided. It relies on infinite words and Büchi automata, as well as the projection of behavioral event sequences on observable events.

ω-words and ω-languages

Classical languages and regular languages only contain finite words, although they may contain an infinite number of finite words. The language theory is extended to infinite words under the concept of ω-languages.

Definition 4.1 (ω-languages) *An* ω-word *is an infinite word, i.e. an infinite sequence of symbols of an alphabet* E*. It is defined as a function:*

$$\sigma : \mathbb{N} \longrightarrow E$$

The set of all ω*-words on the alphabet* E *is noted* E^ω*. Any subset* $L^\omega \subseteq E^\omega$ *is an* ω-language*.*

We consider more particularly ω-regular languages, defined as follows. L_1 and L_2 being regular languages, L_1^ω is an ω-regular language that contains all the infinite concatenations of words of L_1. Moreover, $L_2 L_1^\omega$ is also an ω-regular language that contains the concatenations of words in L_2 with words in L_1^ω. ω-regular languages are closed under set union, intersection and complement inside E^ω. No other ω-language is regular.

Infinite words cannot be accepted by normal automata, we need to define another type of automata. A Büchi automaton G^ω can be easily derived from the automaton G, since G^ω is defined by the same attribute tuple as the finite state automaton $G^\omega = G = (Q, E, T, q_0, A)$, the difference between the two automata is that G^ω only accepts infinite words. G^ω accepts an infinite trajectory $\sigma = (e_1 e_2 \dots e_n \dots)$ if and only if σ starts from the initial state q_0 and visits a state $a \in A$ infinitely often. An ω-language is regular if and only if there exists a Büchi automaton that accepts it.

To the automaton G that models the system behavior is associated the corresponding Büchi automaton G^ω, and the ω-regular language accepted by G^ω is noted L_{sys}^{max}. Note that the language E^* only contains finite words and does not contain ω-words (E^* and E^ω are disjoint). The ω-language L_{sys}^{ω}, which contains all ω-words constructed as infinite sequences of words of L_{sys}, is in general a superset of $L_{sys}^{\max}$.

The $\mathcal{P}_{OBS}$ function is extended in order to apply to infinite words. In the hypothesis that the system contains no cycle of unobservable events, the projection over the set of observable events of an infinite trajectory is also infinite.

Definition 4.2 (EBS set of observables) *The set of observables OBS is defined as the projection on the observable events of the* ω*-language* L_{sys}^{max}*.*

$$OBS = \mathcal{P}_{OBS}\big(L_{sys}^{max}\big)$$

4.1.2 Fault signatures

For each fault mode $f \in \mathcal{F}_{sys}$, the *f-language*, denoted L_f, describes all possible trajectories in which the faults of f occur and no other fault. L_f is defined as the subset of the automaton language L_{sys}, restricted to the words containing at least one occurrence of every single fault composing f, and no occurrence of any other fault. L_f describes all possible scenarios leading to f. The words of the f-language are called *f-trajectories*.

Because of our particular interest for diagnosability, among the set of f-trajectories, we pay special attention to those that can be obtained when the observation temporal window can be arbitrarily extended, i.e. to observables. The concept of fault signatures hence relies on the extension of the f-language to infinite words. The maximal f-language L_f^{max} is defined as the subset of L_{sys}^{max} restricted to the ω-words that contain at least one occurrence of every fault composing f, and no occurrence of any other fault.

The fault signatures are obtained by projecting maximal f-languages on observable events. For each fault mode $f \in \mathcal{F}_{sys}$, the set of observable infinite trajectories obtained by projection of the ω-words of L_f^{max} over the set of observable events is called the *f-signature*. With the above definitions, it is possible to define the signature function *Sig* as the function associating its f-signature to any fault mode $f \in \mathcal{F}_{sys}$:

Definition 4.3 (EBS Fault signature) *The maximal fault language L_f^{max} is defined by:*

$$\begin{aligned} \forall f \in \mathcal{F}_{sys}, L_f^{max} &= \{\sigma \in L_{sys}^{max} | \forall F_i \in F_{sys}, F_i \in f \Leftrightarrow F_i \in \sigma\} \\ &= L_{sys}^{max} \cap \Big(L_f . \big(E \setminus (F_{sys} \setminus f) \big)^{\omega} \Big) \end{aligned}$$

The fault signature *is the projection of the maximal fault language on observable events:*

$$\forall f \in \mathcal{F}_{sys}\,, \quad Sig(f) = \mathcal{P}_{OBS}\big(L_f^{max}\big)$$

From definitions 4.2 and 4.3, it comes easily that the signature of a fault mode is a subset of OBS, and that every observable belongs to the signature of at least one fault mode.

We have defined the set of observables and the fault signature for event-based approaches, we now prove that the diagnosability definition as originally stated for EBS can be expressed in terms of fault signatures in the same way as SBS diagnosability definition.

4.2 Formal Comparison

4.2.1 Preliminary definitions

The definitions provided in chapter 2, adapted to the unified definitions and notations given in part 3 are now recalled. The SBS definition is just a reminder of the one given in chapter 3. The concepts of single fault or multiple fault, that in fact assess the presence or absence of all faults in the system, are now referred to as fault modes $f_i \in \mathcal{F}_{sys}$.

Definition 4.4 (SBS Diagnosability recalled) *A SBS is diagnosable if and only if:*

$$\big(\forall f_i, f_j \in \mathcal{F}_{sys}, f_i \neq f_j\big), Sig(f_i) \cap Sig(f_j) = \emptyset$$

In the original EBS definition 2.15, the fault events previously noted $e_{f_i} \in E_f$ are now referred to as faults $F_i \in F_{sys}$. Moreover, a significant difference is introduced in the definition: the original definition only applies to trajectories ending with a fault event ($s \in L_{\rightarrow e_{f_i}}$), the definition reformulated here applies to any trajectory containing the fault at any index ($F_i \in s$). The proof of equivalence between the two definitions 2.15 and 4.5 is given below.

Definition 4.5 (EBS Diagnosability reformulated) *An EBS is diagnosable if and only if:*

$$\forall F_i \in F_{sys}, \exists n_i \in \mathbb{N}, \forall s \in L_{sys} \,|\, (F_i \in s), \forall t \in E^* \,|\, (st \in L_{sys}),$$
$$\|t\| \geq n_i \Rightarrow \big(\forall u \in \mathcal{P}_{OBS}^{-1}\big(\mathcal{P}_{OBS}(st)\big), F_i \in u\big)$$

Proof Let $p(F_i, n_i, s, t)$ denote the logical sentence

$$\|t\| \geq n_i \Rightarrow \big(\forall u \in \mathcal{P}_{OBS}^{-1}\big(\mathcal{P}_{OBS}(st)\big), F_i \in u$$

The language $L_{\rightarrow F_i}$ is obviously a subset of the language $L_{F_i} = \{s \in L_{sys} | F_i \in s\}$. Consequently, if the proposition $\forall t \in E^* \,|\, (st \in L_{sys}), p(F_i, n_i, s, t)$ holds for all s in L_{F_i}, it trivially holds for any s in $L_{\rightarrow F_i}$. Then, definition 4.5 $\Rightarrow$ definition 2.15.

Let $F_i \in F_{sys}$, $n_i \in \mathbb{N}$ and $s \in L_{\rightarrow F_i}$ be such that $\forall t \in E^* \,|\, (st \in L_{sys}), p(F_i, n_i, s, t)$. Then, for any t_2 such that $stt_2 \in L_{sys}$, we also have $p(F_i, n_i, s, tt_2)$, since $\|tt_2\| \geq \|t\| \geq n_i$. Let now $t_1, t' \in E^*$ be such that $t_1 t' = tt_2$ and $\|t\| = \|t'\|$. We then have $p(F_i, n_i, s, t_1 t')$, and since $\|t\| = \|t'\| \geq n_i$, we have $p(F_i, n_i, st_1, t')$. Let $s' = st_1$, and let us consider which language can s' range over. s is a word that ends with F_i, and t_1 is a prefix of tt_2 which ranges over all the continuations of s in L_{sys}. Since L_{sys} is live, t_1 ranges over all the continuations of s in L_{sys}. Finally, we have that if $\forall s \in L_{\rightarrow F_i}, \forall t \in E^* | (st \in L_{sys}), p(F_i, n_i, s, t)$ then $\forall s' \in L_{sys} | (F_i \in s'), \forall t' \in E^* | (s't' \in L_{sys}), p(F_i, n_i, s', t')$, and definition 2.15 $\Rightarrow$ definition 4.5. ■

4.2.2 Equivalence

In this section, proof is given of the equivalence between the diagnosability definition in the EBS and SBS approaches. We first provide a third formulation of the EBS diagnosability (already given in definitions 2.15 and 4.5) by extending it to fault modes by definition 4.6, which provides a better insight into the definition interpretation. This result then allows us to complete the proof.

Our third formulation of the EBS diagnosability accounts for all the system fault modes: normal mode (which is actually trivial), single faults as well as multiple faults.

Definition 4.6 (EBS Diagnosability with fault modes) *An EBS is diagnosable if and only if:*

$$\forall f \in \mathcal{F}_{sys}, \exists n_f \in \mathbb{N}, \forall s \in L_f, \forall t \in E^* \mid (st \in L_{sys}),$$
$$\|t\| \geq n_f \Rightarrow \forall u \in \mathcal{P}_{OBS}^{-1}\big(\mathcal{P}_{OBS}(st)\big), \forall F_i \in f, F_i \in u$$

Theorem 4.1 *Given F_{sys} the set of faults and $\mathcal{F}_{sys}$ the set of fault modes of a system, the EBS diagnosability definition formulated for faults $F_i \in F_{sys}$ (definition 4.5) is equivalent to the EBS diagnosability definition formulated for fault modes $f \in \mathcal{F}_{sys}$ (definition 4.6).*

Proof Let us consider a fault mode f and a trajectory s in the f-language L_f. The diagnosability condition of definition 4.5 is verified for each $F_i \in f$ with possibly different n_i values. Taking the largest value of all these n_i values as n_f makes the condition of definition 4.6 true. Conversely, if definition 4.6 holds for all the fault modes containing a fault F_i with possibly different n_f values, then by taking the largest of all these n_f values as n_i, definition 4.5 holds. It follows that definition 4.5 is equivalent to definition 4.6, which accounts explicitly for fault modes $f = \{F_i\}, i = 1, \ldots, n$. ■

The above result shows that the EBS diagnosability definition can be given in terms of fault modes (instead of faults), like the SBS diagnosability definition. The EBS diagnosability definition is now proved to be equivalent to the SBS diagnosability definition.

Theorem 4.2 *The EBS diagnosability (definition 4.5) is equivalent to the unified diagnosability (definition 3.9).*

Proof It is first proved that if definition 4.6 is verified for one fault mode f_2, then for any fault mode f_1 such that $f_1 \setminus f_2 \neq \emptyset$, the statement $Sig(f_1) \cap Sig(f_2) = \emptyset$ holds.

Let f_1 and f_2 be two fault modes such that $f_1 \setminus f_2 \neq \emptyset$, and let the conditions expressed in definition 4.6 be verified. The set $A_1 = \{\mathcal{P}_{OBS}(st^\omega), s \in L_{f_1}, st^\omega \in L_{sys}^{max}\}$ is a superset of $Sig(f_1)$, since it includes ω-words not in $L_{f_1}^{max}$ (if another fault occurs in t). Yet, all the trajectories in $\mathcal{P}_{OBS}^{-1}(A_1)$ contain at least all faults in f_1. From definition 4.6, all the words of $\mathcal{P}_{OBS}^{-1}(Sig(f_2))$ contain at least

one occurrence of all the faults in f_2, and no occurrence of any other fault F_i (otherwise definition 4.6 would be violated for $f_2 \cup \{F_i\}$). Since $f_1 \setminus f_2 \neq \emptyset$ this implies that A_1 and $Sig(f_2)$ are disjoint, and consequently $Sig(f_1) \cap Sig(f_2) = \emptyset$.

The statement above is easily generalized to any pair of fault modes (f_i, f_j), since if $f_i \neq f_j$ then either $f_i \setminus f_j \neq \emptyset$ or $f_j \setminus f_i \neq \emptyset$. Hence definition 4.5 $\Leftrightarrow$ definition 4.6 $\Rightarrow$ definition 3.9.

Assume now that the condition expressed in definition 3.9 is true: for any fault mode $f_k \neq f_l$, all words in $Sig(f_k)$ are distinct from all words in $Sig(f_l)$.

Let f_i be a fault mode, consider all the fault modes f_k such that $f_i \subseteq f_k$ and the set $A_i = \{\mathcal{P}_{OBS}(st^\omega), s \in L_{f_i}, st^\omega \in L_{sys}^{max}\}$. Then from the definition of fault signature, we have $A_i \subseteq \bigcup_{f_k \supseteq f_i} Sig(f_k)$. Consequently, all words in $\mathcal{P}_{OBS}^{-1}(A_i)$ contain at least all faults in f_i (otherwise definition 3.9 would not hold).

This allows us to state that for all ω-words st^ω such that $s \in L_{f_i}, st^\omega \in L_{sys}^{max}$, we have $\mathcal{P}_{OBS}(st^\omega) \in A_i$, and $\mathcal{P}_{OBS}^{-1}\big(\mathcal{P}_{OBS}(st^\omega)\big) \subseteq \mathcal{P}_{OBS}^{-1}(A_i)$. Consequently, $\forall u \in \mathcal{P}_{OBS}^{-1}\big(\mathcal{P}_{OBS}(st^\omega)\big), \forall F_i \in f_i, F_i \in u$.

Expressing the property above in terms of finite words now completes the demonstration. The set $Pre(L_{f_i})$ contains the most prefixed words of L_{f_i}, i.e. $Pre(L_{f_i}) = \{s \in L_{f_i} \mid s' \text{ prefix of } s \Rightarrow s' \notin L_{f_i}\}$. For any $s \in Pre(L_{f_i})$, we denote $nmax_s$ the length of the longest word(s) in $\mathcal{P}_{OBS}^{-1}\big(\mathcal{P}_{OBS}(s)\big)$ and define $n_s = nmax_s - \|s\|$ (in the absence of cycles of unobservable events, the finite bound $nmax_s$ exists). We then define n_{f_i} as the greatest value for n_s when s ranges over $Pre(L_{f_i})$.

Let us reexpress the property: for all $s \in L_{f_i}$, for all t^ω such that $st^\omega \in L_{sys}^{max}$, let t be the prefix of t^ω of length n_{f_i}. For ω-words, we stated that $\forall u \in \mathcal{P}_{OBS}^{-1}\big(\mathcal{P}_{OBS}(st^\omega)\big), \forall F_i \in f_i, F_i \in u$, and since t is a prefix of t^ω, $\mathcal{P}_{OBS}^{-1}\big(\mathcal{P}_{OBS}(st)\big)$ contains only prefixes of words in $\mathcal{P}_{OBS}^{-1}\big(\mathcal{P}_{OBS}(st^\omega)\big)$. Hence, the (infinite) continuations of the words of $\mathcal{P}_{OBS}^{-1}\big(\mathcal{P}_{OBS}(st)\big)$ contain all the faults in f_i, and since $\|t\| = n_{f_i}$, words in $\mathcal{P}_{OBS}^{-1}\big(\mathcal{P}_{OBS}(st)\big)$ necessarily contain all the faults in f_i.

This means that for any trajectory st such that $s \in L_{f_i}, st \in L_{sys}$, $\|t\| \geq n_{f_i} \Rightarrow \forall u \in \mathcal{P}_{OBS}^{-1}\big(\mathcal{P}_{OBS}(st)\big), \forall F_i \in f_i, F_i \in u$, which is the condition of definition 4.6. Generalizing to any f_i, it implies definition 4.6 and definition 4.5. ■

4.3 Examples

This section presents two examples that illustrate the results reported in this chapter. The first example illustrates the fault signatures in an EBS and compares it to the classical diagnoser approach, and the second compares EBS and SBS approaches in an operational way. Bridges between state variables in the SBS view and events in the EBS view are provided and diagnosability analysis

is performed along the SBS and the EBS diagnosis approaches.

4.3.1 Fault signatures for EBS diagnosability

Let us consider again the example that was used in section 2.2 to illustrate the diagnoser approach for diagnosability, described by figure 2.7 page 49. It was explained why the first system is diagnosable while the second is not.

Diagnosability is now tested by building the fault signatures instead of the diagnoser. In both systems, there is only one fault, and consequently two fault modes: normal noted $\emptyset$, and faulty noted $\{f\}$. Figure 4.1 presents the details of the construction of fault signatures for both systems. In system n°1, the signatures are disjoint, however system n°2, the ω-word a^ω belongs to both signatures. System n°1 is hence diagnosable, and system n°2 is not.

The equivalence of these diagnosability results with the ones obtained by the diagnoser approach illustrates the equivalence of definitions 4.5 and 3.9: the signature approach to diagnosability shows the same results as the diagnoser approach.

	System n°1	System n°2
L_{sys}	a^*, a^*faab^*	a^*, a^*fa^*, a^*faab^*
L_{sys}^{max}	a^ω, a^*faab^ω	$a^\omega, a^*fa^\omega, a^*faab^\omega$
$L_{\emptyset}^{max}$	a^ω	a^ω
$L_{\{f\}}^{max}$	a^*faab^ω	$a^*fa^\omega, a^*faab^\omega$
$Sig(\emptyset)$	a^ω	a^ω
$Sig(\{f\})$	a^*b^ω	a^ω, a^*b^ω

Figure 4.1: Maximal languages and fault signatures for the two systems illustrated in figure 2.7 page 49.

4.3.2 Operational comparison of SBS and EBS

The system represented in figure 4.2 is inspired of [Puig *et al.*, 2005]. It is composed of two water tanks with heights y_1 and y_2, and a pump connected by a water flow channel. Both tanks supply consumers c_1 and c_2. The delays τ_1, respectively τ_2, correspond to the time needed for the water to reach tank2 from tank1, and tank1 from the pump. It has two operating modes: *pump on* and *pump off*. We consider faults in sensors y_1, y_2, c_1 and c_2, named respectively F_{y1}, F_{y2}, F_{c1} and F_{c2}.

The example is limited to single faults and it is assumed that the system does not switch its operating mode between the occurrence of a fault and the

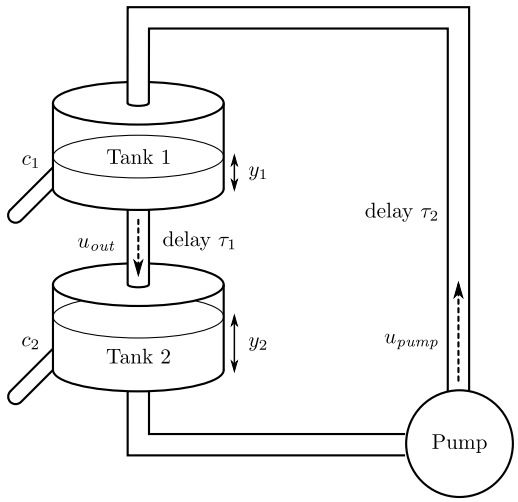


Figure 4.2: A water flow system

apparition of its symptoms, in order to simplify the models of the system.

Continuous model, state-based diagnosis

The discretized and linearized non-linear dynamic equations are:

$$\begin{aligned}
y_1(t+\Delta t) &= y_1(t) - k_1 c_1(t) + k_2 u_{pump}(t-\tau_2) - k_3 u_{out}(t) \\
u_{out}(t) &= k\sqrt{y_1(t)} \\
&\cong k_4 y_1(t) \\
u_{pump} &= k[a(h-y_2)^2 + b(h-y_2) + c] \\
&\cong k_5 + k_6 y_2(t) \\
y_2(t+\Delta t) &= y_2(t) - k_7 c_2(t) + k_8 u_{out}(t-\tau_1) - k_9 u_{pump}(t)
\end{aligned}$$

where Δt is the sampling time. u_{pump} being the flow through the pump, we can state that when the pump is off, we have $u_{pump}(t) = 0$.

From these equations, it is possible to predict the values for y_1 and y_2 with:

$$\begin{aligned}
\hat{y_1}(t+\Delta t) &= (1-k_3k_4)y_1(t) - k_1c_1(t) + k_2k_6y_2(t-\tau_2) + k_2k_5 \\
\hat{y_2}(t+\Delta t) &= (1-k_9k_6)y_2(t) - k_7c_2(t) + k_8k_4y_1(t-\tau_1) - k_9k_5
\end{aligned}$$

Pump on mode

$$\begin{aligned}
\hat{y_1}(t+\Delta t) &= (1-k_3k_4)y_1(t) - k_1c_1(t) \\
\hat{y_2}(t+\Delta t) &= y_2(t) - k_7c_2(t) + k_8k_4y_1(t-\tau_1)
\end{aligned}$$

Pump off mode

From the equations above, two consistency tests can be obtained in the form of analytical redundancy relations:

$$\begin{aligned} r_1(t+\Delta t) &= y_1(t+\Delta t) - \hat{y_1}(t+\Delta t) \\ &= y_1(t+\Delta t) - \big[(1-k_3k_4)y_1(t) - k_1c_1(t) + k_2k_6y_2(t-\tau_2) + k_2k_5\big] \\ r_2(t+\Delta t) &= y_2(t+\Delta t) - \hat{y_2}(t+\Delta t) \\ &= y_2(t+\Delta t) - \big[(1-k_9k_6)y_2(t) - k_7c_2(t) + k_8k_4y_1(t-\tau_1) - k_9k_5\big] \end{aligned}$$

Pump on mode

$$\begin{aligned} r_1(t+\Delta t) &= y_1(t+\Delta t) - \hat{y_1}(t+\Delta t) \\ &= y_1(t+\Delta t) - \big[(1-k_3k_4)y_1(t) - k_1c_1(t)\big] \\ r_2(t+\Delta t) &= y_2(t+\Delta t) - \hat{y_2}(t+\Delta t) \\ &= y_2(t+\Delta t) - \big[y_2(t) - k_7c_2(t) + k_8k_4y_1(t-\tau_1)\big] \end{aligned}$$

Pump off mode

Using these analytical redundancy relations, we deduce the fault signature matrices shown in figure 4.3.

The fault signature matrices indicate that the system is not diagnosable since, for example, the observable (*Pump on*, $r_1 = 1, r_2 = 1$) belongs to two fault signatures.

	F_{y1}	F_{y2}	F_{c1}	F_{c2}
r_1	1	1	1	0
r_2	1	1	0	1

Pump on mode

	F_{y1}	F_{y2}	F_{c1}	F_{c2}
r_1	1	0	1	0
r_2	1	1	0	1

Pump off mode

Figure 4.3: Fault signature matrices for the system

Discrete event model, dynamic diagnosis

For the EBS model of the system, the following events are used : p_{on},p_{off}, fired when the pump is turned on or off ; F_S fired when a fault occurs on sensor S ; r_1, r_2 fired when analytical redundancy relations r_1 and r_2, are violated.

The automaton is shown in Figure 4.4. An arc labelled $a.b$ represents two arcs labelled a and b, a leading to a state in which only b may occur.

From the automaton and following section 4.1, it is possible to build the signatures for all the faults (see Figure 4.5). Remember that all the events except faults are observable. The fault signatures are disjoint sets, the system is hence diagnosable.

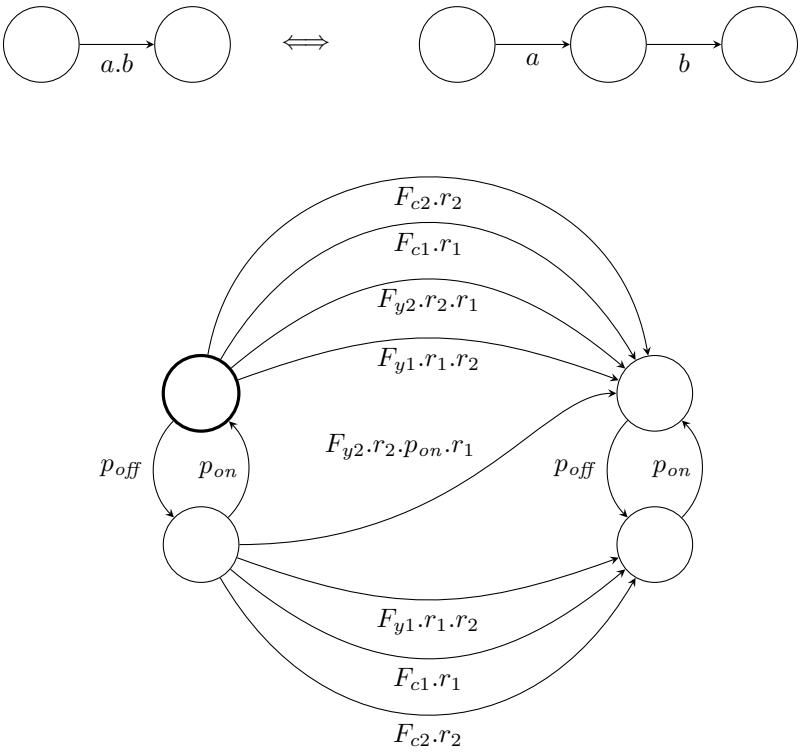


Figure 4.4: Automaton describing the system

4.3.3 Results

These examples show that, although EBS and SBS diagnosability definitions are formally equivalent, operational diagnosability assessment critically depends on the nature of observables.

In the SBS approach, diagnosability is not achieved, as fault signatures are not disjoint. ($Pump\ on, r_1 = 1, r_2 = 1$) is a signature for both F_{y1} and F_{y2}, and ($Pump\ off, r_1 = 0, r_2 = 1$) is a signature for both F_{y2} and F_{c2}.

In the EBS model, in the *pump on* mode, the symptoms $r_1 = 1$ and $r_2 = 1$ appear in the order ($r_1 r_2$) for F_{y1} and in reverse order ($r_2 r_1$) for F_{y2}. Taking this order into account permits fault discrimination between F_{y1} and F_{y2} in dynamic diagnosis. In addition, in the *pump off* mode, both F_{y2} and F_{c2} are followed by the r_2 symptom, but only in the case of F_{y2}, a p_{on} command will be followed by the r_1 symptom. Notice that diagnosability stands on the assumption that the pump will be turned on some time: it is only after the p_{on} command that the faults can be discriminated.

Fault	Signature
$\emptyset$	$(p_{on}p_{off})^{\omega}$
F_{c1}	$(p_{on}p_{off})^{*}\mathbf{r_1}(p_{on}p_{off})^{\omega}$ $(p_{on}p_{off})^{*}\mathbf{p_{on}r_1}(p_{off}p_{on})^{\omega}$
F_{c2}	$(p_{on}p_{off})^{*}\mathbf{r_2}(p_{on}p_{off})^{\omega}$ $(p_{on}p_{off})^{*}\mathbf{p_{on}r_2}(p_{off}p_{on})^{\omega}$
F_{y1}	$(p_{on}p_{off})^{*}\mathbf{r_1r_2}(p_{on}p_{off})^{\omega}$ $(p_{on}p_{off})^{*}\mathbf{p_{on}r_1r_2}(p_{off}p_{on})^{\omega}$
F_{y2}	$(p_{on}p_{off})^{*}\mathbf{r_2p_{on}r_1}(p_{off}p_{on})^{\omega}$ $(p_{on}p_{off})^{*}\mathbf{p_{on}r_2r_1}(p_{off}p_{on})^{\omega}$

Figure 4.5: Fault signatures (discriminant sub words are bolder).

4.4 Conclusion of the comparison

It has been shown that the EBS diagnosability definition could be stated in terms of fault signatures. This result proves that the definitions given in chapter 3 unify both state-based and event-based approaches to diagnosability. A formal comparison shows that EBS diagnosability analysis can be done by constructing the fault signatures or equivalently by using the diagnoser approach, and the same results are obtained. An operational comparison shows that EBS and SBS approaches do not generally give the same diagnosability results, this is due to the representation formalism, and illustrates the considerations provided in section 1.4. The choice of the knowledge to be represented, and the formalism used to represent this knowledge influence the final diagnosability results.

The formal comparison between SBS and EBS approaches related in section 4.2 is based on the adaptation of the EBS definition from faults to fault modes. We could have equivalently adapted the SBS definition to faults. This idea has led to the considerations provided in the next chapter, signatures for partial fault modes.

Chapter 5

Signatures for partial fault modes

This chapter describes how the unification work described in the previous chapters leads to a new characterization of diagnosability that allows one to perform diagnosability analysis efficiently.

The concept of signature has been applied, until now, only to fault modes. In this chapter, we define *partial fault modes*, and adapt the definition of signature to them. We show that computing the signatures for partial fault modes allows us to decrease the number of signature comparisons for diagnosability analysis.

This chapter is based on the *variable* representation of fault modes (see definition 3.5 page 62), initially defined in state-based approaches. In event-based approaches, it is easier to model fault modes as sets (see definition 3.4), but the translation from the set representation to the variable representation is straightforward when using binary mode variables, as described in section 3.1.3.

The concepts of partial fault mode and their signature are model independent. They are applied to a distributed context, with distributed constraint-based models. An algorithm, for computing the signatures for partial fault modes is presented, based on a diagnosis algorithm defined in [Pucel *et al.*, 2007, Ardissono *et al.*, 2005, Console *et al.*, 2007]. Diagnosability analysis based on partial fault mode signatures is then performed in this context.

5.1 Partial fault modes

A *fault mode* is related to the whole system: it describes the behavioral mode of all the components in the system. In the variable representation (see definition 3.5), it is represented by an assignment to *all* mode variables. Diagnosis consists

in deciding in which fault mode the system is, by assessing which faults have occurred and which have not.

Our approach is based on the analysis of *partial fault modes*, that are related to only *some* of the system components.

The set of variables of the system is noted V. $V_{mode} \subset V$ is the set of mode variables, these variables are generally noted $m_i, i \in \{1 \dots n\}$. V_{OBS} is the set of observable variables, noted $o_j, j \in \{1 \dots m\}$.

Definition 5.1 (Partial fault mode) *A* partial fault mode *is defined by assigning a value to* some *of the system mode variables. A partial fault mode in which* all *mode variables are assigned is a* fault mode*.*

The scope *of a partial fault mode pfm is the set containing the mode variables assigned by pfm. It is noted* Sco*(pfm).*

The rank *of a partial fault mode is the cardinality of its scope.*

Two partial fault modes pfm_1 and pfm_2 are alternative *if and only if they have the same scope, but are not equal.*

A partial fault mode pfm_1 refines *another partial fault mode pfm_2 if and only if* $\text{Sco}(pfm_2) \subset \text{Sco}(pfm_1)$ *and* $pfm_1 \Rightarrow pfm_2$*.*

For example, $m_1 = ok \wedge m_2 = ok$ and $m_1 = ok \wedge m_2 = ab$ are alternative partial fault modes, while $m_1 = ab \wedge m_2 = ok$ refines $m_1 = ab$. Fault modes are partial fault modes with the greatest rank, i.e. the number of mode variables. Fault modes are all alternatives with one another, and cannot be refined.

Reasoning on partial fault modes allows one to reason on some instantiated parts of the system while keeping the rest unconstrained. Actually, two alternative partial fault modes describe two different fault situations in a subsystem, and two sets of fault situations in the system. Hence, defining signatures for partial fault modes allows to one perform diagnosability analysis starting from subparts of the system.

Definition 5.2 (Partial fault mode Signature) *The* signature *of a partial fault mode pfm is the union of the signatures of all the fault modes that refine pfm.*

$$Sig(pfm) = \bigcup_{f \in \mathcal{F}_{sys} \,|\, f \text{ refines } pfm} Sig(f)$$

The point behind this definition is that in some types of formalisms for expressing the system model, it is possible or even simpler to compute the signature of a partial fault mode than the signature of a fault mode. In this case, it is possible to perform diagnosability analysis directly on partial fault modes, which, as we will see in the next section, can be more efficient.

Definition 5.3 (Partial fault mode Discriminability) *Two partial fault modes are* discriminable *if and only if their signatures are disjoint.*

In the special case where the considered partial fault modes are fault modes, this definition falls back to definition 3.8. The following property is a straightforward consequence of the previous definitions:

Theorem 5.1 *Two partial fault modes pfm_1 and pfm_2 are discriminable if and only if every refinement of pfm_1 is discriminable from every refinement of pfm_2.*

Diagnosability definitions based on discriminability generally apply to fault modes, however property 5.1 allows us to state the notion of diagnosability in a more generic way, fault modes being particular cases of partial fault modes:

Definition 5.4 (Partial fault mode Diagnosability) *A partial fault mode is* diagnosable *if and only if it is discriminable from all its alternatives.*

A system is diagnosable if and only if all its partial fault modes are diagnosable.

Here again, if restricted to fault modes, this definition falls back to definition 3.9. In general, it is sufficient that all the partial fault modes at a given rank n are diagnosable for the system to be diagnosable. Proving this statement is pretty straightforward with theorem 5.1.

5.2 Diagnosability analysis with partial fault modes

The definitions and properties of partial fault modes allow one to decrease the number of signatures to be compared. When signatures for partial fault modes can be computed efficiently, this approach provides improved performance for diagnosability analysis.

Diagnosability analysis relies on checking the discriminability of pairs of alternative partial fault modes. In the previous approaches, only pairs of total fault modes are checked. In this approach, we first check the discriminability of alternative pairs of partial fault modes of rank 1. Such comparison provides information about a whole set of pairs of fault modes. Then if comparing pairs of partial fault modes of rank 1 is not sufficient, pairs of rank 2 are compared, and so on.

Figure 5.1 illustrates the comparisons that are performed for 3 variables and arbitrary discriminability results. The analysis is done in 3 steps:

1. First, rank 1 pairs of partial fault modes (left column) are checked. The pair $\binom{m_1=ok}{m_1=ab}$ is discriminable, hence we know that the 8 pairs of rank 2

partial fault modes, and the 16 pairs of total fault modes that refine this pair are also discriminable.

2. Then we check the 10 rank 2 pairs of partial fault modes (central column) that were not discarded at rank 1 analysis. At rank 2, 3 discriminable pairs are found, thus discarding 8 other pairs of total fault modes.

3. Finally, we check the remaining 4 pairs of rank 3 partial fault modes, which are in fact pairs of fault modes. At rank 3, no new discriminable pair is found.

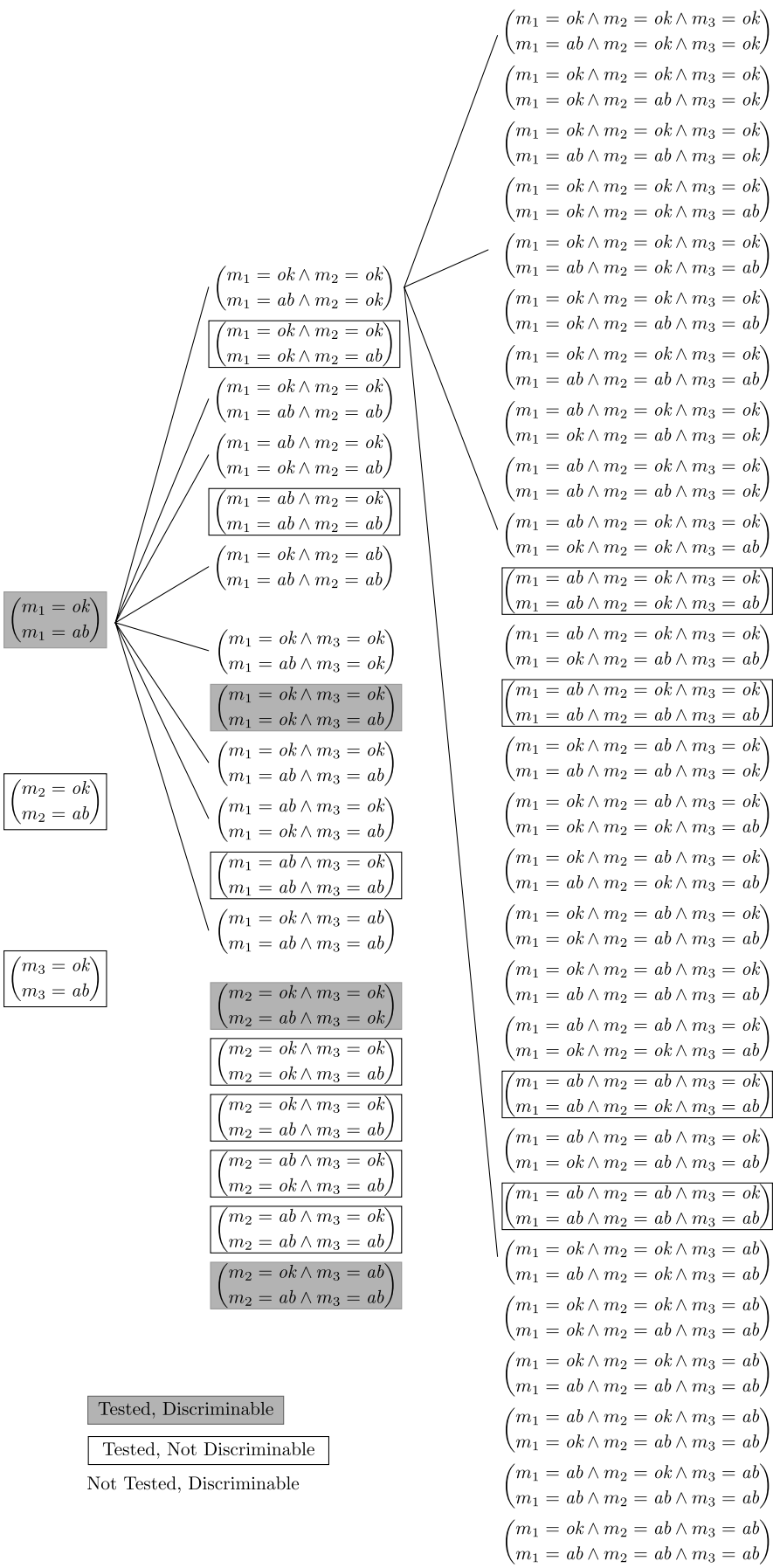


Figure 5.1: Partial fault mode signature comparisons. For 3 binary mode variables, there are 28 pairs of fault modes to compare. In this example, the results are obtained with only 17 comparisons. The algorithmic benefit grows with the number of mode variables.

5.3 Distributed diagnosability analysis

The overall approach described in the previous section is now applied to a distributed diagnosability analysis approach. As already mentioned, the model used in this approach is a state-based model that uses constraint networks. It was originally described in [Ardissono *et al.*, 2005, Console *et al.*, 2007].

5.3.1 Constraint networks

The formalism of constraint networks has similarities with the first-order logic described in section 2.1.2. We first provide definitions for some concepts that are used throughout this section. In this formalism, we consider a set V of variables that range over discrete, finite sets of values. The set of values that a variable $v_i \in V$ can take is its domain, noted $\mathrm{Dom}(v_i)$.

In the following, we assume there is a total order on the set of variables V, and we speak indifferently of sets and tuples of variables. The domain of a set/tuple of variables of V is the Cartesian product:

$$\mathrm{Dom}\big((v_1, v_2, \ldots, v_n)\big) = \mathrm{Dom}(v_1) \times \mathrm{Dom}(v_2) \times \ldots \times \mathrm{Dom}(v_n)$$

Definition 5.5 (Assignment, constraint, satisfaction) *An* assignment γ *is defined by a pair* (S_γ, val_γ) *where* $S_\gamma \subseteq V$ *is the assignment's* scope*, and* $val_\gamma \in \mathrm{Dom}(S_\gamma)$ *is a tuple of values for the variable tuple* S_γ*.*

In the following, the scope S_γ *is noted* $\mathrm{Sco}(\gamma)$*, and the value of a variable* $v \in \mathrm{Sco}(\gamma)$ *is noted* $\gamma(v)$*.* γ *is:*

- *A* total assignment *if and only if* $\mathrm{Sco}(\gamma) = V$*.*
- *The unique* empty assignment *if and only if* $\mathrm{Sco}(\gamma) = \emptyset$*.*
- *A* partial assignment *if and only if* $\emptyset \subset \mathrm{Sco}(\gamma) \subset V$*.*

A constraint C *is defined by a pair* (S_C, R_C) *where* $S_C \subseteq V$ *is the constraint's scope, and* $R_C \subseteq \mathrm{Dom}(S_C)$ *denotes all the value tuples allowed for* S_C *by the constraint. The notation* $\mathrm{Sco}(C) = S_C$ *denotes the constraint scope.*

An assignment γ satisfies *a constraint* C *if and only if they have the same scope, and the value tuple defined by* γ *belongs to the constraint relation* R_C*. An assignment* γ *satisfies a set of constraints if and only if it satisfies every constraint in this set.*

Two constraints C *and* C' *are* equivalent*, noted* $C \equiv C'$ *if and only for any assignment* γ*,* (γ *satisfies* C) $\Leftrightarrow$ (γ *satisfies* C')*.*

An assignment can be seen as a constraint that is only satisfied by itself. Conversely, a constraint can be defined by the set of assignments that satisfy it.

Definition 5.6 (Restriction, extension) *The* restriction *or* projection *of an assignment γ on a non empty set of variables $S \subset \mathrm{Sco}(\gamma)$ is the unique assignment γ' such that $\mathrm{Sco}(\gamma') = S$ and $\forall v \in S, \gamma'(v) = \gamma(v)$. For any set $T \subseteq V$, the notation $\gamma|_T$ denotes the restriction of γ on $T \cap \mathrm{Sco}(\gamma)$.*

γ is an extension *of γ' if and only if γ' is the projection of γ on $\mathrm{Sco}(\gamma')$.*

The restriction *of a constraint C on a set of variables $S \subset \mathrm{Sco}(C)$, noted $C|_S$ is the unique constraint such that $\mathrm{Sco}(C') = S$, and for any assignment γ, (γ satisfies C) $\Rightarrow$ ($\gamma|_S$ satisfies $C|_S$) and (γ satisfies $C|_S$) $\Rightarrow$ (at least one extension of γ to $\mathrm{Sco}(C)$ satisfies C).*

C is an extension *of C' if and only if C' is the projection of C on $\mathrm{Sco}(C')$.*

We finally introduce the concept of consistency, which is fundamental for constraint-based reasoning. It captures the idea of compatibility between assignments and constraints. The operation of combination is also introduced, and is indifferently denoted by the operator "$\wedge$" or the operator "$\cup$". In this document we prefer the notation "$\wedge$".

Definition 5.7 (Consistency, combination) *Two constraints C and C' are* consistent *if and only if either $\mathrm{Sco}(C) \cap \mathrm{Sco}(C') = \emptyset$, or there exists an assignment that satisfies both $C|_{\mathrm{Sco}(C)\cap\mathrm{Sco}(C')}$ and $C'|_{\mathrm{Sco}(C)\cap\mathrm{Sco}(C')}$.*

Two assignments γ and γ' are consistent *if and only if either $\mathrm{Sco}(\gamma) \cap \mathrm{Sco}(\gamma') = \emptyset$, or $\gamma|_{\left(\mathrm{Sco}(\gamma)\cap\mathrm{Sco}(\gamma')\right)} = \gamma'|_{\left(\mathrm{Sco}(\gamma)\cap\mathrm{Sco}(\gamma')\right)}$.*

An assignment γ is consistent *with a constraint C if and only if either $\mathrm{Sco}(\gamma) \cap \mathrm{Sco}(C) = \emptyset$ or $\gamma|_{\left(\mathrm{Sco}(\gamma)\cap\mathrm{Sco}(C)\right)}$ satisfies $C|_{\left(\mathrm{Sco}(\gamma)\cap\mathrm{Sco}(C)\right)}$.*

The combination *of two consistent constraints C and C' is the unique constraint $C'' = C \wedge C'$ such that $\mathrm{Sco}(C'') = \mathrm{Sco}(C) \cup \mathrm{Sco}(C')$ and such that for any assignment γ, (γ satisfies C'') $\Leftrightarrow$ ($\gamma|_{\mathrm{Sco}(C)}$ satisfies C and $\gamma|_{\mathrm{Sco}(C')}$ satisfies C').*

The combination *of two consistent assignments γ and γ' is the unique assignment $\gamma'' = \gamma \wedge \gamma'$ such that $\mathrm{Sco}(\gamma'') = \mathrm{Sco}(\gamma) \cup \mathrm{Sco}(\gamma')$ and such that $\forall v \in \mathrm{Sco}(\gamma), \gamma''(v) = \gamma(v)$ and $\forall v \in \mathrm{Sco}(\gamma'), \gamma''(v) = \gamma'(v)$.*

The combination *of an assignment γ and a constraint C is the constraint $C' = C \wedge \gamma$ such that $\mathrm{Sco}(C') = \mathrm{Sco}(C)$ and for any assignment γ', (γ' satisfies C') $\Leftrightarrow$ (γ' satisfies C and γ' consistent with γ).*

Consistency captures a similar concept as satisfaction, but is slightly more general in the sense that it applies to assignments and constraints that do not have the same scope.

As we will see in next section, a system can be modeled by a constraint or

a set of constraints, and defining mode and observable variables allows us to perform diagnosis and diagnosability analysis.

5.3.2 Diagnosis approach

The diagnosability analysis approach that we present in this section relies on a diagnosis approach described in [Ardissono *et al.*, 2005, Console *et al.*, 2007] that makes use of a constraint-based model. In this approach, not only the model is reused for diagnosability analysis, but also an important part of the diagnosis algorithm. We first describe the diagnosis context and algorithm.

Constraint-based reasoning for distributed diagnosis

Constraint-based models can be used to represent a state-based system. In this case, the model M consists in a set of constraints over the characterizing variables. The constraints describe the system behavior, by accepting exactly all the situations that may be reached during operation. Introducing *mode* variables allows one to qualify the system behaviors as normal or faulty, hence permitting to perform diagnosis.

The overall principle of diagnosis reasoning with constraint-based models is to receive as input an assignment *obs* on some observed variables, and determine which assignments to mode variables are consistent with the model and the observations. The diagnosis problem can be seen as a specific constraint satisfaction problem, that consists in listing all the assignments to mode variables that are consistent with $M \wedge obs$. Very efficient algorithms exist in the centralized case [Hamscher *et al.*, 1992, Darwiche, 1999, Dechter, 2003].

Definition 5.8 (Constraint-based diagnosis) *Let M be a model and obs be an observation. We denote by V_{mode} the set of mode variables. An assignment γ is a* diagnosis candidate *for M and obs if and only if $\mathrm{Sco}(\gamma) = V_{mode}$, and γ is consistent with $M \wedge obs$.*

An assignment γ is a partial diagnosis *for M and obs if and only if $\mathrm{Sco}(\gamma) \subset V_{mode}$ and every extension of γ to V_{mode} is a diagnosis candidate.*

In some contexts (geographic distribution, collaboration between companies, etc), diagnosis cannot be performed by a single entity. Diagnosis is then performed by several entities by means of *local diagnosers*, which may communicate with a common *supervisor* (decentralized approach), or directly with one another (distributed approach).

The overall architecture, initially described in [Ardissono *et al.*, 2005, Console *et al.*, 2007] is based on several *local diagnosers* $LD_1, \ldots, LD_n$ which cooperate with a *supervisor* D. Each local diagnoser LD_i is responsible for a component C_i (or a set of components), while D puts together information from

local diagnosers and selects which local diagnosers to question further in order to diagnose problems.

Each local diagnoser possesses a *model* of its components; the approach makes the following assumptions about models:

- Each model is given as a set of constraints over finite-domain variables.
- For some components there is a distinguished *mode* variable that expresses which behaviour mode the component is in. We consider, for the sake of simplicity, that there is only one normal mode named *ok* and only one fault mode named *ab*.
- Interactions with other components are represented by means of "shared" variables (where "shared" means that each model has its own variable, and an implicit equality constraint exists between the two). These variables are called *interface* variables.

5.3.3 Constraint propagation control

In such contexts, the most limited resource for diagnosis computation is generally the communication rate. It is often preferred to decrease the amount of communications, even at the cost of a reasonable computation overhead. In particular, it is desirable to limit the assignments to the variables that are really relevant in the diagnosis process. For example, if an observation received by local diagnoser LD_1 results from a fault in the subsystem managed by local diagnoser LD_2, then other local diagnosers should not be involved in the diagnosis process, except if their subsystems participate in the communication between the subsystems managed by LD_1 and LD_2.

The task of the local diagnosers and the supervisor is based on *partial assignments* to model variables (in particular, to mode variables and to interface variables). A partial assignment corresponds to some hypothesis of behaviour on a part of the system. The operation performed by a local diagnoser in order to explain an abnormal behaviour is called the *Extend* operation, each output partial assignment being an extension of an input partial assignment, and is used to propagate hypotheses across the system, either to find local causes of an abnormal situation or to explore the consequences of an hypothesis.

Reasoning on partial assignments is synergistic with the notion of admissibility. Some partial assignments are admissible with respect to the system model, and the diagnosis reasoning is done exclusively on such assignments. We present a definition, and explain how this property facilitates diagnosis reasoning.

Definition 5.9 (Admissibility) *Let M be a model and let γ be a partial assignment. We say that γ is* admissible *with respect to M if and only if:*

1. *γ is consistent with M;*

2. *the restriction of* $M \wedge \gamma$ *to variables* not *assigned by* γ *is equivalent to the restriction of* M *itself to the same variables:*

$$(M \wedge \gamma)|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\gamma)} \equiv M|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\gamma)}$$

Let us now illustrate and explain the concept of admissibility. When one considers a partial assignment, one faces the question of what are the total assignments that are consistent with this assignment and the model. In the general case, such an operation is done by appending to an assignment γ the various tuples contained by $(M \wedge \gamma)|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\gamma)}$. In the case of an admissible assignment, this operation is simply done by appending the tuples of $M|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\gamma)}$, as illustrated in figure 5.2.

For diagnosis, the concept of admissibility is very important: let *obs* be a partial assignment to some observed variables, and let γ be an extension of *obs* that assigns *some* mode variables, but not all. If γ is admissible, this means that the tuple of unassigned mode variables can range over its original domain, i.e. that *the restriction of* γ *to* $\mathrm{Sco}(\gamma) \cap V_{mode}$ *is a partial diagnosis*, because the model always allows all the values in their domain for mode variables.

Reasoning on admissible partial assignments allows one to perform diagnosis without involving unneeded local diagnosers and analyzing unneeded parts of the system. Ideally, keeping the partial assignments as general as possible, by only manipulating "minimal" admissible extensions of the observations would be the best. However, as discussed in [Console *et al.*, 2007], in general this cannot be done without local diagnosers sharing more information with each other about their local models (something that is not desirable in a distributed environment). Then the weaker notion of *complete set of admissible extensions* is used.

Definition 5.10 (Complete set of admissible extensions) *Let* M *be a model and let* γ *be a partial assignment. A set* E *of admissible assignments extending* γ *is* complete *if every total assignment consistent with* $M \wedge \gamma$ *is an extension of some* $\delta \in E$*.*

In the diagnosis approach, for each assignment α in input, a local diagnoser computes a complete set of extensions (with respect to the local model M_i) for $\alpha \wedge \omega$ where ω are local observations that are performed by the local diagnoser and can, of course, discard some hypotheses.

The overall diagnostic process starts from an abnormal observation *obs* in component i, and its local diagnoser is asked to compute a complete set of admissible extensions of *obs*.

If an extension computed by the local diagnoser LD_i assigns a value to an interface variable shared by service local diagnoser LD_j, then LD_j is invoked by the supervisor to further extend the assignment, taking into account the local model, and the observations in the components monitored by LD_j, which may discard some hypotheses. At the end of the supervisor loop, *a complete*

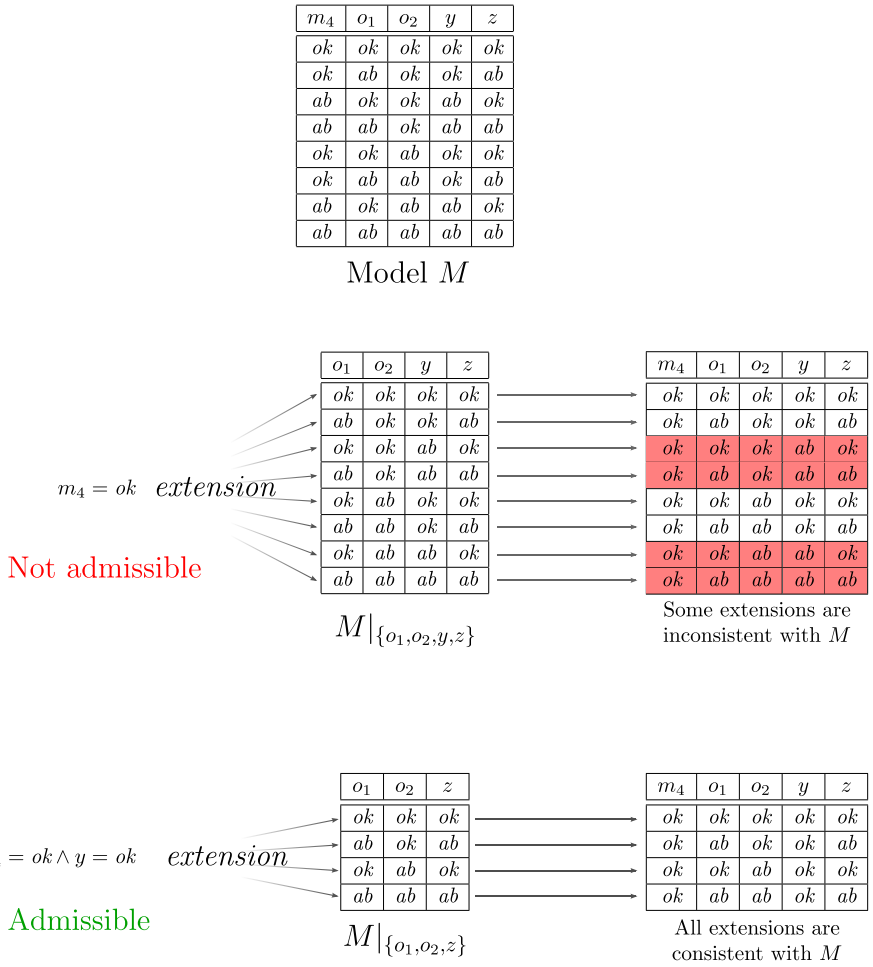


Figure 5.2: Admissible and non admissible partial assignments. $m_4 = ok$ is not admissible with respect to the model M. On the contrary, $m_4 = ok \wedge y = ok$ is admissible : the tuple of unassigned variables can range over the values allowed by M. In other words, $M \wedge (m_4 = ok \wedge y = ok)$ does not constrain variables (o_1, o_2, z) more than M alone.

set of admissible extensions for the observations obtained in the whole system is computed. The extensions in this set can be restricted to mode variables in order to obtain a complete list of partial diagnoses.

In the approach described in the following sections, the same decentralized algorithm is used to predict observable consequences of some faults, *independently* of the presence of other faults. This is the base of diagnosability analysis.

5.3.4 Diagnosability analysis

The diagnosis algorithm follows this general scheme:

1. Receive an observation as input.
2. *Extend*: compute a complete set of admissible extensions for the input.
3. Restrict these extensions to mode variables.
4. Return this restriction as a complete set of partial diagnoses.

Intuitively, the complete set of admissible extensions contains all the possible causes for an abnormal observation. However, the model does not contain any information about causality. The extension operation explores the causes or consequences of the input hypothesis without distinction. It can hence be used to compute the consequences of the presence and absence of some faults in the system, and generate the fault signatures as consistent assignment on observable variables. The general scheme of signature generation is the following:

1. Receive a partial fault mode as input.
2. *Extend*: compute a complete set of admissible extensions for the input.
3. Restrict these extensions to observable variables.
4. Return this restriction as a complete "partial signature".

These so called "partial signatures" are an important aspect of this diagnosability analysis approach. This section presents several theorems that prove that these "partial signatures" can be used to check if the signatures of alternative partial fault modes intersect, and deduce their discriminability.

The first theorem shows that the *Extend* operation only needs to be performed for rank 1 partial fault modes. Complete sets of admissible extensions for higher rank partial fault modes can be obtained by combining those for rank 1.

Theorem 5.2 *Let pfm_1 and pfm_2 be two consistent partial fault modes, and let us assume to have a complete set* $\mathrm{Ext}(pfm_i)$ *of admissible extensions for each of them. Then the set:*

$$\{\alpha_1 \wedge \alpha_2 \mid \alpha_1 \in \mathrm{Ext}(pfm_1), \alpha_2 \in \mathrm{Ext}(pfm_2), \alpha_1 \textit{ consistent with } \alpha_2\}$$

is a complete set of admissible extensions for $pfm_1 \wedge pfm_2$.

Proof Let α_1 be any element of $\mathrm{Ext}(pfm_1)$ and α_2 any element of $\mathrm{Ext}(pfm_2)$ such that α_1 and α_2 are consistent. It is pretty straightforward that $\alpha_1 \wedge \alpha_2$ is a partial assignment and is an extension of $pfm_1 \wedge pfm_2$.

Now let us prove that $\alpha_1 \wedge \alpha_2$ is admissible. First, $\alpha_1 \wedge \alpha_2$ is trivially consistent with M. From definition 5.6 it is easily proved that for any assignment γ, and for any sets $S'' \subset S' \subset \mathrm{Sco}(\gamma)$, we have $(\gamma|_{S'})|_{S''} = \gamma|_{S''}$. Consequently, we have:

$$\begin{aligned}
M|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\alpha_1\wedge\alpha_2)} &\equiv \Big(M|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\alpha_1)}\Big)|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\alpha_1\wedge\alpha_2)} \\
&\equiv \Big(\big(M\wedge\alpha_1\big)|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\alpha_1)}\Big)|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\alpha_1\wedge\alpha_2)} \\
&\equiv \big(M\wedge\alpha_1\big)|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\alpha_1\wedge\alpha_2))} \\
&\equiv \Big(\big(M\wedge\alpha_1\big)|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\alpha_2)}\Big)|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\alpha_1\wedge\alpha_2)} \\
&\equiv \Big(\big(M\wedge\alpha_1\wedge\alpha_2\big)|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\alpha_2)}\Big)|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\alpha_1\wedge\alpha_2)} \\
&\equiv \big(M\wedge\alpha_1\wedge\alpha_2\big)|_{\mathrm{Sco}(M)\setminus\mathrm{Sco}(\alpha_1\wedge\alpha_2)}
\end{aligned}$$

Hence $\alpha_1 \wedge \alpha_2$ is admissible.

Finally, let us prove that the set of admissible extensions defined in theorem 5.2 is complete. Let *ext* be any total extension of $pfm_1 \wedge pfm_2$ consistent with M. *ext* is trivially an extension of pfm_1 (resp. pfm_2). Since $\mathrm{Ext}(pfm_1)$ (resp. $\mathrm{Ext}(pfm_2)$) is complete, then *ext* is an extension of some assignment $\gamma_1 \in \mathrm{Ext}(pfm_1)$ (resp. $\gamma_2 \in \mathrm{Ext}(pfm_2)$). Consequently, *ext* is an extension of $\gamma_1 \wedge \gamma_2$ which means that the set defined in theorem 5.2 is complete. ■

The next property is the one on which the whole approach rests: it states that the discriminability of two alternative partial fault modes can be assessed by comparing their respective complete sets of admissible extensions.

Theorem 5.3 *Let pfm_1 and pfm_2 be two alternative partial fault modes, and let us assume to have a complete set* $\mathrm{Ext}(pfm_i)$ *of admissible extensions for each of them. Then pfm_1 and pfm_2 are discriminable if and only if for any $(\alpha_1, \alpha_2) \in \big(\mathrm{Ext}(pfm_1) \times \mathrm{Ext}(pfm_2)\big)$, $\alpha_1|_{V_{OBS}}$ is not consistent with $\alpha_2|_{V_{OBS}}$.*

Proof For any partial fault mode *pfm*, every observable $obs \in Sig(pfm)$ is the projection on V_{OBS} of a total assignment γ such that γ is consistent with $M \wedge pfm$. If $\mathrm{Ext}(pfm)$ is a complete set of admissible extensions for *pfm*, then there exists a $\delta \in \mathrm{Ext}(pfm)$ such that γ is an extension of δ. Consequently, $\gamma|_{V_{OBS}} = obs$ is an extension of $\delta|_{V_{OBS}}$.

Let us suppose that pfm_1 and pfm_2 are not discriminable, and let $obs \in Sig(pfm_1) \cap Sig(pfm_2)$. Then, the previous paragraph result implies that there exists an $\alpha_1 \in \text{Ext}(pfm_1)$ and an $\alpha_2 \in \text{Ext}(pfm_2)$ such that obs is an extension of both $\alpha_1|_{V_{OBS}}$ and $\alpha_2|_{V_{OBS}}$. Consequently, $\alpha_1|_{V_{OBS}}$ and $\alpha_2|_{V_{OBS}}$ are consistent.

If $\alpha_1|_{V_{OBS}}$ and $\alpha_2|_{V_{OBS}}$ are consistent, then there exists an assignment obs that extends both $\alpha_1|_{V_{OBS}}$ and $\alpha_2|_{V_{OBS}}$ to V_{OBS} and is consistent with M. Then, there exists a total assignment γ that extends obs and is consistent with $M \wedge \alpha_1$ (resp. $M \wedge \alpha_2$). γ is then necessarily consistent with $M \wedge pfm_1$ (resp. $M \wedge pfm_2$). Consequently, $obs = \gamma|_{V_{OBS}}$ belongs to $Sig(pfm_1)$ (resp. $Sig(pfm_2)$). Since obs belongs to the signatures of pfm_1 and pfm_2, these two partial fault modes are not discriminable. ■

The next property is the most complex. It enables us to have an early detection of hopelessly non-discriminable pairs, so to avoid refining them at higher ranks.

Theorem 5.4 *Let pfm_1 and pfm_2 be two alternative undiscriminable partial fault modes. Let D be their common scope, and $\text{Ext}(pfm_1)$ and $\text{Ext}(pfm_2)$ be two respective complete sets of admissible extensions.*

Let $\text{Sco}(\text{Ext}(pfm_i)) = \bigcup_{\alpha_i \in \text{Ext}(pfm_i)} \text{Sco}(\alpha_i)$ for $i \in \{1, 2\}$. We consider a mode variable $m \notin \big(\text{Sco}(\text{Ext}(pfm_1)) \cup \text{Sco}(\text{Ext}(pfm_2))\big)$ and two (possibly equal) partial fault modes pfm'_1 and pfm'_2 such that $\text{Sco}(pfm'_1) = \text{Sco}(pfm'_2) = \{m\}$.

We define $pfm''_1 = pfm_1 \wedge pfm'_1$ and $pfm''_2 = pfm_2 \wedge pfm'_2$, and have that pfm''_1 is discriminable from pfm''_2 if and only if pfm'_1 is discriminable from pfm'_2.

Let us illustrate this theorem by an example. Let m_1 and m_2 be two mode variables, and let $m_1 = ok$ and $m_1 = ab$ be undiscriminable. Let us finally assume that the complete sets of admissible extensions $\text{Ext}(m_1 = ok)$ and $\text{Ext}(m_1 = ab)$ never assign the variable m_2.

We already know by theorem 5.1 that if $m_2 = ok$ is discriminable from $m_2 = ab$ then $(m_1 = ok \wedge m_2 = ok)$ is discriminable from $(m_1 = ab \wedge m_2 = ab)$. Theorem 5.4 tell us that if $m_2 = ok$ is not discriminable from $m_2 = ab$, the hypotheses allow us to deduce that $(m_1 = ok \wedge m_2 = ok)$ is undiscriminable from $(m_1 = ab \wedge m_2 = ab)$. The following pairs are built in a similar way and are also undiscriminable:

- $(m_1 = ok \wedge m_2 = ok)$ and $(m_1 = ab \wedge m_2 = ok)$
- $(m_1 = ok \wedge m_2 = ab)$ and $(m_1 = ab \wedge m_2 = ok)$
- $(m_1 = ok \wedge m_2 = ab)$ and $(m_1 = ab \wedge m_2 = ab)$

Proof It is supposed that pfm_1 and pfm_2 are alternative and undiscriminable. Then there exists $\alpha_1 \in \text{Ext}(pfm_1)$ and $\alpha_2 \in \text{Ext}(pfm_2)$ such that $\alpha_1|_{V_{OBS}}$ and $\alpha_2|_{V_{OBS}}$ are consistent. Let $m \in V_{mode} \setminus \big(\text{Sco}(\text{Ext}(pfm_1)) \cup \text{Sco}(\text{Ext}(pfm_2))\big)$

and let pfm'_1 and pfm'_2 be two (possibly equal) partial fault modes such that $\mathrm{Sco}(pfm'_1) = \mathrm{Sco}(pfm'_2) = \{m\}$.

We are now proving that if pfm'_1 is not discriminable from pfm'_2 then $pfm_1 \wedge pfm'_1$ is not discriminable from $pfm_2 \wedge pfm'_2$.

α_1 is consistent with M, pfm'_1, pfm'_2, and $\alpha_2|_{V_{OBS}}$. Moreover, α_2 is consistent with M, pfm'_1, pfm'_2, and $\alpha_1|_{V_{OBS}}$. Consequently, the following constraints are satisfiable:

- $C_1 \equiv M \wedge \alpha_1 \wedge pfm'_1 \wedge \alpha_2|_{V_{OBS}}$
- $C_2 \equiv M \wedge \alpha_2 \wedge pfm'_2 \wedge \alpha_1|_{V_{OBS}}$

Recall that $\alpha_1|_{V_{OBS}}$ is consistent with $\alpha_2|_{V_{OBS}}$. If pfm'_1 and pfm'_2 are equal or undiscriminable, then $(M \wedge pfm'_1)|_{V_{OBS}}$ is consistent with $(M \wedge pfm'_2)|_{V_{OBS}}$. Consequently, $C_1|_{V_{OBS}}$ and $C_2|_{V_{OBS}}$ are consistent, which makes $pfm_1 \wedge pfm'_1$ undiscriminable from $pfm_2 \wedge pfm'_2$.

On the other hand, if pfm'_1 and pfm'_2 are discriminable, then it comes from theorem 5.1 that $pfm_1 \wedge pfm'_1$ and $pfm_2 \wedge pfm'_2$ are discriminable. ■

5.3.5 Algorithm

We now detail the algorithm that allows to perform distributed diagnosability analysis using constraint-based models and limiting the constraint propagation by using partial fault mode discriminability analysis.

$\textsc{Output} = \emptyset$; $\textsc{Disc}_1 = \emptyset$; $k = 1$;
$\textsc{ToDo}_1 = \{(m = ok, m = ab) \mid m \text{ mode variable}\}$;
while $(k \leq maxrank \wedge \textsc{ToDo}_k \neq \emptyset)$
  for each pair $(pfm_1, pfm_2) \in \textsc{ToDo}_k$
    $\textsc{Ext}_1 = \textbf{Extend}(pfm_1)$; $\textsc{Ext}_2 = \textbf{Extend}(pfm_2)$;
    $\textbf{AddDisc}(\textsc{Disc}_k, \textsc{Ext}_1, \textsc{Ext}_2, pfm_1, pfm_2)$;
    $\textbf{AddToDo}(\textsc{ToDo}_{k+1}, \textsc{Ext}_1, \textsc{Ext}_2, pfm_1, pfm_2)$;
  $\textsc{Output} = \textsc{Output} \cup \textsc{Disc}_k$;
  $\textsc{Disc}_{k+1} = \textbf{Update}(\textsc{Disc}_k, k+1)$;
  $\textsc{ToDo}_{k+1} = \textsc{ToDo}_{k+1} \setminus \textsc{Disc}_{k+1}$;
return $\textbf{Expand}(\textsc{Output}, k)$;

The algorithm has a main loop that proceeds rank by rank until either the maximum rank has been reached, or all the pairs of partial fault modes of higher ranks need not be analyzed thanks to the properties. At the end of the algorithm Output contains the set of all pairs of discriminable alternative

partial fault modes of all ranks (including those that the algorithm did not explicitly analyze).

At iteration k, ToDo_k contains the set of pairs of alternative partial fault modes of rank k that should be analyzed for discriminability. The goal of iteration k is to find discriminable pairs of rank k, adding them to the output set, and to prepare the pairs that should be analyzed during iteration $k+1$. For these reasons it computes two sets: Disc_k (discriminable pairs of rank k) and ToDo_{k+1} (pairs to be analyzed in the next iteration).

The **Extend** function takes as input a partial fault mode *pfm* and returns a complete set of admissible extensions for *pfm*. This complete set of admissible extensions is either computed by constraint propagation as described in section 5.3.3, or obtained by combining the extensions of rank 1 partial fault modes, as allowed by theorem 5.2.

The function **AddDisc** checks the discriminability of a pair using theorem 5.3 and if discriminable adds it to the Disc_k set.

The **AddToDo** function is called for undiscriminable pairs, and computes the pairs of extensions to be checked using theorem 5.4 as follows. Let us denote by D the common scope of pfm_1 and pfm_2. Then for each partial assignment $\alpha_1 \in \text{Ext}_1, \alpha_2 \in \text{Ext}_2$, if α_1 or α_2 assigns a value to a mode variable $m \notin D$, then the following set of refinements is added to ToDo_{k+1}:

$$\{(pfm_1 \wedge (m = v_1), pfm_2 \wedge (m = v_2)) \mid v_{1,2} \in \{ok, ab\}\}$$

Finally, the set Disc_{k+1}, representing refinements of rank $k+1$ of discriminable pairs, is computed from Disc_k and subtracted from ToDo_{k+1}, since these pairs are trivially discriminable. Disc_k is added to the final output set.

Since the analysis, thanks to search space pruning, could end before reaching the maximum rank, the final output set is obtained by expanding all the discriminable pairs found during the loop to higher ranks.

5.3.6 Example

The notions described in this section are illustrated by a small example. The way in which we model components in this example is taken from [Ardissono *et al.*, 2005], although we give here a simplified version.

In the example there are two local diagnosers LD_1 and LD_2, in charge of three and two components respectively. The diagnosability analysis reasoning is distributed on those two local diagnosers, that communicate with a Supervisor. Each component model has a *mode* variable, representing the correctness status of the component, and one or more *data variables* that represent the correctness status of input and/or output data.

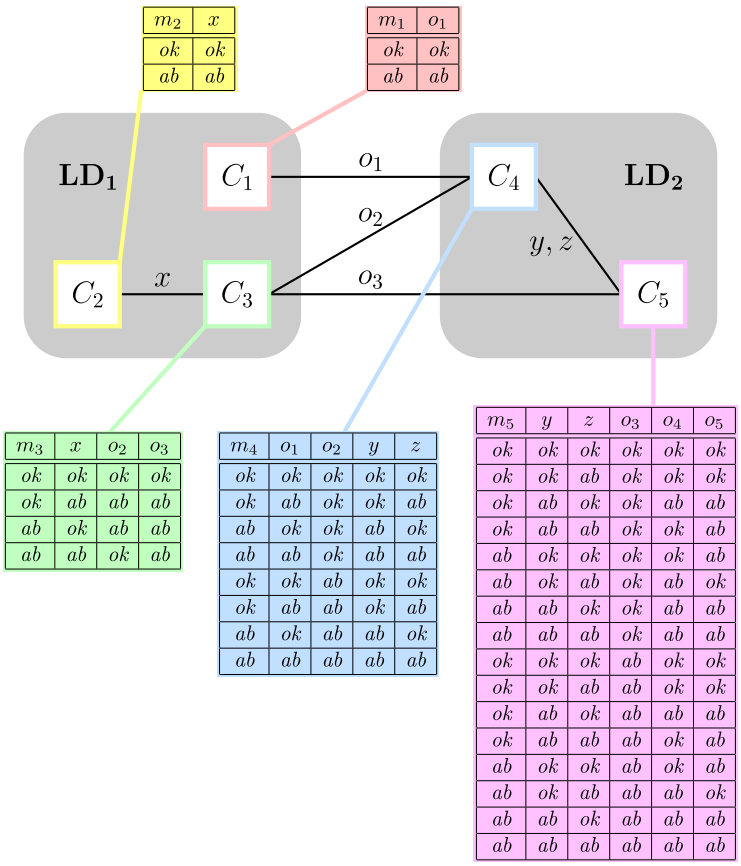


Figure 5.3: A distributed constraint-based model. Two local diagnosers LD_1 and LD_2 are in charge of respectively 3 and 2 components. The model of a component is represented by a constraint and defined by the list of the satisfying variable tuples. An implicit equality constraint exists between the variables of different components that have the same name, for example $C_2.x = C_3.x$.

Each component is modelled as a constraint expressing how the correctness of output data depends on the correctness of the inputs and of the component itself. For example, component C_3 has one inputs and two outputs. The model states that if either the component or the input is incorrect, then both the outputs are incorrect as well. However, if *both* the activity and the input are incorrect, then only output o_3 is incorrect (o_2 is correct because the two abnormalities mask each other).

The composed model is typically obtained by adding equality constraints between connected variables; in order to keep the example as simple as possible we directly used the same variable name rather than adding an equality constraint.

In this example we apply the algorithm defined in the previous section to the system described in figure 5.3. In a slight abuse of notation, "m_i is diag-

nosable" is used to express that $m_i = ok$ and $m_i = ab$ are discriminable, hence diagnosable.

Rank 1

pfm	mode vars					observable vars				
	m_1	m_2	m_3	m_4	m_5	o_1	o_2	o_3	o_4	o_5
$m_1 = ok$	ok	$*$	$*$	ok	ok	ok	$*$	$*$	ok	ok
	ok	$*$	$*$	ok	ab	ok	$*$	$*$	ok	ab
	ok	$*$	$*$	ab	$*$	ok	$*$	$*$	ab	ab
$m_1 = ab$	ab	$*$	$*$	ok	ok	ab	$*$	$*$	ok	ok
	ab	$*$	$*$	ok	ab	ab	$*$	$*$	ab	ok
	ab	$*$	$*$	ab	ok	ab	$*$	$*$	ok	ab
	ab	$*$	$*$	ab	ab	ab	$*$	$*$	ab	ab
$m_2 = ok$	$*$	ok	ok	$*$	$*$	$*$	ok	ok	$*$	$*$
	$*$	ok	ab	$*$	$*$	$*$	ab	ab	$*$	$*$
$m_2 = ab$	$*$	ab	ok	$*$	$*$	$*$	ab	ab	$*$	$*$
	$*$	ab	ab	$*$	$*$	$*$	ok	ab	$*$	$*$
$m_3 = ok$	$*$	ok	ok	$*$	$*$	$*$	ok	ok	$*$	$*$
	$*$	ab	ok	$*$	$*$	$*$	ab	ab	$*$	$*$
$m_3 = ab$	$*$	ok	ab	$*$	$*$	$*$	ab	ab	$*$	$*$
	$*$	ab	ab	$*$	$*$	$*$	ok	ab	$*$	$*$
$m_4 = ok$	$*$	$*$	$*$	ok	ok	$*$	$*$	$*$	ok	ok
	ok	$*$	$*$	ok	ab	ok	$*$	$*$	ok	ab
	ab	$*$	$*$	ok	ab	ab	$*$	$*$	ab	ok
$m_4 = ab$	$*$	$*$	$*$	ab	ab	$*$	$*$	$*$	ab	ab
	ok	$*$	$*$	ab	$*$	ok	$*$	$*$	ab	ab
	ab	$*$	$*$	ab	ok	ab	$*$	$*$	ok	ab
$m_5 = ok$	$*$	$*$	$*$	ok	ok	$*$	$*$	$*$	ok	ok
	ok	$*$	$*$	ab	ok	ok	$*$	$*$	ab	ab
	ab	$*$	$*$	ab	ok	ab	$*$	$*$	ok	ab
$m_5 = ab$	$*$	$*$	$*$	ab	ab	$*$	$*$	$*$	ab	ab
	ok	$*$	$*$	ok	ab	ok	$*$	$*$	ok	ab
	ab	$*$	$*$	ok	ab	ab	$*$	$*$	ab	ok

Figure 5.4: The admissible extensions of all partial fault modes of rank 1 restricted to mode and observable variables.

At step one the Supervisor computes a complete set of admissible extensions of all partial mode assignments of rank 1 (see figure 5.4). The information gathered at this stage suffices for the rest of the analysis, and the Supervisor does not need to invoke the local diagnosers anymore. As we said earlier, only observable and mode variables are kept, while interface variables are discarded (none in this case, since all interface variables are observable, see figure 5.3).

At this point the Supervisor starts to perform the diagnosability analysis from rank 1. Looking at the observable variables, we see m_1 and m_4 are diagnosable, since the extensions of alternative pairs restricted to observable variables are not consistent. Thus, the pair of assignments $(m_1 = ok, m_1 = ab)$ (resp. $(m_4 = ok, m_4 = ab)$) is inserted in the DISC_1 set. As a consequence, each refinement of $m_1 = ok$ (resp. $m_4 = ok$) is discriminable from each refinement of

$m_1 = ab$ (resp. $m_4 = ab$). These pairs of refinements are inserted in the DISC_2 set for further use.

Continuing with rank 1 analysis, the algorithm finds that:

- m_2 is not diagnosable (considering only restrictions to observable variables, the 2nd extension of $m_2 = ok$ is consistent with the 1st extension of $m_2 = ab$)
- pairs of partial fault modes in the scope $\{m_2, m_3\}$ need to be checked, since m_3 is present in the extensions of m_2 (property 5.4).

Therefore, the algorithm inserts in the TODO_2 all pairs of partial fault modes in the scope $\{m_2, m_3\}$:

$$(m_2 = ok \wedge m_3 = ok, m_2 = ok \wedge m_3 = ab)$$
$$(m_2 = ok \wedge m_3 = ok, m_2 = ab \wedge m_3 = ok)$$
$$(m_2 = ok \wedge m_3 = ok, m_2 = ab \wedge m_3 = ab)$$
$$(m_2 = ok \wedge m_3 = ab, m_2 = ab \wedge m_3 = ok)$$
$$(m_2 = ok \wedge m_3 = ab, m_2 = ab \wedge m_3 = ab)$$
$$(m_2 = ab \wedge m_3 = ok, m_2 = ab \wedge m_3 = ab)$$

Analyzing the last fault mode variable, m_5, the algorithm determines that it is also non diagnosable (considering only restrictions to observable variables, the 2nd extension of $m_5 = ok$ is consistent with the 1st extension of $m_5 = ab$), and that it needs to check at rank 2 the pairs of partial fault modes with scopes $\{m_1, m_5\}$ or $\{m_4, m_5\}$.

Some of those pairs are contained into the DISC_2 set, and do not need to be checked, since m_1 and m_4 are diagnosable. Therefore only the following four combinations are inserted in TODO_2:

$$(m_5 = ok \wedge m_1 = ok, m_5 = ab \wedge m_1 = ok)$$
$$(m_5 = ok \wedge m_1 = ab, m_5 = ab \wedge m_1 = ab)$$
$$(m_5 = ok \wedge m_4 = ok, m_5 = ab \wedge m_4 = ok)$$
$$(m_5 = ok \wedge m_4 = ab, m_5 = ab \wedge m_4 = ab)$$

Rank 2

At this stage, rank 2 analysis can start: combining the results of extend at rank 1, all extensions of the partial fault modes contained in TODO_2 are computed (see figure 5.5).

Examining the six pairs of partial fault modes assigning m_2 and m_3, the algorithm can determine that they are all discriminable except $m_2 = ok \wedge m_3 = ab$ and $m_2 = ab \wedge m_3 = ok$. The relative extensions do not mention any other mode variables, therefore the algorithm concludes that refinements of $(m_2 = ok \wedge m_3 = ab)$ are not discriminable from refinements of $(m_2 = ab \wedge m_3 = ok)$.

pfm	mode variables					observable variables				
	m_1	m_2	m_3	m_4	m_5	o_1	o_2	o_3	o_4	o_5
$(m_2 = ok \wedge m_3 = ok)$	$*$	ok	ok	$*$	$*$	$*$	ok	ok	$*$	$*$
$(m_3 = ok \wedge m_3 = ab)$	$*$	ok	ab	$*$	$*$	$*$	ab	ab	$*$	$*$
$(m_2 = ab \wedge m_3 = ok)$	$*$	ab	ok	$*$	$*$	$*$	ab	ab	$*$	$*$
$(m_2 = ab \wedge m_3 = ab)$	$*$	ab	ab	$*$	$*$	$*$	ok	ab	$*$	$*$
$(m_5 = ok \wedge m_1 = ok)$	ok	$*$	$*$	ab	ok	ok	$*$	$*$	ab	ab
	ok	$*$	$*$	ok	ok	ok	$*$	$*$	ok	ok
$(m_5 = ab \wedge m_1 = ok)$	ok	$*$	$*$	ok	ab	ok	$*$	$*$	ok	ab
	ok	$*$	$*$	ab	ab	ok	$*$	$*$	ab	ab
$(m_5 = ok \wedge m_1 = ab)$	ab	$*$	$*$	ok	ok	ab	$*$	$*$	ok	ok
	ab	$*$	$*$	ab	ok	ab	$*$	$*$	ok	ab
$(m_5 = ab \wedge m_1 = ab)$	ab	$*$	$*$	ok	ab	ab	$*$	$*$	ab	ok
	ab	$*$	$*$	ab	ab	ab	$*$	$*$	ab	ab
$(m_5 = ok \wedge m_4 = ok)$	$*$	$*$	$*$	ok	ok	$*$	$*$	$*$	ok	ok
$(m_5 = ok \wedge m_4 = ab)$	ok	$*$	$*$	ab	ok	ok	$*$	$*$	ab	ab
	ab	$*$	$*$	ab	ok	ab	$*$	$*$	ok	ab
$(m_5 = ab \wedge m_4 = ok)$	ok	$*$	$*$	ok	ab	ok	$*$	$*$	ok	ab
	ab	$*$	$*$	ok	ab	ab	$*$	$*$	ab	ok
$(m_5 = ab \wedge m_4 = ab)$	$*$	$*$	$*$	ab	ab	$*$	$*$	$*$	ab	ab

Figure 5.5: The admissible extensions of rank 2 partial fault modes contained in the ToDo_2 set.

Examining the 4 pairs of partial fault modes assigning m_5 and m_1 or m_4, the algorithm can determine that $m_5 = ok \wedge m_1 = ab$, $m_5 = ab \wedge m_1 = ab$, $m_5 = ok \wedge m_4 = ok$ and $m_5 = ab \wedge m_4 = ok$ are diagnosable. On the other hand, the pairs $(m_5 = ok \wedge m_1 = ok, m_5 = ab \wedge m_1 = ok)$ and $(m_5 = ok \wedge m_4 = ab, m_5 = ab \wedge m_4 = ab)$ are not discriminable, their extensions mention respectively m_4 and m_1. The algorithm puts in the ToDo_3 set the following combinations:

$$(\, m_5{=}ok \wedge m_1{=}ok \wedge m_4{=}ok \;,\; m_5{=}ab \wedge m_1{=}ok \wedge m_4{=}ok \,)$$
$$(\, m_5{=}ok \wedge m_1{=}ok \wedge m_4{=}ab \;,\; m_5{=}ab \wedge m_1{=}ok \wedge m_4{=}ab \,)$$
$$(\, m_5{=}ok \wedge m_1{=}ab \wedge m_4{=}ab \;,\; m_5{=}ab \wedge m_1{=}ab \wedge m_4{=}ab \,)$$

In fact, pairs with different values for m_1 or m_4 are discarded due to property 5.1 (m_1 and m_4 being diagnosable, these combinations are in Disc_3).

pfm	mode variables					observable variables				
	m_1	m_2	m_3	m_4	m_5	o_1	o_2	o_3	o_4	o_5
$(m_5 = ok \wedge m_1 = ok \wedge m_4 = ok)$	ok	$*$	$*$	ok	ok	ok	$*$	$*$	ok	ok
$(m_5 = ab \wedge m_1 = ok \wedge m_4 = ok)$	ok	$*$	$*$	ok	ab	ok	$*$	$*$	ok	ab
$(m_5 = ok \wedge m_1 = ok \wedge m_4 = ab)$	ok	$*$	$*$	ab	ok	ok	$*$	$*$	ab	ab
$(m_5 = ab \wedge m_1 = ok \wedge m_4 = ab)$	ok	$*$	$*$	ab	ab	ok	$*$	$*$	ab	ab
$(m_5 = ok \wedge m_1 = ab \wedge m_4 = ab)$	ab	$*$	$*$	ab	ok	ab	$*$	$*$	ok	ab
$(m_5 = ab \wedge m_1 = ab \wedge m_4 = ab)$	ab	$*$	$*$	ab	ab	ab	$*$	$*$	ab	ab

Figure 5.6: The admissible extensions of rank 3 partial fault modes contained in the ToDo_3 set.

Rank 3

By checking the observable variables for each pair (see figure 5.6), the algorithm finds that the 1^{st} and the 3^{rd} pairs are discriminable which makes the respective partial fault modes diagnosable, while the 2^{nd} pair is not discriminable. Since the extensions do not constrain other fault modes, all the pair refinements are not discriminable as well. Therefore the algorithm stops at rank 3.

Results

Two partial fault modes are not discriminable if and only if they refine the pairs $(m_2 = ok \wedge m_3 = ab, m_2 = ab \wedge m_3 = ok)$ or $(m_1 = ok \wedge m_4 = ab \wedge m_5 = ok, m_1 = ok \wedge m_4 = ab \wedge m_5 = ab)$. Partial fault modes that do not refine any of the 2 above are diagnosable. In terms of fault modes, 20 fault modes are not diagnosable (80 non discriminable pairs can be built), and 12 are diagnosable.

5.4 Conclusion

This chapter has shown that the concept of fault signature can be extended to partial fault modes, which can decrease the number of signature comparison in order to analyze diagnosability. Moreover, expressing the results of diagnosability analysis in terms of pairs of discriminable partial fault modes is smaller, and we claim that this format is more suited to be fed back to the designer.

Although the concepts of fault mode and signature apply to both SBS and EBS, the algorithm described here applies to SBS. We are confident that a similar approach can be done for EBS, although the problem of computing signatures for partial fault modes in a tractable way is open.

Chapter 6

Signatures for state dependent properties

This chapter presents a generalization of diagnosability not only to fault modes and partial fault modes, but to any set of states of a SBS. In a state based modeling framework, any condition or property of the system generally expresses as a set of states. This generalization hence allows one to analyze the diagnosability of, for instance, repair preconditions or of a given quality of service.

Diagnosis and diagnosability analysis are now mature research fields, and the problem of integrating diagnosis into a general purpose supervision tool is receiving increasing interest. The integration of diagnosability analysis into a general design support tool is also a very significant issue [wsdiamond, 2005 2008, Cordier *et al.*, 2007].

In [Cordier *et al.*, 2007], a new definition of diagnosability is provided, based on the idea that some faults need not be discriminated, for example because the same repair can repair them. This consideration has led to the definition of macrofaults for which diagnosability definition is a generalization of our unified diagnosability definition 3.9. This work was developed during this thesis, more precisely after the work described in chapter 4, and before the one described in this chapter. Then, our new definition of diagnosability is given, and we prove that it is a generalization of both our unified definition 3.9 and the one given in [Cordier *et al.*, 2007]. Finally, an example illustrates how this definition can greatly facilitate the integration of diagnosability and diagnosis into a general supervision tool.

6.1 Macrofault diagnosability

In [Cordier *et al.*, 2007], the idea that a repair may repair several fault modes is addressed by the definition of *macrofaults*. A definition of diagnosability of macrofaults is proposed, this definition is used for defining formally self-healability. This work follows the results of chapter 4, and accounts for the unified definition of diagnosability for fault modes. The diagnosability definition is based on the idea that *for repair, not all pairs of fault modes need to be discriminable*: the set of available repairs may have a coarser granularity than elementary faults. Then the fault modes that do not need to be discriminated one from another are gathered into a macrofault.

This raises a significant difference compared to the fault mode approach. Whereas fault modes are disjoint, macrofaults may overlap if a fault mode belongs to two macrofaults. In the macrofault approach, it is considered that when the system state belongs to several macrofaults, it belongs to an overlapping fault mode, and identifying one of the macrofaults with certainty is enough for the system to be diagnosable.

In this approach, only covering sets of macrofaults are considered, i.e. sets of macrofaults such that every fault mode belongs to at least one macrofault. Consequently, whatever the system state is, at least one of the macrofaults is present.

Definition 6.1 (Macrofault, Characteristic signature) *A* macrofault MF_i *is a set of fault modes.* MF_i *is present if and only if the system is in one of the fault modes* $f_j \in MF_i$*.*

A characteristic signature $cSig(MF_i)$ *is a set of observations that allow one to assess with certainty that the macrofault* MF_i *is present.*

$$cSig(MF_i) \subseteq \Big(\bigcup_{f_j \in MF_i} \big(Sig(f_i)\big) \setminus \bigcup_{f_k \notin MF_i} \big(Sig(f_k)\big) \Big)$$

Note that there are several possible characteristic signatures for each macrofault. If O is a characteristic signature for a macrofault MF_i, then any $O' \subseteq O$ is also a characteristic signature for MF_i.

Definition 6.2 (Macrofault Diagnosability) *A covering set of macrofaults* $\{MF_i\}$*, i.e. a set of macrofaults that cover all the fault modes, is diagnosable if and only if there exists a set of characteristic signatures for these macrofaults that form a partition of* OBS*.*

When such a partition is established as illustrated in figure 6.1, it is always possible to find out at least one present macrofault. As a state may belong to several macrofaults (this is the case for all states in f_2 in figure 6.1), an observation can also correspond to several macrofaults. However, it is only needed, for each observation, to assess with certainty that one macrofault is present. Being aware of all the present macrofaults allows the supervisor to

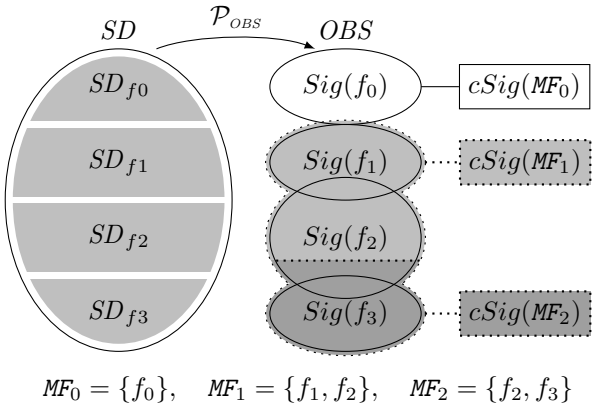


Figure 6.1: Macrofault diagnosability: the fault modes f_1, f_2, f_3 are not diagnosable. The set of macrofaults $\{MF_0, MF_1, MF_2\}$ is diagnosable.

choose and plan the repair, but this is not needed to achieve diagnosability of macrofaults.

This definition is a generalization of the unified diagnosability definition 3.9, since fault modes are particular macrofaults. Because macrofaults may overlap when they contain a common fault mode, this definition applies to a greater range of sets of states than the fault mode definition.

Macrofault diagnosability is less constrained than fault mode diagnosability (definition 3.9), in the sense that in a system verifying fault mode diagnosability, any covering set of macrofaults is diagnosable.

6.2 Diagnosability revisited

This section presents a new definition of diagnosability, which applies to any set of states, that may correspond to any state dependent property. It is based upon the analysis of the set of states in which a property holds. It is a generalization of existing diagnosability definitions which only apply to sets of states characterized by the presence or absence of some faults. Comparisons show that this new definition is consistent with the existing ones.

6.2.1 System representation

The system is assumed to be described by a formula *sd* which can be expressed in propositional logic. The set of variable tuples for which *sd* is satisfied is denoted *SD*, it contains all variable tuples satisfying *sd* and describes the set of all the system states, faulty or non faulty. We recall that the set of variables

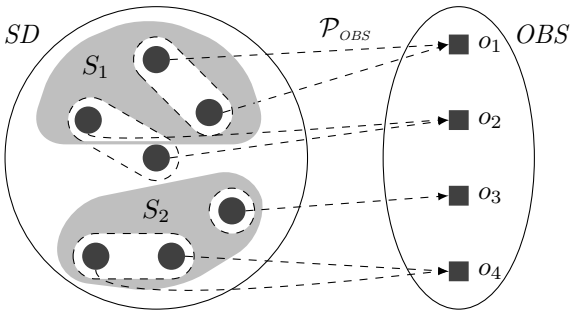


Figure 6.2: The set of states S_1 is not diagnosable. S_2 is diagnosable.

is denoted V, O denotes the set of *observable variables*, and the set OBS contains all the reachable tuples of observable variables. In other words, it contains the restrictions of the variable tuples $SD$to the variables in O. The projection on observables $\mathcal{P}_{OBS}$ is in this context a function from SD to OBS that associates each state s of SD to its corresponding observation in OBS by removing unobservable variables from the variable tuple defining s.

We have seen before that fault modes can be described by an assignment to mode variables (see definition 3.5 page 62) which is a particular case of formula. A fault mode can hence be associated to a set of states. The notation SD_{f_i} denotes the set of states associated to the fault mode f_i. The signature of a fault mode is defined as the projection on observables of its set of states:

$$\forall f_i \in \mathcal{F}_{sys}, \quad Sig(f_i) = \mathcal{P}_{OBS}(SD_{f_i})$$

6.2.2 Diagnosability of a set of states

We now introduce a new definition of diagnosability. This definition is stated in terms of *states* instead of observables and signatures. This aspect is original and provides a dual characterization of the classical definitions, that can prove useful is some contexts where characterizing the set of observables is more difficult than the set of states.

Definition 6.3 (Diagnosable block) *Let $=_{OBS}$ be the equivalence relation defined on SD by:*

$$\forall s_1, s_2 \in SD, s_1 =_{OBS} s_2 \Leftrightarrow \mathcal{P}_{OBS}(s_1) = \mathcal{P}_{OBS}(s_2)$$

Each equivalence class of $=_{OBS}$ is called a diagnosable block *of the system. The set of diagnosable blocks of the system is the quotient set of SD by $=_{OBS}$.*

Definition 6.4 (Generalised Diagnosability) *A property or its corresponding set of states $S \subseteq SD$ is diagnosable if and only if S is exactly a union of diagnosable blocks.*

Figure 6.2 depicts a system with 7 states and 4 possible observations. The diagnosable blocks are represented by white sets with dashed lines. Observation

o_2 is received in two different states, one inside S_1 and one outside. Thus, when observing o_2, a supervisor is unable to decide whether the system is in S_1 or not. On the other hand, it is always possible to decide from the observations whether the system state belongs to S_2 or not.

6.2.3 Comparison with unified diagnosability

Since definition 6.4 applies to any set of states, it applies in particular to fault modes. It is shown now that when applied to fault modes, this definition is equivalent to the unified diagnosability definition 3.9.

Theorem 6.1 *A system is diagnosable according to the unified definition 3.9 if and only if for every fault mode f, SD_f is diagnosable according to definition 6.4.*

Proof Assume that the system is not diagnosable in the sense of definition 3.9. There exists two fault modes f_i and f_j whose signature intersect, i.e. there exists a state $s_i \in SD_{fi}$ and another state $s_j \in SD_{fj}$ leading to the same observation. These two states obviously belong to the same diagnosable block, say d, and, since SD_{fi} and SD_{fj} are disjoint, none is a superset of d. Since diagnosable blocks form a partition of SD, s_i (resp. s_j) does not belong to any other diagnosable block than d. Hence, SD_{fi} (resp. SD_{fj}) is not a union of diagnosable blocks, and thus is not diagnosable according to definition 6.4.

Assume now that one fault mode f_i is not diagnosable as of definition 6.4, i.e. SD_{f_i} is not a union of diagnosable blocks. Then since both diagnosable blocks and fault modes states form a partition of SD, there exists a diagnosable block d containing a state s_i of SD_{f_i} and at least one state s_j belonging to another fault mode SD_{f_j}. These two states lead to the same observation o, which necessarily belongs to both $Sig(f_i)$ and $Sig(f_j)$. Consequently the signatures of all fault modes are not all pairwise disjoint, i.e. the system is not diagnosable according to definition 3.9. ■

6.2.4 Signature and preemptability

Definition 6.4 expresses the diagnosability of a single property. This definition is now extended to a set of properties. For this, the classical notion of signature is extended and the notion of preemptability is introduced. The new definition of the signature applies to sets of states as opposed to the unified definition 3.9 that applies to fault modes.

Definition 6.5 (Signature of a set of states) *The signature of a set of states S, or of the property p mapped to S, is the set of observations that can be obtained when the system is under one of these states:*

$$Sig(S) = \{\mathcal{P}_{OBS}(s), s \in S\}$$

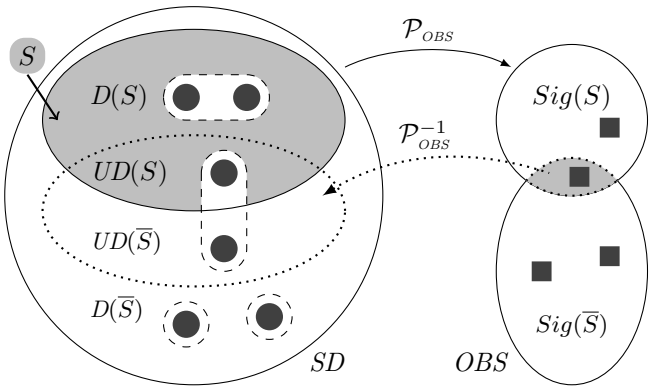


Figure 6.3: Signature $Sig(S)$, diagnosable space $D(S)$ and undiagnosable space $UD(S)$ of a set of states S.

This definition applies equally to the complement set $\overline{S}$. As sets of states generally overlap, comparing their signatures with one another does not bring much information. It is worthy to compare their signatures with the signatures of their respective complements. Indeed, if a set of states corresponds to a given property of the system, its complement corresponds to the negation of the property.

Definition 6.6 (Diagnosable space, Undiagnosable space) *The* diagnosable space $D(S)$ *(resp.* undiagnosable space $UD(S)$*) of a set of states S mapped to a property p is the subset of S in which it is possible (resp. impossible) to assert from the observations whether the property p holds.*

$$\begin{aligned} UD(S) &= S \cap \mathcal{P}^{-1}_{OBS}\big(Sig(S) \cap Sig(\overline{S})\big) \\ D(S) &= S \setminus UD(S) \end{aligned}$$

As illustrated in figure 6.3, $D(S)$ can also be defined as the union of the diagnosable blocks included in S. The diagnosable blocks that intersect but are not included in S form $UD(S) \cup UD(\overline{S})$. The intersection of this set with S gives $UD(S)$. Hence, when a set of states is diagnosable, its undiagnosable space is empty.

When a property p is undiagnosable, it can be preemptable if its undiagnosable space is included in the union of the diagnosable spaces of other properties. In this case these other properties may preempt p in the sense that when the validity of p is uncertain, one of these other properties is valid, which may be enough to take a decision.

Definition 6.7 (Preemptability) *A property or its corresponding set of states S is* preemptable *with respect to a set of properties $\{S_i | 1 \leq i \leq n\}$ if and only if:*

$$UD(S) \subseteq \bigcup_{i=1}^{n} \big(D(S_i)\big)$$

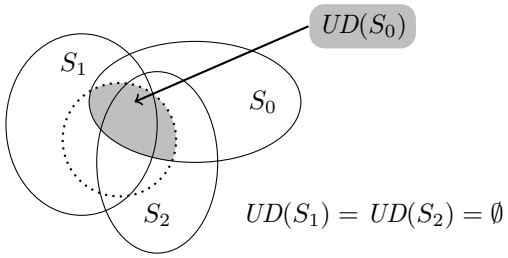


Figure 6.4: The set of states S_0 is preemptable.

Or equivalently:

$$\big(Sig(S) \cap Sig(\overline{S})\big) \subseteq \bigcup_{i=1}^{n} \big(Sig(S') \setminus Sig(\overline{S'})\big)$$

Figure 6.4 illustrates a set S_0 whose undiagnosable space is included into two diagnosable sets S_1 and S_2.

6.2.5 Diagnosability of a set of properties

This section presents a definition of diagnosability for a set of properties that accounts for the mutual influence that properties may have with one another by the means of preemptability.

Definition 6.8 (Diagnosability of a set of properties) *A set of properties is diagnosable if and only if each property is either diagnosable or preemptable with respect to this set.*

When this definition applies to a set of disjoint sets of states, then it falls back to definition 6.4 applied to each set. Moreover, if a set of sets of states is diagnosable according to definition 6.8, then the union of all its sets of states is diagnosable according to definition 6.4. Indeed, let S_i be the set of states corresponding to the i-th property of a diagnosable set of properties. For each i, $UD(S_i)$ is either empty or included in the union of the diagnosable sets of other sets of states. Hence, $\bigcup_i S_i = \bigcup_i D(S_i)$ is diagnosable since each $D(S_i)$ is a union of diagnosable blocks.

6.2.6 Comparison with macrofault diagnosability

Now it is shown that definition 6.8 is equivalent to definition 6.2 when applied to macrofaults.

Theorem 6.2 *A covering set of macrofaults is diagnosable according to definition 6.2 if and only if it is diagnosable according to definition 6.8.*

Proof First, given a macrofault MF_i, let us consider $Sig(D(MF_i))$. This set contains no observation from a state in which MF_i is absent, and is hence a characteristic signature for MF_i. Let us map each macrofault MF_i to the set $\Sigma_i = Sig(D(MF_i)) \setminus \bigcup_{j<i} Sig(D(MF_j))$. Σ_i is a characteristic signature for MF_i since it is a subset of $Sig(D(MF_i))$.

Second, let $o \in Sig(D(MF_i))$. We have either $o \notin \Sigma_i$, or $o \in \Sigma_j$ with $j < i$, and if $o \in \Sigma_i$ then necessarily $o \notin \Sigma_k$ with $k \neq i$. Hence, the set of all Σ_i forms a partition of the set $\bigcup_i Sig(D(MF_i))$.

The previous statements are now used to establish the equivalence. The covering set of macrofaults $\{MF_0 \dots MF_n\}$ is diagnosable according to definition 6.8 if and only if the set of all $D(MF_i)$ covers SD (see section 6.2.5), which is equivalent to the set $\bigcup_i Sig(D(MF_i))$ covering OBS. Consequently, from the statements above, it follows that the set $\Pi = \{\Sigma_0 \dots \Sigma_n\}$ partitions OBS, and the set of macrofaults is diagnosable according to definition 6.2. ■

6.3 Application to repair preconditions

The ultimate goal of diagnosis is to achieve a level of self-awareness that eases decision making about the actions to be undertaken to bring back the system into a nominal state. Diagnosis is hence driven by repair goals. This is why the definitions of diagnosability 6.4 and 6.8 are now applied to sets of states that map to repair preconditions. When a repair precondition is diagnosable, it is possible to decide when to apply the repair and when not. It is a complementary approach to fault diagnosis, the knowledge of which fault is present and the knowledge of how to repair it are different, and both are important for a system supervisor. Knowing which fault happened but being unable to decide which repair is suited is odd. On the other hand, knowing how to repair a faulty system without knowing the details of the faults is a problem for low cost maintenance or feedback to the system designers. Hence, diagnosability analysis of repair preconditions is a complement to fault diagnosability analysis.

A repair is an action or a process that puts a system back to a normal state from a faulty abnormal state. Repairs can be plans driven by goals [Friedrich *et al.*, 2005], reconfigurations [ten Teije *et al.*, 2004], or other actions. In most approaches, repairs have preconditions, which generally define a set of states. In the case of repair plans, the plan may contain actions that bring additional information about the system state, thus refining the diagnosis. Plans may also contain conditional branchings, especially in order to react to additional diagnosis information.

A repair may not be executed in every state of the system for various reasons. An action or plan that repairs a system from a given state may damage it even more in some other states. For example changing a wheel is not possible if the vehicle is not at full stop. Also, it is considered in this paper that repairs being the system back in a normal state, partial repairs are not considered. For

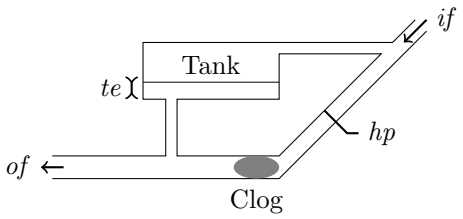


Figure 6.5: A pipe and a tank

example, changing one wheel repairs a vehicle with one flat tire, but it does not repair a vehicle with two flat tires.

In most cases, non faulty states do not belong to repair preconditions, since it is not useful to repair a normal system. However, when there is an ambiguity about the presence for a fault, some supervision policies consider that it is better to repair a normal system than let the system run with the fault. Consequently, normal states may belong to repair preconditions.

Definition 6.9 (Repair precondition) *A repair precondition is a set of states, in which the repair can be applied, and in which the application of the repair brings the system to a normal state.*

This definition implies that if two repair preconditions are verified at the same time, only one repair needs to be applied.

No assumption is made in this paper about the relation between fault modes and repair preconditions. A repair may be applicable in only some of the states of a fault mode, while each fault mode may be repaired differently according to the current system state.

Each repair precondition is described by a logic proposition rp_i. The proposition rp_i generally constrains mode variables as well as variables defining the operational state of the system. The set $\mathcal{RP}_i \subseteq SD$ contains all the system states fulfilling rp_i. The set $\overline{\mathcal{RP}}_i$ is the complement of $\mathcal{RP}_i$ in SD, it is the set of system states for which the i-th repair is not suited.

6.4 Example

The concepts and definitions described in the previous sections are illustrated by a simple example. It is shown that definitions 6.4 and 6.8 allow us to analyse diagnosability at different levels (faults, macrofaults, or repair preconditions) and that the returned information may be different and complementary.

System The system consists in a fluid pipe with variable input flow, which is supposed to provide a constant output flow. A tank is used to compensate

flow variations. This tank is filled when the input flow is higher that the expected output, and provides water when the input flow is too low.

Faults Two faults are considered. First, the pipe may be clogged, which reduces greatly the flow capacity of the pipe. Second, the tank is supposed to be always able to deliver water, however in exceptional conditions, the tank may occur to be empty. When this occurs, the input flow is directed in priority to the tank. If the input flow is sufficiently high, it can supply both the empty tank and the output.

Sensors A pressure sensor is placed in the pipe, in order to detect abnormally high pressures. This happens when the pipe is clogged, and there is input flow.

Model The model of this system contains five variables:

- *if* describes the input flow and has 3 values: *none*, *low*, and *high*.
- *of* is Boolean and equals 1 when there is an output flow.
- *hp* is Boolean and equals 1 when the pressure inside the pipe is abnormally high.
- *pc* is Boolean and equals 1 when the pipe is clogged.
- *te* is Boolean and equals 1 when the tank is empty.

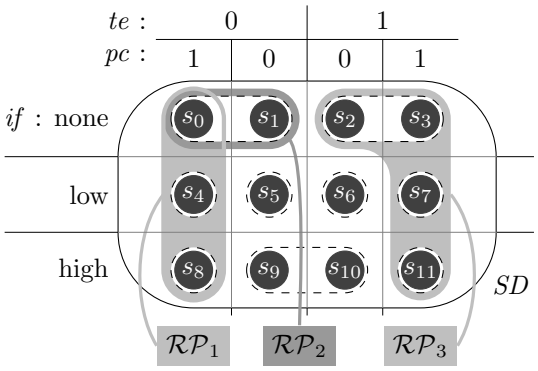
The behaviour of the system is described by the following constraints:

$$\begin{aligned} of = 1 &\Leftrightarrow \big(te = 0 \vee (if = \text{high} \wedge pc = 0)\big) \\ hp = 1 &\Leftrightarrow (if \neq \text{none} \wedge pc = 1) \end{aligned}$$

Observables The variables *if* , *of* and *hp* are observable.

Repairs The following repairs are available:

1. It is possible to unclog the pipe thanks to a chemical action ($rp1$). This repair can be applied when the pipe is clogged. For safety reasons, it must not be applied when the pipe is not clogged. If the tank is empty, this repair is not sufficient to bring back the system in a normal state.
$$rp_1 : (pc = 1 \wedge te = 0)$$
2. It is also possible to unclog the pipe mechanically ($rp2$). The action consists in sending someone on site and clean the pipe. This repair can only be applied when the pipe is empty (no input flow), but is not sufficient if the tank is empty, since it will not bring the system to a normal state. This repair can if necessary be applied in the normal mode: once the cleaner is on site, if the pipe is not clogged, then the cleaner will do nothing.
$$rp_2 : (te = 0 \wedge if = \text{none})$$
3. Finally, if the tank is empty, it is possible to redirect the whole flow through the tank ($rp3$). This permits to mechanically unclog the pipe if needed. However, the manipulations involved in this repair require that there is no flow in the pipe.
$$rp_3 : \big(te = 1 \wedge (if = \text{none} \vee pc = 1)\big)$$



States	Observation		
	if	*hp*	*of*
s_0, s_1	none	0	1
s_2, s_3	none	0	0
s_4	low	1	1
s_5	low	0	1
s_6	low	0	0
s_7	low	1	0
s_8	high	1	1
s_9, s_{10}	none	0	1
s_{11}	high	1	0

Figure 6.6: States, diagnosable blocks and repair plans of the system.

The system has 12 different states, represented in figure 6.6 as well as the diagnosable blocks and their corresponding observations. The application of definitions 6.4 and 6.8 to the fault modes, macrofaults and repair preconditions are now illustrated on this system.

6.4.1 Fault mode diagnosability analysis

The system has 4 fault modes, named *normal*, *pipe clogged*, *tank empty* and *pipe & tank* that define 4 sets of states whose diagnosability is analyzed. The analysis details are described in figure 6.7.

According to definition 6.4, none of these fault modes is diagnosable. Moreover, fault modes are by definition disjoint sets, and the notion of preemptability is not relevant when dealing with disjoint sets of states, since disjoint sets cannot preempt one another. Consequently, no fault mode is diagnosable according to definition 6.8 either.

Fault mode	States	Intersected blocks	*UD*
SD_{normal}	$\{s_1, s_5, s_9\}$	$\{s_0, s_1\}$ and $\{s_9, s_{10}\}$	$\{s_1, s_9\}$
$SD_{pipe\ clogged}$	$\{s_0, s_4, s_8\}$	$\{s_0, s_1\}$	$\{s_0\}$
$SD_{tank\ empty}$	$\{s_2, s_6, s_{10}\}$	$\{s_9, s_{10}\}$ and $\{s_2, s_3\}$	$\{s_2, s_{10}\}$
$SD_{pipe\ \&\ tank}$	$\{s_3, s_7, s_{11}\}$	$\{s_2, s_3\}$	$\{s_3\}$

Figure 6.7: Fault modes diagnosability results.

Macrofault	States	Intersected blocks	*UD*
MF_1	$\{s_1, s_2, s_5, s_6, s_9, s_{10}\}$	$\{s_0, s_1\}$ and $\{s_2, s_3\}$	$\{s_1, s_2\}$
MF_2	$\{s_0, s_4, s_8\}$	$\{s_0, s_1\}$	$\{s_0\}$
MF_3	$\{s_2, s_3, s_6, s_7, s_{10}, s_{11}\}$	$\{s_9, s_{10}\}$	$\{s_{10}\}$

Figure 6.8: Macrofaults diagnosability results.

Repair precondition	States	Intersected blocks	*UD*
$\mathcal{RP}_1$	$\{s_0, s_4, s_8\}$	$\{s_0, s_1\}$	$\{s_0\}$
$\mathcal{RP}_2$	$\{s_0, s_1\}$	none	$\emptyset$
$\mathcal{RP}_3$	$\{s_2, s_3, s_7, s_{11}\}$	none	$\emptyset$

Figure 6.9: Repair preconditions diagnosability results.

In each table, the column "Intersected blocks" lists the diagnosable blocks that intersect but are not subset of the corresponding set of states.

6.4.2 Macrofault diagnosability analysis

Let us consider for example the set of macrofaults defined by $\{MF_1, MF_2, MF_3\}$ with $MF_1 = \{$*normal*, *tank empty*$\}$ and $MF_2 = \{$*pipe clogged*$\}$ and $MF_3 = \{$*tank empty*, *pipe & tank*$\}$. The diagnosability analysis is given in Table 6.8.

None of these macrofaults is diagnosable with respect to definition 6.4. Moreover, only MF_3 is preemptable, with $UD(MF_3) \subset D(MF_1)$, which is not enough to make the set of macrofaults $\{MF_1, MF_2, MF_3\}$ diagnosable according to definition 6.8.

6.4.3 Repair precondition diagnosability analysis

The sets of states corresponding to the repair preconditions are represented in figure 6.6. Diagnosability analysis provides the results indicated in Table 6.9.

The undiagnosable spaces of repair preconditions $\mathcal{RP}_2$ and $\mathcal{RP}_3$ are empty, which means these sets of states are unions of diagnosable blocks. They are diagnosable, according to definition 6.4. Moreover, $\mathcal{RP}_1$ is not diagnosable, but $UD(\mathcal{RP}_1) \subset D(\mathcal{RP}_2)$, it is preemptable. The set of repair preconditions $\{\mathcal{RP}_1, \mathcal{RP}_2, \mathcal{RP}_3\}$ is diagnosable with respect to definition 6.8.

Part III

Application and algorithmic aspects

This part presents the application of the diagnosability approach described in chapter 5 to web services, or more generally to systems which comply to Service Oriented Architectures (SOA). The generalized definition provided in chapter 6 has not led yet to any application work since it has been designed during the implementation of the approach of chapter 5.

The application of the signature based diagnosability analysis is done in service oriented architectures, which prove to be a particularly constrained context. The system distribution, the high modularity and the privacy of some data make supervision difficult in such an environment. Our implementation uses partial assignments that allow us to comply to both distribution and privacy constraints, and we will see that our implementation offers a sufficient modularity to fit those architectures.

Chapter 7

Applicative context: Service Oriented Architectures and Web Services

Service Oriented Architectures (SOA) have recently benefited from an important interest from the industry. SOA are useful for process analysis, design or specification, in many kinds of processes: software, manufactures, or administration. The concepts introduced by SOA allows to express and manipulate functional knowledge (see section 1.3), which makes SOA an attractive paradigm for process analysis, particularly in environments with heterogeneous or evolutive components.

7.1 Introduction to Service Oriented Architectures

SOA originated in the context of software for commercial transactions on the internet. In such context, components are very heterogeneous and evolutive (hardware, software, web servers, data bases, etc), and a system can be involved in various transactions that fulfill various purposes. The only constant characteristic of a system in this context is the function, or *service*, it provides. This is why services (or functions) are the central concept of SOA. Atomic services, called *activities*, can be composed by a *business process* in order to provide an elaborated service. An execution, or instance, of a business process is a *workflow*. A new workflow is started each time a consumer, or client, *invokes* the service. The business process can also be called a workflow model.

Example 7.1 (Shop) *A shop offers two services. The first service receives a list of items, reserves them if available, and outputs some information about these items: reservation number, availability, price, shipment delay, etc. The second service performs the confirmation and payment of a reservation. It takes as input the reservation number and bank account coordinates, launches the shipment process and outputs a bill. The shipment process is delegated to another service.*

The payment service cannot be invoked at any time with any reservation number. If the reservation number is not valid, it will output an error. Hence, the two services are part of a unique business process.

In the example above, two services are linked by a unique business process. Actually, the business process that reserves items and the business process that receives the payment are two parts of a greater business process destined to sell items. This global business process is associated to a global selling service that encapsulates the reservation and payment services.

The business process realized by a service provider is generally not published to consumers. The provider only publishes an abstraction of its business process that allows consumers to invoke the service, like the structure of the request and response messages. Some more elaborated frameworks allow the provider to publish to consumers abstract processes, that describe several message exchanges, conditional branchings and such control structures. This abstraction leads to a great flexibility, since equivalent services can easily be substituted. However for model-based reasoning this is problematic since no entity is aware of the details of the whole process.

The two most important aspects of SOA when considering system supervision are the great modularity, and the model privacy. These aspects are developped in the next sections.

7.2 Orchestration and choreography

The possibility of having several business partners executing several parts of a business process leads to two critically different contexts for process analysis, called orchestration and choreography. The gap between boths context is not very clear. The position and definitions adopted here are defined in [Eder *et al.*, 2006], along with a formal meta-model of orchestrations and choreographies.

Definition 7.1 (Orchestration and choreography) *An* orchestration *is a business process executed by a unique service provider. A* choreography *is a business process executed by two or more service providers.*

Orchestrations and choreographies analysis have been addressed separately in the literature. The problems of choreographies conformance and

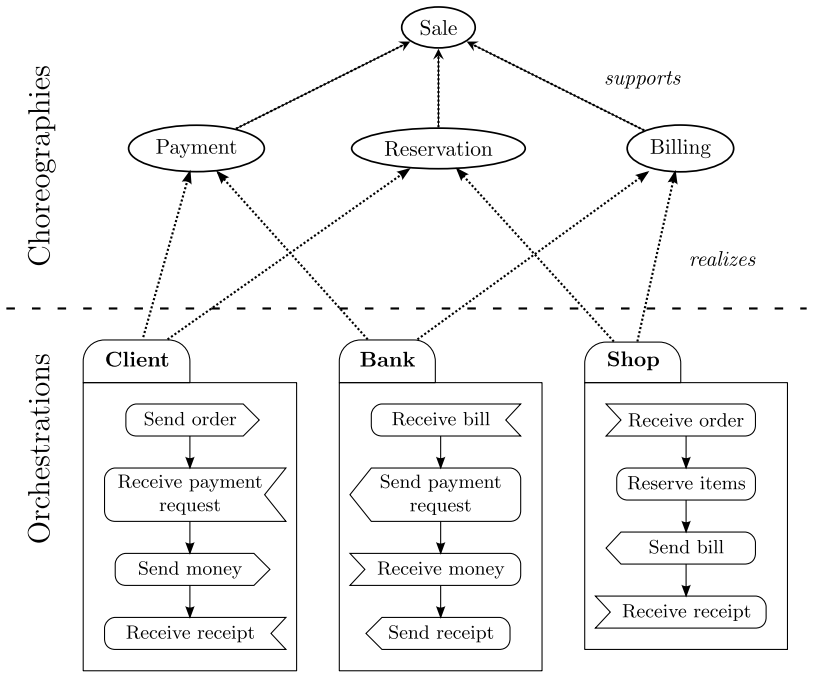


Figure 7.1: Composing orchestrations on different partners lead to choreographies. Conversely, the total sale choreography can be decomposed into various supporting choreographies, or further in orchestrations.

automated composition of orchestrations have been addressed many times [Hamadi and Benatallah, 2003, Baldoni *et al.*, 2004, van der Aalst *et al.*, 2006, Busi *et al.*, 2005]. Various languages address the problem of description and execution of business processes: BPEL4WS [OASIS, 2007], WS-CDL [W3C, 2005b], etc.

Both orchestrations and choreographies are business processes, and consequently are processes. There are various relations between orchestrations and choreographies, according to the context of the service providers:

- An orchestration can be split into several orchestrations (see example 7.1).
- An orchestration executed by a general service can be internally realized by a choreography of internal services.
- A choreography can be divided into orchestrations, by considering the processes executed by each partner. These orchestrations *realize* the choreography, see figure 7.2.
- A choreography can also be divided into other sub-choreographies involving less service providers. These sub-choreographies *support* the former choreography, see figure 7.2.

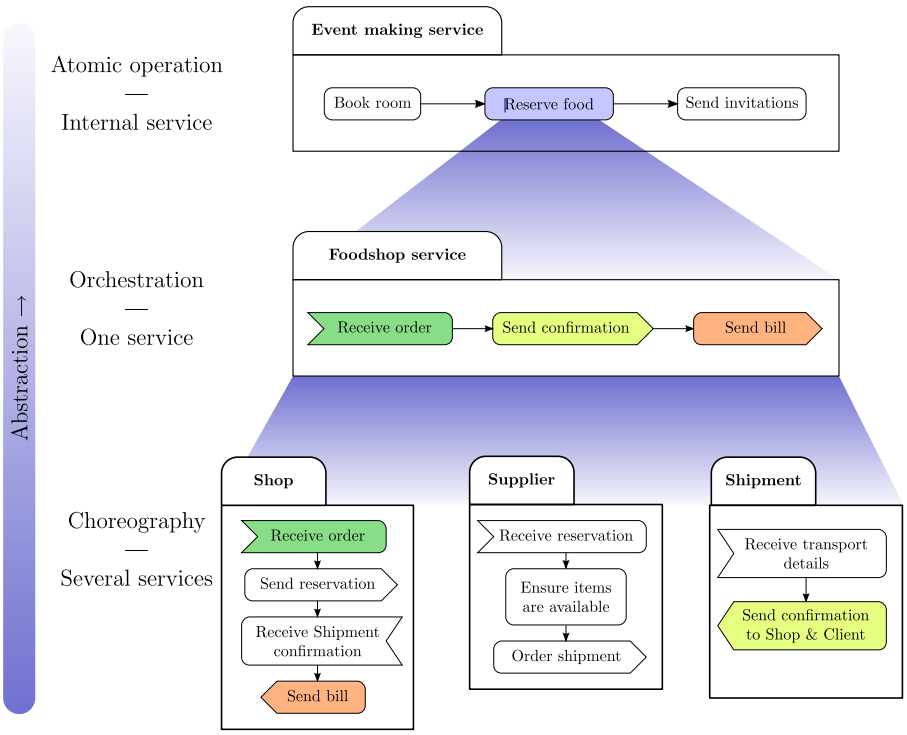


Figure 7.2: The same process, seen at different abstraction levels. At the lowest abstraction level, the 3 participants are considered as different partners, and the process is seen as a choreography. At a higher level of abstraction, the whole Foodshop is considered as a single service encapsulating the Shop, the Supplier and the Shipment; the process is then seen as an orchestration, and communication details are abstracted out. Finally, at a even higher level of abstraction, the process of buying food is seen as an atomic operation of which the details are not relevant for the model of the whole larger process of preparing a ceremony.

The distinction between choreography and orchestration totally depends on the definition of the different providers. For the sake of clarity and performance, several service providers can be abstracted into one, for example the departments of a society that deliver themselves various services can be abstracted into a single service, the society, when dealing with large external processes. The same business process can be seen, according to the abstraction level, as a choreography, an orchestration, or even an internal operation, as illustrated in figure 7.2.

A service provider has in general no restriction about its clients, and a service can be used by several consumers. A business process can consequently be part of several larger business processes, and share some sub-parts with other business processes.

SOA are used for modelling and analysing existing processes, but the most important application is for distributed software programming, especially web services.

7.3 Web services

The ambassador technology of SOA is the web services specification, that contains several languages based on XML and adopted as a standard by the W3C. The capacity of SOA to integrate components from different companies has made it very interesting for the management of commercial transactions over the Internet. This sector has been very dynamic these last few years.

The web services specification initially relies mainly on 3 languages: SOAP, WSDL and UDDI. These languages allow the specification of communication protocols for service invocation, and to provide a directory of services. Several other languages are candidate for standardization by the W3C, since the 3 standard languages do not allow the specification of business processes.

7.3.1 XML

The *eXtensible Markup Language* [W3C, 2006] was first specified in 1998 as a subset of SGML (Standard Generalized Markup Language). It is a markup language that allows users to specify their own markups. It is the base language for most standard and non standard languages used in specifications related to web services. It has a text format, and is designed to be human readable.

A fundamental extension of XML is the XPath (XML Path Language) that provides means to select nodes in an XML document based on its tree representation and extract values under string, integer, Boolean, and other formats. XML Schema [W3C, 2004] is another important extension for defining and validating schemas, using the language XSD (XML Schema Definition). A schema defines markup names, and constraints over the markups (for example, one may want the markup `<paragraph>` to appear inside the markup `<chapter>`). Since XSD is an extension of XML, it is defined in an XSD file. Another important extension is the XSLT language (XML Stylesheet Language Transformation) [W3C, 1999] that expresses transformation of XML documents into other documents, based on markup patterns or functional programming. XSLT is also an extension of XML and can be described in an XSD file. Parsers and transformers for XML files have been implemented in various programming languages.

Various other extensions of XML are used for many applications ranging over the description of text, images, mathematical formulas, general documents, or remote telescope control or even theology semantics, although the relevance of these extensions can vary as much as the topics they address. It is also used to specify many aspects of distributed programming in SOA.

```
<?xml version="1.0" encoding="UTF-8"?>
<BOOK xml:id="simple_book"
    xmlns="http://docbook.org/ns/docbook" version="5.0">
  <TITLE>Very simple book</TITLE>
  <CHAPTER xml:id="chapter_1">
    <TITLE>Chapter 1</TITLE>
    <PARA>Hello world!</PARA>
    <PARA>I hope that your day is proceeding
        <emphasis>splendidly</emphasis>!</PARA>
  </CHAPTER>
  <CHAPTER xml:id="chapter_2">
    <TITLE>Chapter 2</TITLE>
    <PARA>Hello again, world!</PARA>
  </CHAPTER>
</BOOK>
```

Figure 7.3: DocBook is an XML-based language used to describe text documents. Source: wikipedia [Wikipedia,].

7.3.2 SOAP

The *Simple Object Access Protocol* [W3C, 2007a] is a standard language based on XML. It describes message structures for data transfer between applications. In several aspects it is similar to HTML (HyperText Markup Language), the standard language used to describe web pages:

- It can be transported by HTTP[1] or SMTP[2] to its destination.
- The root SOAP markups `<soap:Envelope>`, `<soap:Header>`, `<soap:-Body>` are very similar to their structural equivalent in HTML `<html>`, `<head>` and `<body>`.

The main difference is that SOAP contains no predefined markups related to the data it encapsulates. HTML contains predefined markups that allow to describe a web page. In opposition, SOAP, as an XML-based language, is extensible, and the inner markups are defined by the designer according to the communication needs. SOAP can easily be treated using XSLT transformers, in order for example to add styling information and to produce a human readable web page.

7.3.3 WSDL

The *Web Service Description Language* [W3C, 2007b] is a standard language for describing web services interfaces. It is generally used together with SOAP and XSD for describing the structure of the input and output messages.

[1] HyperText Transfer Protocol: the standard protocol for querying web servers.
[2] Simple Mail Transfer Protocol: the standard protocol for reading and writing emails.

A web service description defines *interfaces*, the provided services and their associated messages. A *binding* links the messages defined by an interface to a communication protocol, e.g. SOAP. Finally, an *endpoint* maps a binding to an address, generally an URL[3]. In short, the same service may be accessed with different protocols at different URL addresses.

Several operation patterns are defined for interfaces:

In-Only The service receives an input and never answers.

Robust In-Only The service receives an input and may output fault messages, but no other type of message.

In-Out The service receives a request and outputs a response. A fault message may replace or complete the response.

In-Optional-Out The service receives an input, and may output a response and/or a fault message if necessary.

Out-Only The service sends a notification to the consumer and does not await answers.

Robust Out-Only The service sends a notification to the consumer and accepts responses containing fault messages.

Out-In The service notifies the consumer and awaits for a response. A fault messages replaces or completes the answer.

Out-Optional-In The service sends a notification to the consumer and accepts a response and/or a fault message if any.

The patterns of style "in-only" or "in-out" are suited for use with SOAP over HTTP protocol, in a client-server manner. The "out-only" or "out-in" patterns are more difficult to implement over HTTP, since some particular mechanisms like permanent connections are needed. When using SOAP over SMTP (i.e. in emails), or even in peer-to-peer networks, all those patterns are directly realizable.

Although the specification of WSDL 2.0 allows more complex patterns to be defined, only the eight above are part of the W3C recommendation and are likely to be understood by all programming tools. Consequently, WSDL is not able to describe stateful services realized by business processes.

7.3.4 UDDI

The *Universal Description Discovery and Integration* standard [OASIS, 2004] was originally proposed as a core web service standard. It provides a directory

[3]Uniform Resource Locator: a string describing the location of a file or service on the Internet, e.g. `http://www.laas.fr/`

of web services, organised in yellow pages, white pages and green pages. A UDDI directory is a web service that accepts standard SOAP search requests and replies WSDL descriptions of web services that correspond to the request.

The use of UDDI has decreased a lot recently. Analysts explain that automated discovery and consumption of services is not the way companies do business, and UDDI as a way to interact with external companies has not met great success. It is said to be useful inside companies as a way to find mobile endpoints: if the endpoint of a service changes (the software has moved to another computer), the WSDL file of this service must be updated. This can be done by publishing the new WSDL file to a UDDI directory that lists all the mobile endpoints of the society. Service consumers get the new endpoint by consulting the updated UDDI directory.

7.3.5 WS-BPEL

The *Web Services Business Process Execution Language* is an Oasis standard [OASIS, 2007]. It provides a way do describe business processes based on web services. The basic operation is the invocation of a web service, and the aim of the business process is to provide web services. It relies on the languages SOAP, WSDL and also XSLT and XPath.

The language can characterize executable and abstract business processes. Executable business processes describe the workflow that must be carried on by the business partner during the transaction. It is an implementation, and can be executed by a BPEL engine to provide actual services. Abstract business processes are not complete, and allow to describe only some aspects of an executable business process. They can be used for specifying the behavior exposed to a choreography partner, or for specifying templates processes.

The language offers the standard control instructions: parallel branching and joins, conditional branching, loops, etc. Moreover, data in SOAP request and responses can be accessed and computed with XPath. The data can be reorganized from existing message structures (response of a web service) to new structures (request to another web service) with XSLT transforms.

7.3.6 Semantic web services and Ontologies

Automated composition of web services and choreography conformance checking are two very active research field in the web community. The consideration of semantic metadata can considerably help to solve such problems: when a service requires an integer number as input, knowing that this parameter is the number of ordered products allows one to provide much more focused algorithms. Languages have been defined for semantic description of web services, like WSDL-S [W3C, 2005a] that allows semantic annotations inside WSDL files.

Ontologies are a mean to organize, structure, compare and map semantic information. Semantic knowledge is expressed thanks to an ontology. The composition of services that use different ontologies for semantic annotation requires ontology manipulation. Several languages address this need, among which OWL-S [DAML,] is the most used, in particular in the research community.

7.4 Diagnosis requirements in Service Oriented Architectures

SOA permit a very high level of modularity and flexibility. Since services are defined by their interface, substituting a service with an equivalent one inside a business process is fairly easy. Moreover, a service is meant to be involved in several different business processes with different companies.

Abstraction and modularity are challenges for supervision, in the sense that nobody can access the information about the whole business process: the model is distributed among the business partners. Supervision of such systems requires cooperation between the different actors of the process. However, since the different companies do not publish the internal details of the workflow they execute, supervision modules, while cooperating, must not publish private information either. Cooperation must be set up by communicating only on public data. Moreover, since the same service may be involved into several business processes, its supervision module needs to cooperate with various combinations of partners. Finally, some business partners may provide supervision functions while other partners don't. Consequently, supervision systems in SOA must also be very modular, abstract and flexible.

Since the supervision system must offer the same modularity as the system, it is natural to cast it into a Service Oriented Architecture. For diagnosis, each service is associated to a diagnoser D_i, that is aware of its internal implementation. A given diagnoser D_i exposes its services to other diagnosers. Diagnosis is performed by a workflow, that can start either from inside the diagnoser D_i (if symptoms appear in the service supervised by D_i) or from another diagnoser that invokes diagnosis services on this diagnoser (if symptoms appear in a partner's service). According to the business process to supervise, the diagnoser can cooperate with the diagnosers of different partners, or with no diagnoser if the partner is not equipped with diagnosis capabilities.

In service oriented architectures, the internal implementation of services is not published to business partners. Consequently, the composition of services is based on functional knowledge (see section 1.3), since structural knowledge is private. A business process is a complex function that is realized by the composition (or aggregation) of several smaller functions that correspond to the inner services.

As a consequence, the models used for diagnosis in service oriented archi-

tecture are organized around the functional knowledge. Hence the description of faults, observations and communications is done at the functional level. Of course, internally, a local diagnoser can use structural knowledge (if any) to further refine the diagnosis at the structural level. However, the distributed diagnosis architecture that results form the constraints of service oriented architectures makes use of functional knowledge for communication between the diagnosers.

The algorithm described in chapter 5 is suited to be implemented into services oriented architectures for several reasons. First, its hierarchical structure reproduces the way services are composed. It can be adapted to orchestrated cases as well as choreographed cases, in this case the different partners need to agree in order to have a common global diagnoser to which they will publish a minimum of information, in particular for diagnosability analysis.

Chapter 8

Implementation and test case

The diagnosis algorithm described in chapter 5 has specifically been designed for web services. However, it is suited for all kinds of distributed contexts, and its modularity makes it a good candidate for service oriented architectures in general. This chapter describes the details of the implementation, and presents an illustrative test case.

As explained in the previous chapter, the diagnosis and diagnosability model is organized around functional knowledge. The model is supposedly derived from a workflow specification, for example in BPEL4WS language. In that case, we do not include any structural knowledge in the local models. Services are functions that compose activities, i.e. sub functions. The focus is put on the instance level, i.e. every execution of the workflow is monitored separately. Diagnosis aims at assessing which activity(ies) fail to deliver its (their) function.

8.1 Binary decision diagrams

The problem of checking the admissibility (see definition 5.9) of a partial assignment can be very complex if implemented incorrectly. In addition, exploring the space of the extensions of a partial assignment in order to find a complete set of admissible extensions is computationally expensive. We chose to use Binary Decision Diagrams to store the models and check admissibility. This section presents an overview of BDDs and how they can be used to represent a system model.

From now on, let us assume that all the variables used in the system model are Boolean, i.e. they range over the domain $\{0, 1\}$ and $\mathrm{Dom}(V) = \mathbb{B}^{|V|}$ (see definition 5.5 page 88). We associate each constraint C to the Boolean function

$\mathcal{B}$ such that:

- $\mathcal{B} : \mathbb{B}^{|V|} \longrightarrow \mathbb{B}$
- For any Boolean tuple $(v_1, v_2 \dots v_n) \in \mathbb{B}^{|V|}$, $\mathcal{B}\big((v_1, v_2 \dots v_n)\big) = 1$ if and only if the assignment $\gamma = \big(V, (v_1, v_2 \dots v_n)\big)$ is consistent with C.

Each constraint can consequently be represented by a Boolean expression. The ternary operator if-then-elsen noted *ite*, is a very powerful operator for canonical representation of Boolean expressions.

Definition 8.1 (If-then-else boolean operator) *The* if-then-else *operator ite is a ternary operator defined as:*

$$ite(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$$

x is the test variable, y the positive test clause, and z the negative test clause.

The Shannon expansion of a Boolean formula f on a variable x_i consists in whiting f under the following form:

$$f = ite(x_i, f \wedge x_i, f \wedge \neg x_i)$$

The formula $f \wedge x_i$ (resp. $f \wedge \neg x_i$) can be obtained by replacing every occurrence of x_i in f by 1 (resp. 0).

The *if-then-else normal form* is obtained by applying the Shannon expansion to f on all the variables. Given a total order on the variables used in a set of Boolean formulas, the if-then-else normal form applied in increasing variable order is a canonical (i.e. unique) representation of Boolean formulas. *Two formulas are equivalent if and only if their expression in if-then-else normal form is equal (given a total order on variables).*

The if-then-else normal form can be used to store compactly and manipulate efficiently a great number of Boolean formulas, by the means of Reduced Ordered Binary Decision Diagrams (ROBDDs) [Meinel and Theobald, 1998]:

Definition 8.2 (Reduced Ordered Binary Decision Diagram) *A* Binary Decision Diagram *(BDD) is a rooted, directed, acyclic graph, which consists of decision nodes and two terminal nodes called 0-terminal and 1-terminal.*

Each decision node is labeled by a Boolean variable and has two child nodes called low child and high child. The edge from a node to a low (high) child represents an assignment of the variable to 0 (1).

Such a BDD is called ordered *if the different variables appear in the same order on all paths from the root. A BDD is said to be* reduced *if it does not contain any isomorphic subgraphs.*

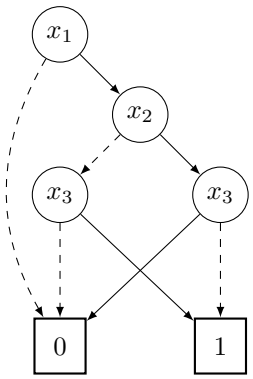


Figure 8.1: A Reduced Ordered Binary Decision Diagram (ROBDD) for the expression $ite(x_1, ite(x_2, ite(x_3, 0, 1), ite(x_3, 1, 0)), 0)$ that is equivalent to $x_1 \wedge (x_2 \neq x_3)$.

Efficient algorithms are given in [Meinel and Theobald, 1998] in order to compute operations on ROBDDs. This efficiency led us to use this data structure to represent the assignments and constraints manipulated during diagnosability analysis.

8.2 Software architecture

The diagnosis architecture has been extended from its specification in chapter 5. We initially considered several local diagnosers coordinated by a global diagnoser that plays the role of supervisor of all the local diagnosers. We now introduce a hierarchical architecture, in which a global diagnoser can also have a supervisor, all the diagnosers are organized into a tree with a root supervisor. All the leafs are local diagnosers, and all the other diagnosers are global diagnosers that supervise the diagnosers placed below them in the structural hierarchy.

We have implemented a prototype in Java, although the BDD library, called BuDDy is internally coded in C++. The main software components are detailed in this section.

8.2.1 Diagnosers

The software architecture is centered over the different types of diagnosers. Several interfaces and implementations are defined, and a class diagram is given in figure 8.2.

Diagnoser interface: this is the most abstract interface, it defines the methods that any kind of diagnoser should implement:

- *StartDiagnosis* is called by a diagnoser (or by the system) on its supervising diagnoser. It is used to launch a diagnosis process from bottom up. The requested diagnoser extends the inputs, and then calls its supervisor's *StartDiagnosis* method. The input to this method is a set of assignments, that represents the abnormal observation (when called by the system) or the hypotheses that explain the initial observation.
- *ExtendForDiagnosis* is called by the supervisor, or by the *StartDiagnosis* method. It receives some assignments as input, and computes a complete set of admissible extensions with respect to the model of the subsystem this diagnoser is in charge of. The implementations of this method completely differ in *local* and *global* diagnosers.
- *GetDiagnosis* allows one to get the partial diagnoses for the monitored subsystem at the end of the diagnosis process.

OrchestratedDiagnoser interface: this interface provides methods that need to be implemented by diagnosers that accept a supervisor. This methods provide a "supervisor endpoint" that can be used either by a classical *global diagnoser* supervisor, or by a *diagnosability analyzer* supervisor. The methods are:

- *ExtendForDiagnosability* is always called by the supervisor. Its only difference with the *ExtendForDiagnosis* method is that the results are restricted not only to interface variables, but also to mode and observable variables.
- *(Get/Set)Supervisor* gets or sets the supervisor of this diagnoser for the supervised instance of business process.
- *Get(Mode/Observable/Interface)Variables* are called by the supervisor in order to qualify the assignments that are returned by the *extend* operations. In particular, the knowledge of mode and observable variables is required for diagnosability analysis. Such knowledge is not needed by the supervisor for diagnosis, but for diagnosability analysis, it is necessary to know mode and observable variables.
- *GetDomain(Variable)* returns the domain of the specified variable of the subsystem this diagnoser is in charge of. This method is used to create the initial alternative rank 1 partial fault modes, in the case of non binary mode variables. We have assumed that all the variables are binary, which results from the use of ROBDDs. However, the software architecture is ready to release this assumption, which is illustrated by the existence of this method.

LocalDiagnoser interface: this interface provides methods that need to be implemented only by local diagnosers. It specifies methods that allow to manage observations: since global diagnosers are not in contact with the system, only local diagnosers need to manage observations.

- *AddObservation(Assignment)* merges the specified new observation with the previous observations if any, into a unique assignment to observable variables.

- *GetObservations* returns an assignment to observable variables containing the values observed in the monitored business process instance.

AbstractDiagnoser class: this abstract class provides a partial implementation of the *Diagnoser* and *OrchestratedDiagnoser* interfaces. It implements the supervisor and diagnoses management. The *Extend* methods are not implemented since they depend on the model.

AbstractLocalDiagnoser class: this abstract class adds the methods defined in the *LocalDiagnoser* interface to the *AbstractDiagnoser* class. It implements the observations management.

BDDLocalDiagnoser class: this class adds the implementation of all the model related methods to the *AbstractLocalDiagnoser* class. It relies on the *BDDLocalModel* class described in next section to store the model and perform admissibility checks.

GlobalDiagnoser class: this class adds the implementation of all the model related methods to the *AbstractDiagnoser* class. It relies on the structural description of its subsystem, and delegates the *Extend* operations to the relevant diagnosers of which this is the supervisor.

The **DiagnosabilityAnalyzer** class is not part of the diagnoser type hierarchy, since it is not a diagnoser itself. It is a component of a different type, that is placed above the topmost supervisor of the diagnoser structural hierarchy. This class first computes all pairs of alternative rank partial fault modes, and relies on the methods *getModeVariables*, *extendForDiagnosability*, and *getDomain* defined in the *OrchestratedDiagnoser* type. Later, in order to check the discriminability of a pair of partial fault modes, the method *getObservableVariables* is used to restrict the admissible extensions to observable variables. Figure 8.3 shows the activation sequences of the diagnosers during diagnosis and diagnosability analysis.

8.2.2 Assignments and constraints

The BDD library does not provide an easy way to encode and decode assignments and constraints from XML, text, or any other format used for communication between diagnosers, or between a diagnoser and a human user. Since diagnoser need to exchange sets of assignments, a java class *Assignment* is used to represent an assignment. It is based on a hash-based mapping between objects representing variables, and objects representing values. Such a structure is easy to convert into a textual format.

Constraints are only needed during the *Extend* operation. They are used to represent a local model, and to check the admissibility of partial assignments. These operations are internal to the local diagnosers, it is hence not necessary to translate the model under a communication format. Models are directly stored under the form of a BDD, however they are initially specified by the designer

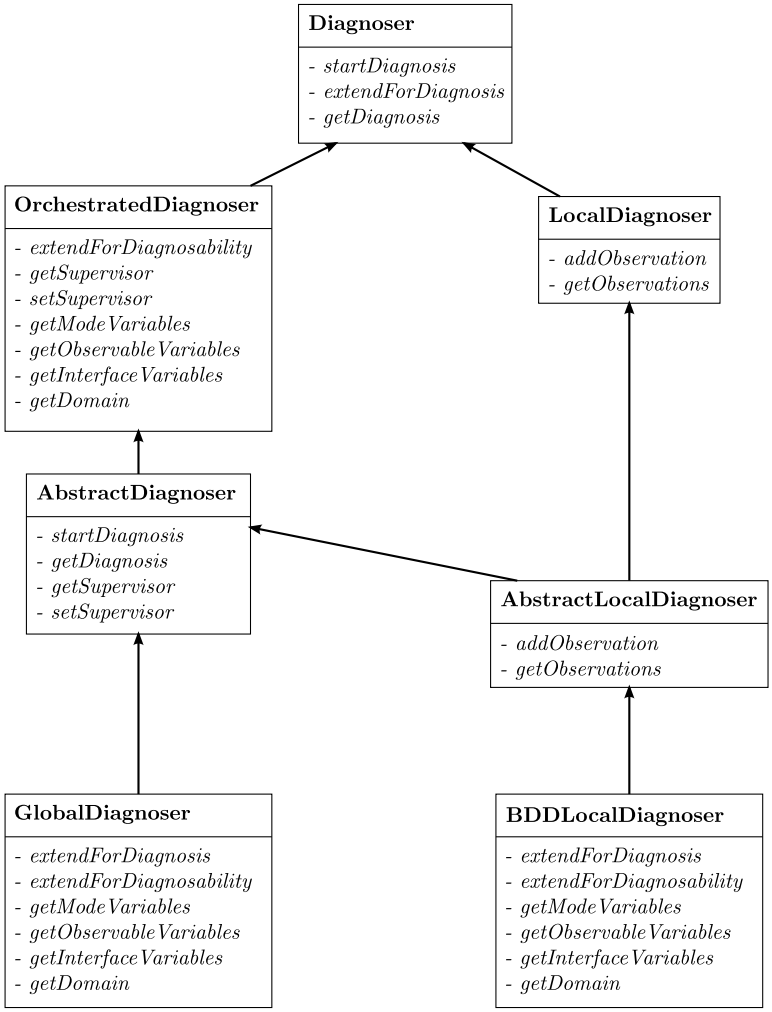


Figure 8.2: Class diagram for the different types of diagnosers.

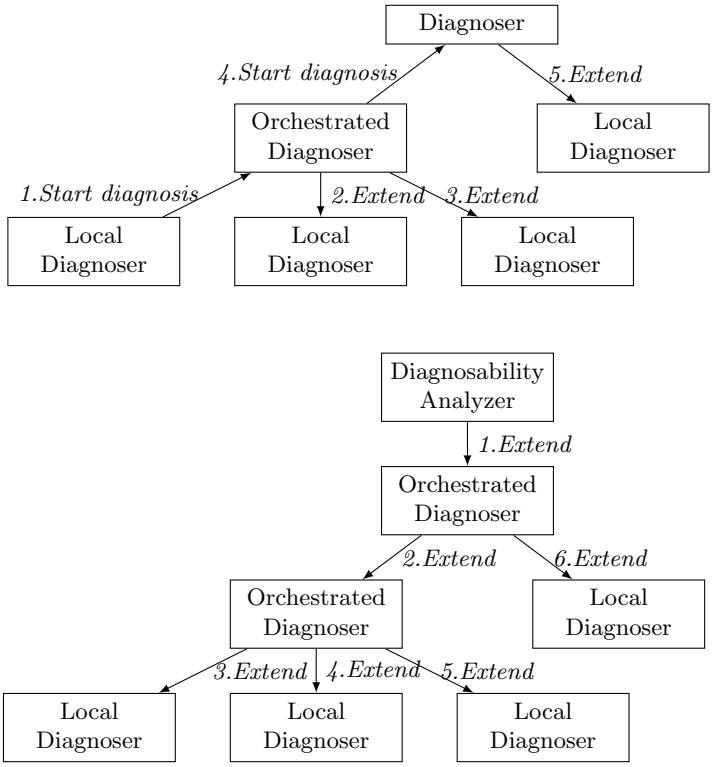


Figure 8.3: Diagnoser activation sequences for diagnosis (above) and diagnosability analysis (below). This picture gives an example of the order in which the local diagnosers are involved in the process. In general, some diagnosers may never be invoked and some may be invoked twice (as discussed in [Ardissono *et al.*, 2005]).

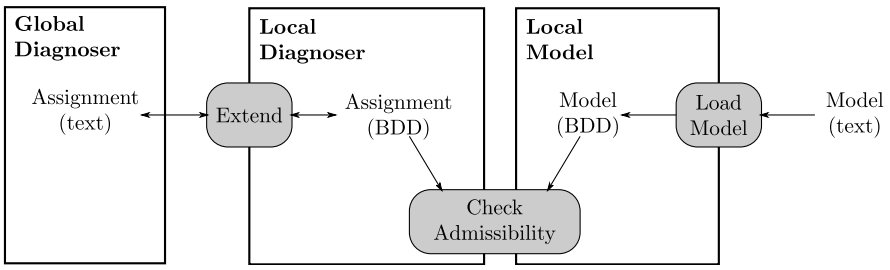


Figure 8.4: Assignment and constraint representations used in the program. Text representation relies on a hash-based mapping and is used for communication with the supervisor or with the global diagnoser. BDD representation is used for computing the complete set of admissible extensions.

in a text file. Figure 8.4 shows the different representations of assignments and constraints inside and outside the local diagnoser.

8.3 Implementation aspects

8.3.1 Test case

The test case consists in a food shop web service that provides a service of menu composition and online payment. The company has an online shop (that does not have a physical counterpart) and several warehouses (WH1,. . . , WHn) located in different areas that are responsible for stocking imperishable goods and physically delivering items to customers, depending on the area each customer lives in.

Customers (C1,. . . , Ck) interact with the FoodShop Company in order to place their orders, pay the bills and receive their goods. In case of perishable items, that cannot be stocked, or in case of out-of-stock items, the FoodShop Company must interact with several suppliers (SUP1,. . . , SUPm).

Although most of the interactions in this example are electronic, and take place between Web Services, in some cases there are physical actions and interactions that are performed by humans (e.g. the sending of a package). These too are modeled in the context of Web Services.

In each conversation the following actors take part: one customer, the online shop, one warehouse, and a variable number of suppliers, which could also be 0 (represented in gray). When a customer places an order, the shop first selects the warehouse that is closest to the customer's address, and that will thus take part in the conversation. Ordered items are split into two categories: perishable (cannot be stocked, so the warehouse cannot possibly have them in stock) and imperishable (the warehouse might have them).

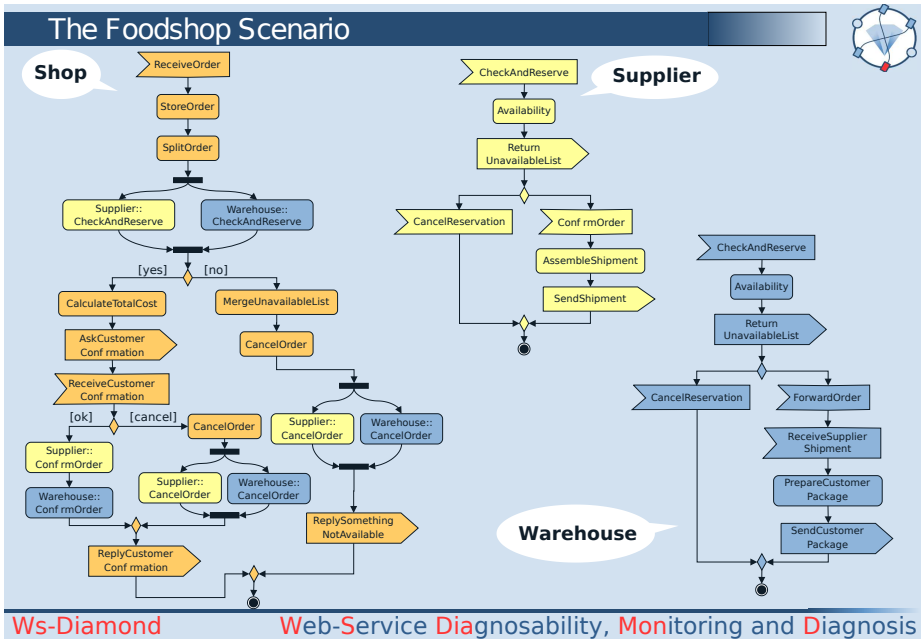


Figure 8.5: FoodShop workflow. The test case consists in 3 web services, a Shop, a Supplier and a Warehouse.

Perishable items are handled directly by the shop, while imperishable items are handled by the warehouse. The first step is to check whether the ordered items are available, either in the warehouse or from the suppliers (we have not considered items exchanges among different warehouses, in order not to make the example too complicated). If they are, they are temporarily reserved in order to avoid conflicts between several orders. Once the shop receives all the answers on availability, it can decide whether to give up with the order (again, in order to keep things simple, this happens whenever there is at least one unavailable item) or to proceed. In the former case, all item reservations are canceled and the conversation ends.

If the order goes on, the shop computes the total cost (items + shipping) with the aid of the warehouse, that provides the shipping costs. Then it sends the bill to the customer, that can decide whether to pay or not. If the customer does not pay, all item reservations are canceled and the conversation ends here. If the customer pays, then all item reservations are confirmed and all the suppliers (in case of perishable or out-of-stock items) are asked to send the goods to the warehouse. The warehouse will then assemble a package and send it to the customer

The initial example description above has been restricted to one warehouse and one supplier, and the workflow has evolved to the one depicted in figure 8.5.

8.3.2 Implementation

The implementation of the software architecture described above has the following advantages:

Feedback to the designer: as described in chapter 5, the diagnosability analysis approach provides the results in terms of pairs of *partial fault modes*. This is particularly convenient for the designer, since it allows one to reason about subparts of the system and consider few faults at a time, as illustrated in figure 8.7.

The played scenario illustrates the obtained discriminability results for a selection of partial fault modes, in particular for a pair in which one fault is masked by another fault.

Modularity: the two diagnoser implementations can be connected to a supervisor, possibly a DiagnosabilityAnalyzer. Consequently, it is possible to analyze the diagnosability of any subsystem being monitored by one diagnoser. In figure 8.3 the diagnosability analyzer is placed above the highest global diagnoser, but could have been placed above any diagnoser.

When a service is involved in several business processes, for each instance of the different workflows, an instance of the service diagnoser is created. According to the business process realized in the workflow, the diagnoser will have a different supervisor that will connect it to the diagnosers of the involved services.

Figures 8.6 and 8.7 show how partial fault modes can describe more briefly the diagnosability of a system, and how the prototype takes advantage of these in its result browsing interface. In this case, only three faults are considered, and the results in terms of partial fault modes are human readable. However, with more faults, even expressed with partial fault modes, the diagnosability results need a browsing tool, that allows the designer to focus on only a few faults at a time.

Fault mode 1			Fault mode 2			
StoreOrder_mode	SplitOrder_mode	Assemble_mode	StoreOrder_mode	SplitOrder_mode	Assemble_mode	Discriminable ?
ok	*ok*	*ok*	*ab*	*ok*	*ok*	yes
ok	*ok*	*ok*	*ok*	*ab*	*ok*	yes
ok	*ok*	*ok*	*ab*	*ab*	*ok*	yes
ok	*ok*	*ok*	*ok*	*ok*	*ab*	yes
ok	*ok*	*ok*	*ab*	*ok*	*ab*	yes
ok	*ok*	*ok*	*ok*	*ab*	*ab*	yes
ok	*ok*	*ok*	*ab*	*ab*	*ab*	yes
ab	*ok*	*ok*	*ok*	*ab*	*ok*	yes
ab	*ok*	*ok*	*ab*	*ab*	*ok*	no
ab	*ok*	*ok*	*ok*	*ok*	*ab*	yes
ab	*ok*	*ok*	*ab*	*ok*	*ab*	yes
ab	*ok*	*ok*	*ok*	*ab*	*ab*	yes
ab	*ok*	*ok*	*ab*	*ab*	*ab*	no
ok	*ab*	*ok*	*ab*	*ab*	*ok*	yes
ok	*ab*	*ok*	*ok*	*ok*	*ab*	yes
ok	*ab*	*ok*	*ab*	*ok*	*ab*	yes
ok	*ab*	*ok*	*ok*	*ab*	*ab*	no
ok	*ab*	*ok*	*ab*	*ab*	*ab*	yes
ab	*ab*	*ok*	*ok*	*ok*	*ab*	yes
ab	*ab*	*ok*	*ab*	*ok*	*ab*	no
ab	*ab*	*ok*	*ok*	*ab*	*ab*	yes
ab	*ab*	*ok*	*ab*	*ab*	*ab*	no
ok	*ok*	*ab*	*ab*	*ok*	*ab*	yes
ok	*ok*	*ab*	*ok*	*ab*	*ab*	no
ok	*ok*	*ab*	*ab*	*ab*	*ab*	yes
ab	*ok*	*ab*	*ok*	*ab*	*ab*	yes
ab	*ok*	*ab*	*ab*	*ab*	*ab*	no
ok	*ab*	*ab*	*ab*	*ab*	*ab*	yes

*pfm*1			*pfm*2			
StoreOrder_mode	SplitOrder_mode	Assemble_mode	StoreOrder_mode	SplitOrder_mode	Assemble_mode	Discriminable ?
ok			*ab*			yes
	ok	*ok*		*ok*	*ab*	yes
	ok	*ok*		*ab*	*ok*	yes
	ok	*ok*		*ab*	*ab*	yes
	ok	*ab*		*ab*	*ok*	yes
ab	*ok*		*ab*	*ab*		no
ok	*ab*	*ok*	*ok*	*ab*	*ab*	no
ab	*ab*	*ok*	*ab*	*ab*	*ab*	no
ok	*ok*	*ab*	*ok*	*ab*	*ab*	no

Figure 8.6: Diagnosability of the system, expressed in terms of pairs of fault modes (left side), and pairs of partial fault modes (right side).

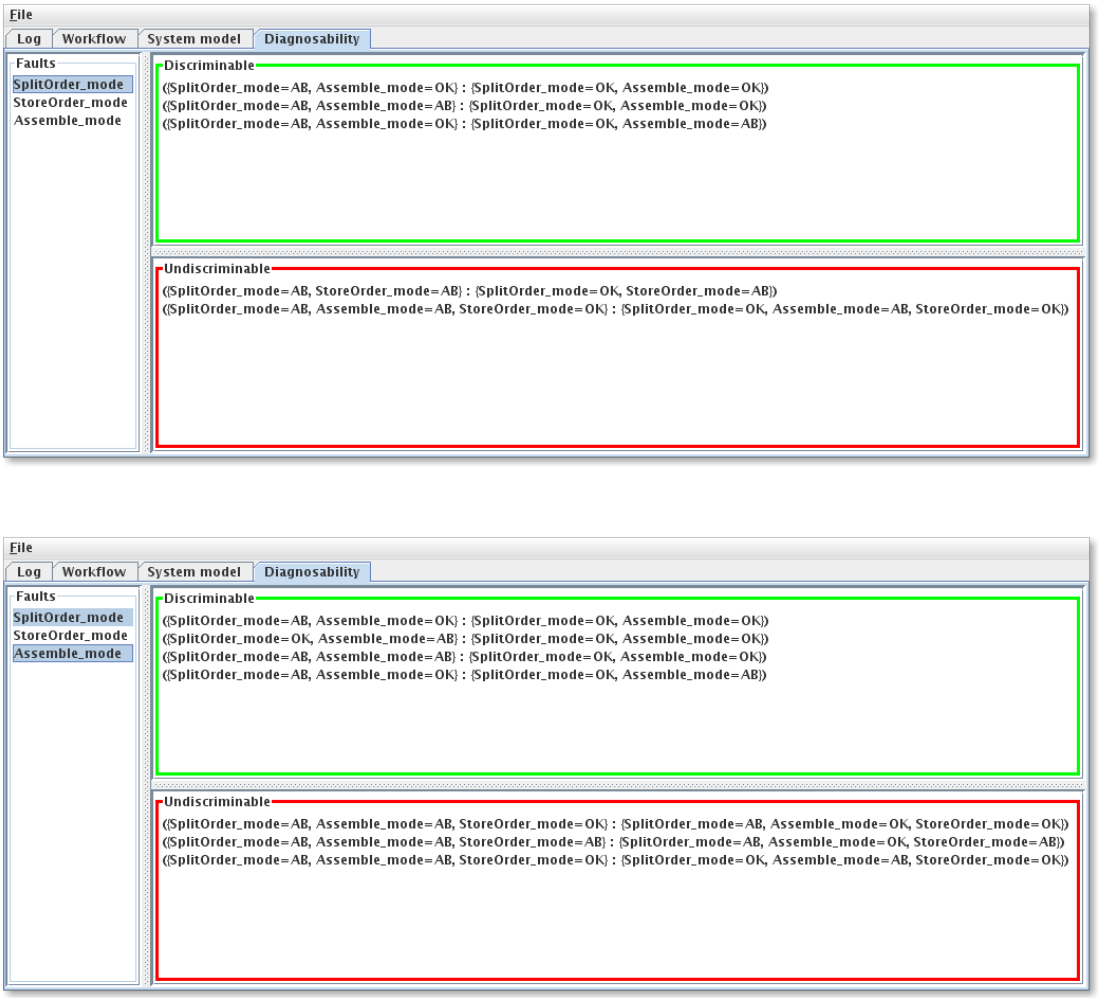


Figure 8.7: Diagnoser feedback screenshot from the prototype. The interface for browsing the diagnosability results is based on the principle of partial fault modes. The designer can select one (or several) faults on the left column, which defines a scope for partial fault modes. The right side displays discriminability information for pairs of alternative partial fault modes that have or extend the selected scope.
Above, the analysis of the fault in the *SplitOrder* activity (highlighted on the left side), below the analysis of the combinations of the faults *SplitOrder* and *Assemble* (also highlighted). In the discriminable pair *(SplitOrder_mode = ok* $\wedge$ *Assemble_mode = ok, SplitOrder_mode = ok* $\wedge$ *Assemble_mode = ab)* the fault *SplitOrder* is present on both sides. This pair is hence not relevant when considering only the fault *SplitOrder* and is visible in the first screen shot above. On the contrary, it brings information for the diagnosability of the fault *Assemble* and is visible in the second screen shot.

Conclusion and perspectives

In this thesis, a unified point of view on diagnosability has been developed, based on a comparison of the different approaches that correspond to the different modeling formalisms used in this research domain. The notions of set of observables and fault signature have been extended to event based approaches, which has allowed us to express diagnosability with a unified formal definition. This definition carries the idea that two different combinations of faults must not be able to provide a common observation for the system to be diagnosable. This unification of diagnosability and the underlying concepts opens many investigation paths. Contributions that are useful in the framework of a specific modeling formalism can be adapted to other formalisms. In particular the concepts of partial fault modes and diagnosable blocks, defined for state based approaches can be adapted to event-based approaches and are likely to provide equivalent generalisations with respect to the classical diagnosability approaches.

The problem of diagnosis based on multiple models with different formalisms was addressed in [Ressencourt, 2008], and illustrates the need for such applications. The unification of diagnosability definitions is an important step towards diagnosability analysis using different models. The hardest part of model-based diagnosability analysis has often been depicted as the modeling stage [Hamscher *et al.*, 1992]. The possibility to use different models is very promising for facilitating the modeling stage. Two different models can be used to model two different system components, or to model two different aspects of the same component. There are many constraints that influence the choice of a modeling formalism, and this work provides more freedom in the choice of a model and a unified framework for diagnosability analysis. This is particularly true for hybrid systems that couple state based and event based models. These should directly benefit from unified diagnosability.

The unified diagnosability definition is independent from the underlying model, and focuses on the relation between faults and observables. This high level point of view allowed us to adapt efficiently a particular constraint based diagnosis algorithm for diagnosability analysis, by introducing the concept of partial fault mode. This approach allows us to reason on subparts of the system, and to decrease significantly the number of signature comparisons. It is at our knowledge a completely new approach, with no equivalent in the state based or the event based worlds. The concept of partial fault mode and its properties

have been applied to a distributed state based approach, showing the duality with constraint based diagnosis [Console *et al.*, 2007] that reasons exclusively on partial assignments. We claim that it is possible to adapt it to event based approaches by adopting the formalism of ω-languages as described in chapter 4. We applied this diagnosability approach in the context of service oriented architectures, and implemented it. This implementation provides a high modularity in response to the constraints of service oriented architectures, and provides an original interface for the designer to browse the results of diagnosability, that benefits from the partial fault modes point of view.

The application of model-based diagnosis and diagnosability analysis to software systems is still a young applicative domain. This field opens many connections with the software safety engineering domain, in which diagnosis is most of the time ignored at the benefit of symptom based reactions that may result quite inappropriate. With the growing need for complex software systems, such applications are likely to receive interest from the research and the industry worlds. The problem of diagnosis in the environment of communication networks has received interest in [Pencolé and Cordier, 2005], at the level of hardware and communication protocols. In this thesis the applicative level of communication networks has been considered. Both applications are complementary and integrating them in a general communicating systems diagnosis framework is an interesting perspective. Adaptive network communications, ad-hoc networks and sensor networks are also interesting application contexts for model-based diagnosis.

Diagnosability analysis takes place during the design phase, and is meant to provide feedback to the system designer for one to optimize the system or its model for diagnosis. The problem of integrating diagnosability into a general design support tool was also addressed in this thesis. In the context of state based systems, the unified definition was generalized in order to account for any set of states. Sets of states can characterize properties like faults, which falls back to standard approaches, but can characterize many other kinds of properties, like repair preconditions, levels of quality of service, etc. This contribution is limited to the characterization of diagnosability, and an algorithmic approach is still to be developed. The adaptation to event based approaches, using the ω-language formalism, should lead to interesting comparison with the supervision patterns approach [Jéron *et al.*, 2006] and the linear temporal logic approach [Jiang and Kumar, 2002], since all of them permit to analyze not only faults, but also any property that can be expressed with a given language. This work provides grounds to apply the techniques of diagnosability analysis to the analysis of many kinds of properties. This is a step towards a validation framework that ensures a system is suited for supervision at design time. The final purpose is to provide means to design more reliable systems more easily.

Quatrième partie

Un point de vue unifié sur la diagnosticabilité

Introduction

Le développement des technologies de l'information et de ses applications dans l'industrie et les services a introduit des systèmes automatisés complexes partout dans notre vie. Le nombre de systèmes dont les utilisateurs ne connaissent pas le fonctionnement interne est en augmentation, sinon en explosion. Jusque dans nos poches, les téléphones portables sont un très bon exemple, sans parler des voitures, avions, de l'industrie agro-alimentaire, l'industrie des cosmétiques, des marchés financiers, etc. Le fait est que dans les sociétés modernes, les gens se fient à des systèmes qu'ils ne comprennent pas totalement.

Alors que ces systèmes abondent autour de nous, leurs défaillances deviennent de plus en plus difficiles à prévoir, comprendre et à réparer. Le besoin d'outils d'assistance à la surveillance de ces systèmes a entraîné des efforts croissants durant ces vingt à trente dernières années. Le problème du diagnostic de fautes a été adressé de maintes fois, et est aujourd'hui un domaine de recherche mature. Les techniques utilisées pour le diagnostic ont évolué d'approches ad hoc vers des approches à base de modèles, considérées plus pratiques et plus adaptatives. Un problème important identifié au cours des nombreuses applications du diagnostic à base de modèles dans l'industrie est que dans la plupart des cas, le problème du diagnostic n'est considéré qu'une fois le système conçu, ce qui rend impossible l'ajout de capteurs, ou même la modification du système afin de fournir de meilleures informations à l'outil de diagnostic.

Aujourd'hui, la capacité d'un système à être diagnostiqué est devenue un argument commercial, et fait partie des spécifications de tous types de systèmes. Il est important pour le concepteur et constructeur d'un système de s'assurer que celui-ci est diagnosticable, c'est-à-dire que les fautes qui peuvent l'altérer sont identifiables relativement facilement. Cet aspect est une clé pour fournir des systèmes plus fiables, dans lesquels les coûts de maintenance sont plus prévisibles. Cette propriété du système, appelée diagnosticabilité, doit être étudiée lors de la conception du système. Elle est vouée à être un facteur à prendre en compte lors de la conception, au même titre que les coûts de fabrication, ou d'autres qualités du produit. Il est alors nécessaire de disposer d'outils permettant d'analyser la diagnosticabilité d'un système et de donner des retours aux concepteurs.

Le problème d'analyse de la diagnosticabilité est encore un domaine de recherche récent. Plusieurs communautés ont développé des approches pour le

diagnostic à base de modèles différentes et séparées, et les solutions pour l'analyse de la diagnosticabilité qui en découlent ont hérité de cette hétérogénéité. Bien que les formalismes utilisés dans ces approches soient très différents, le principe de raisonnement est très similaire. Des travaux ont été menés afin d'unifier ces approches. Cette thèse contribue à ce travail d'unification et propose un point de vue général sur la diagnosticabilité qui prend en compte les approches existantes du diagnostic et de la diagnosticabilité. Cette thèse s'intéresse aussi à l'intégration du diagnostic dans un outil plus général d'assistance à la supervision.

Ce document est constitué de trois parties. En premier lieu, une vue d'ensemble sur le diagnostic et l'analyse de diagnosticabilité est fournie. Dans l'exercice du diagnostic à base de modèle, construire le modèle du système est un aspect difficile et très important. Un état de l'art des formalismes et algorithmes existants révèle une diversité significative dans les formalismes et les approches. Une recherche originale des hypothèses implicites et des différences entre chaque approche et une position argumentée pour des définitions unifiées sont consignées.

La seconde partie contient le coeur de la contribution de cette thèse. Le concept de signature de fautes, importé des approches à base d'états, est choisi comme un concept unificateur pour définir la diagnosticabilité. Il est étendu aux approches à base d'évènements de telle manière qu'une définition unique de la diagnosticabilité tient pour l'ensemble les approches considérées. Ce concept de signature de faute est étendu afin de fournir une approche efficace d'analyse de la diagnosticabilité. Il est encore étendu à des propriétés plus générales comme les préconditions de réparation ou la qualité de service afin d'intégrer plus facilement la diagnosticabilité dans un module de supervision général.

La dernière partie décrit une application aux services web. La technologie des services web est basée sur des langages apparus récemment. Ces langages dérivés de XML (eXtensible Markup Language, langage à balises extensible) sont très modulaires et fortement structurés, et ont bénéficié d'un important intérêt de l'industrie depuis leur apparition peu avant l'an 2000. Leur capacité à implémenter des programmes distribués qui fournissent des services, et à composer ces services dans des services plus complexes, à l'intérieur comme entre plusieurs compagnies, a entraîné un besoin particulier pour la fiabilité de ces systèmes. Les grandes modularité et flexibilité des systèmes créés à partir de ces technologies rend les approches à base de modèles tout à fait adaptées pour leur supervision. L'approche d'analyse de diagnosticabilité proposée pour analyser de tels systèmes réutilise un algorithme de diagnostic existant. Cet algorithme utilise un modèle à base de réseaux de contraintes et repose sur la propagation de contraintes. En modifiant le contexte d'exécution de l'algorithme, un algorithme original d'analyse de diagnosticabilité est obtenu. Cet algorithme a plusieurs avantages, en particulier une architecture modulaire, une économie des communications, ainsi qu'une représentation compacte des résultats pour le concepteur.

Les résultats obtenus dans cette thèse fournissent une base saine et claire pour un essor de l'analyse de la diagnosticabilité. Les définitions unifiées permettent une meilleure communication entre les communautés qui utilisent des

formalismes différents. Les extensions permettent une meilleure intégration du diagnostic et de la diagnosticabilité avec d'autres tâches de supervision. Enfin, cette thèse propose un point de vue de haut niveau, peu technique sur la diagnosticabilité, ce qui peut la rendre plus facile à introduire dans l'industrie.

Chapitre 9

Modèles et approches de la diagnosticabilité

Quiconque conçoit un système automatisé complexe est confronté au problème du diagnostic des fautes. Les solutions à ce problème ont évolué d'approches associatives à des approches à base de modèles. Le problème du diagnostic à base de modèle a été adressé séparément par plusieurs communautés. La communauté d'automatique a développé des technique de Détection et Isolation de Fautes (FDI) pour les systèmes continus et à base d'évènements discrets. La communauté d'Intelligence Artificielle (DX) a développé deux approches, basées respectivement sur la logique et des formalismes à évènements discrets. Ces approches ont été développées dans des courants distincts et parallèles, et leur comparaison révèle de grandes similarités dans le raisonnement général, mais de grandes diversités dans les détails de la modélisation. Lorsque ces communautés s'attelèrent au problème de l'analyse de la diagnosticabilité, la même diversité fut reproduite.

Les industries qui souhaitent ajouter des capacités de diagnostic à leurs systèmes font souvent face à un problème particulier, qui résulte du fait que les concepteurs du systèmes ne sont généralement pas formés en diagnostic. Le diagnostic n'est pas pris en compte au moment de la conception, mais plus tard dans le processus de développement du système. Le problème de cette manière de fonctionner est que le diagnostic ne peut pas influencer la conception du système, par exemple pour mener à l'insertion d'un capteur dans un bloc solide du système. Ceci peut être corrigé en fournissant un retour aux concepteurs à propos des capacités en diagnostic du système et de son instrumentation.

La diagnosticabilité est une propriété d'un système consistant à exhiber différents symptômes pour un ensemble prédéfini de fautes anticipées. Elle est souvent définie comme booléenne, oui/non, mais elle est beaucoup plus complexe. Dans la plupart des cas, lorsque plusieurs fautes sont combinées, les observations sont tellement perturbées qu'elles ne permettent pas de distinguer

une nouvelle faute qui pourrait s'ajouter à celles déjà présentes. Ceci rend le système non diagnosticable. Toutefois, plus on considère de fautes, moins il est probable qu'elles soient toutes présentes en même temps dans le système, et certaines approches de diagnostic se limitent aux fautes uniques ou doubles, et ne considèrent pas les fautes plus complexes. D'autres approches de diagnostic se contentent de fournir les diagnostics minimaux, c'est-à-dire que si une faute unique et une double faute double incluant cette faute unique peuvent expliquer les observations, l'algorithme choisit la faute unique, en général plus probable. Des travaux tentent de définir la diagnosticabilité comme un degré, entier ou réel, mais c'est, au même titre que la représentation booléenne, insuffisant pour exprimer la diagnosticabilité dans sa complexité.

Si un système n'est pas assez diagnosticable, cela peut provenir de deux causes :

- Le système ne contient pas assez de capteurs, ou ceux-ci sont mal placés. Les observations fournies par les capteurs ne permettent pas de discriminer deux situations de faute différentes. La solution est d'ajouter des capteurs dans le système, ce qui n'est pas toujours possible.
- Le système produit suffisamment d'informations, mais le modèle n'est pas assez précis, ou ne contient pas les informations pertinentes pour le diagnostic. Dans ce cas, il n'est pas nécessaire de modifier le système, mais uniquement le modèle utilisé pour le diagnostic.

Il semble naturel d'utiliser le même modèle pour le diagnostic et la diagnosticabilité, puisque les résultats de l'analyse de diagnosticabilité ne seraient pas pertinents par rapport à une approche de diagnostic qui n'utiliserait pas le même modèle. Modéliser un système pour le diagnostic et l'analyse de diagnosticabilité est par conséquent la même tâche.

Chapitre 10

Définitions unifiées

Les approches d'analyse de la diagnosticabilité décrites dans le chapitre 2 varient beaucoup de l'une à l'autre. Toutefois, toutes les approches font référence à des concepts communs qui peuvent servir de base à une analyse de diagnosticabilité globale, indépendante du modèle. Ce chapitre résume les hypothèses et points de vue adoptés dans chaque approche, et décrit la position qui est adoptée dans cette thèse. Les définitions qui en résultent tiennent compte de toutes les approches de la diagnosticabilité, et fournissent un point de vue unifié.

10.1 Fautes et modes de faute

Le concept fondamental dans le diagnostic et la diagnosticabilité, partagé par toutes les approches, est le concept de faute.

Définition 10.1 (ISO Fault) *Dans le document ISO/CD 10303-226 de l'*Organisation Internationale de Standardisation*, une faute est définie comme une condition ou un défaut anormal au niveau du composant, équipement ou sous-système, qui peut mener à un échec.*

Dans la définition 10.1, un échec se produit lorsque le système ne remplit pas le but pour lequel il a été conçu et construit. Cette définition informelle a été interprétée de différentes façons selon les approches de modélisation.

Nous présentons maintenant les différentes interprétations dans les approches de diagnostic à base de modèle du concept de faute. La position adoptée dans cette thèse est présentée et argumentée.

10.1.1 Les différentes interprétations d'une faute

Le concept de faute porte le même nom dans la plupart des approches de diagnostic, bien qu'une comparaison mette à jour qu'il est sujet à des différences d'interprétation significatives.

Dans les approches FDI, l'hypothèse de la faute unique est courante. Dans ce cas, on suppose que deux composants dans le système ne peuvent pas être dans un état anormal en même temps. Chaque faute est une cause différente qui peut mener à un échec du système à remplir sa fonction. Toutefois, cette hypothèse est fréquemment relâchée, et le système peut être sujet à des « fautes multiples ». La relaxation de cette hypothèse rend l'approche de diagnostic plus réaliste dans certains domaines d'application, mais complexifie le raisonnement de diagnostic et rend la diagnosticabilité beaucoup plus difficile à atteindre, surtout si les capteurs peuvent subir des fautes et envoyer des informations erronées au superviseur. En compromis, l'ensemble des fautes considérées peut être limité aux fautes doubles, ou triples, etc. Le comportement du système est influencé par la faute courante.

Dans les approches DX, le comportement de chaque composant dépend de la présence ou absence de fautes. La présence de ces fautes est caractérisée par une ou plusieurs variables discrètes appelées *variables de mode*, chaque combinaison de faute correspond à une valeur du tuple des variables de mode. Un aspect important est que les variables de mode peuvent être binaires (absence/présence de la faute) mais peuvent adopter un domaine fini plus grand si nécessaire.

Dans les approches à base d'évènements, les fautes sont des évènements, et peuvent avoir eu lieu ou non. Comme dans les approches classiques, les fautes sont permanentes, il n'y a pas de distinction entre les trajectoires qui contiennent une ou plusieurs occurrences du même évènement de faute. Le comportement du système résulte de l'ensemble des fautes qui sont arrivées.

Exemple 10.1 *Considérons un système hydraulique dans lequel une vanne peut rester bloquée en position ouverte ou fermée, et un tuyau peut se boucher.*

- Représentation FDI : *il y a **5** fautes différentes : 3 fautes uniques (tuyau bouché, vanne bloquée ouverte, vanne bloquée fermée) et 2 fautes multiples : (tuyau bouché & vanne bloquée ouverte, tuyau bouché & vanne bloquée fermée). Le mode normal et chaque faute sont associés à un ensemble de contraintes qui définissent le comportement du système dans le mode correspondant.*
- Représentation DX : ***2** variables de mode sont associées aux deux composants qui peuvent subir une faute. Le statut de la vanne est défini par la variable :*

  $$vanne \in \{normal, bloquée\ ouverte, bloquée\ fermée\}$$

  Le statut du tuyau est défini par la variable :

  $$tuyau \in \{normal, bouché\}$$

- Représentation SED : ***3** évènements de faute sont définis : « le tuyau se bouche », « la vanne se bloque ouverte », et « la vanne se bloque fermée ».*

Cet exemple illustre la diversité des interprétations de la notion de faute dans les approches. Selon l'approche, 5 fautes, 2 variables ou 3 énènements de faute sont utilisés pour représenter les différents comportements qui peuvent être adoptés par le système.

10.1.2 Dénomination unifiée

Avoir une vue unifiée sur les différentes approches pour la diagnosticabilité nécessite la définition de concepts qui nous permettent d'exprimer les différentes interprétations d'une faute. Cette section propose une définition pour *faute* et *mode de faute* qui sera utilisée dans cette thèse, et les positionne par rapport aux interprétations décrites dans la section précédente. Ces définitions visent à être aussi intuitives que possible, et résultent d'échanges avec les membres des différentes communautés de diagnostic à base de modèle, ainsi que d'autres domaines de recherche.

Définition 10.2 (Faute) *Une* faute *est un évènement élémentaire inattendu qui survient dans un système ou un composant, et qui peut altérer son comportement. Un système ou composant dans lequel une faute est apparue est déclaré fautif, et la faute est dite présente dans le système ou composant. Plusieurs fautes différentes peuvent arriver dans divers ordres sur le même système ou composant.*

Cette définition reprend la vision des approches à base d'évènements d'une faute, dans le sens que la faute est un évènement, et qu'elle est élémentaire. L'interprétation DX correspond aussi à cette définition, puisque le diagnostic consiste à définir quels composants sont fautifs. L'interprétation FDI n'est pas cohérente avec cette définition, puisqu'en FDI, deux fautes ne peuvent pas être présentes dans le système au même moment : le système ne peut pas être au même instant dans une faute unique, et une faute multiple. L'interprétation FDI correspond en fait à ce que nous définissons ici comme un mode de faute.

Définition 10.3 (Mode de faute) *Le mode de comportement du système qui résulte de l'occurrence d'une combinaison donnée de fautes est un* mode de faute*. Un mode de faute correspond à la présence de certaines fautes, et à l'absence des autres fautes dans le système. Le comportement normal du système correspond à l'absence de toutes les fautes, c'est un mode de faute appelé mode normal du système. L'apparition d'une faute modifie le mode de faute courant du système.*

Un mode de faute peut représenter l'absence de fautes (comportement normal), une faute unique, ou une faute multiple. Dans les approches DX, un mode de faute est défini en affectant une valeur à toutes les variables de mode du système. Dans les approches à base d'évènements, le mode de faute résulte de l'ensemble des évènements de faute qui sont apparus dans la trajectoire du système.

10.1.3 Modèles pour les fautes et modes de faute

Une définition unifiée des fautes et modes de faute n'est pas utile si elle ne peut pas être traduite dans les différents langages formels utilisés dans les approches de diagnostic. Dans cette section, des représentations formelles des fautes et modes de faute sont proposées, et leurs liens avec les différents formalismes sont clarifiés.

Une manière directe de modéliser les fautes et les modes de faute est de représenter les modes de faute comme des ensembles de fautes. Lorsque les fautes sont indépendantes, n'importe quelle combinaison de faute est possible dans le système, et l'ensemble des modes de faute est l'ensemble des parties de l'ensemble des fautes. Toutefois, comme il est illustré dans l'exemple 10.1, certaines fautes ne peuvent pas coexister dans le système. Dans le cas général, l'ensemble des modes de faute est inclus dans l'ensemble des parties de l'ensemble des fautes.

Définition 10.4 (Mode de faute : représentation par ensembles) *L'ensemble de toutes les fautes qui peuvent apparaître dans le système est noté F_{sys}. Un mode de faute f est représenté par l'ensemble des fautes qui sont présentes lorsque le système est dans le mode f. L'ensemble de tous les modes de fautes atteignables est noté $\mathcal{F}_{sys}$.*

$$\mathcal{F}_{sys} \subseteq 2^{F_{sys}}$$

Le mode normal est un mode de faute qui ne contient aucune faute, noté $\emptyset$.

Cette définition est facilement traduite dans les formalismes à base d'évènements, puisque les fautes sont représentées par des évènements. Le mode de faute est représenté par l'ensemble des évènements de faute qui appartiennent à la trajectoire du système. L'approche FDI est également compatible avec cette interprétation, puisqu'un ou plusieurs ensembles de contraintes sont associés à chaque mode de faute, indépendamment du formalisme utilisé pour les représenter.

Définition 10.5 (Mode de faute : représentation par variables) *Le système est associé à un ensemble de n* variables de mode *discrètes notées m_i qui couvrent des domaines finis D_i et caractérisent la présence ou l'absence de fautes. Un mode de faute f est défini par l'affectation de toutes les variables de mode à une valeur. L'ensemble de tous les modes de fautes atteignables $\mathcal{F}_{sys}$ est défini par :*

$$\mathcal{F}_{sys} = \left\{(v_1, v_2, \ldots v_n), \forall i \in \{1 \ldots n\}, v_i \in D_i\right\}$$

En relâchant l'hypothèse que les fautes sont binaires et indépendantes, la correspondance entre la représentation par ensembles et la représentation par variables utilisée dans les approches DX est sensiblement plus difficile et arbitraire. Par exemple, une variable de mode avec 3 valeurs possibles, comme $\{normal, faute_1, faute_2\}$, correspond dans la représentation par ensembles à deux

fautes qui s'excluent mutuellement. Cette exclusion peut provenir du fait que les deux fautes représentent deux altérations différentes du même composant. Dans ce cas les fautes font évoluer la variable de mode de la valeur *normal* à $faute_1$ et de *normal* à $faute_2$. Cette exclusion peut aussi provenir du fait que les fautes représentent deux degrés d'altération du même composant. Dans ce cas, elles changent la valeur de la variable de mode de *normal* à $faute_1$ et de $faute_1$ à $faute_2$.

Si une variable de mode à trois valeurs peut représenter deux types de fautes différents, une variable à 4 valeurs peut représenter encore plus de types de fautes :

- 1 type d'altération avec 3 niveaux d'altération ;
- 2 fautes indépendantes ;
- 2 types d'altération, l'un d'entre eux ayant 2 niveaux d'altération ;
- 3 types exclusifs d'altération.

Tous ces systèmes différents peuvent être représentés de la même manière avec des variables de mode, mais ont des représentations par ensembles différentes.

Comme nous l'avons illustré ci-dessus, il est impossible d'établir une correspondance à priori entre les variables de mode et les évènements de faute. Toutefois, en considérant toutes les fautes à la fois, la correspondance est faisable : deux combinaisons différentes de fautes sont nécessairement représentées par deux valeurs différentes du tuple de variables de mode. Inversement, deux valeurs différentes du tuple de variables de mode représentent nécessairement deux situations de fautes différentes dans le système.

10.2 Diagnosticabilité

Les définitions présentées dans le chapitre 2 et résumées dans le chapitre 10 sont maintenant révisées. Les faiblesses identifiées dans ces approches sont corrigées grâce aux définitions de faute et mode de faute de la section précédente.

Définition 10.6 (Signature de faute) *La signature de faute est une fonction* $Sig : \mathcal{F} \rightarrow OBS$. *À chaque mode de faute* f_i, *elle associe les observations qui peuvent être obtenues lorsque le système est dans* f_i.

Cette définition est reprise de la définition de [Cordier *et al.*, 2004, Travé-Massuyès *et al.*, 2006b], la seule différence est qu'elle est exprimée sur des modes de faute.

Le diagnostic fournit un ensemble d'explications, ou de candidats pour l'observation courante. Chaque candidat au diagnostic définit la présence de certaines fautes et l'absence d'autres fautes ; c'est un mode de faute.

Définition 10.7 (Candidat au diagnostic, Diagnostic) *Un mode de faute* f_i *est un* candidat au diagnostic *pour l'observation obs si et seulement si il peut expliquer cette observation :*

$$obs \in Sig(f_i)$$

Un diagnostic *pour obs est l'ensemble des candidats au diagnostic pour obs.*

Définition 10.8 (Discriminability, Detectability) *Deux modes de faute f_i, f_j sont* discriminables *si et seulement si :*

$$Sig(f_i) \cap Sig(f_j) = \emptyset$$

Un mode de faute f_i est détectable *si et seulement si il est discriminable du mode normal :*

$$Sig(f_i) \cap Sig(\emptyset) = \emptyset$$

La propriété de détectabilité n'est pas pertinente pour le mode normal. Selon la définition précédente, le mode normal n'est pas détectable puisqu'il n'est pas discriminable de lui-même. La définition peut être adaptée pour exclure le mode normal, bien que nous n'en voyons pas la nécessité.

Définition 10.9 (Diagnosticabilité) *Un mode de faute est* diagnosticable *si et seulement si il est discriminable de tous les autres modes de faute.*

Un système est diagnosticable si et seulement si toutes les paires de modes de faute sont discriminables :

$$\big(\forall f_i, f_j \in \mathcal{F}_{sys}, f_i \neq f_j\big), Sig(f_i) \cap Sig(f_j) = \emptyset$$

Ces définitions de discriminabilité et diagnosticabilité sont candidates aux définitions unifiées, puisqu'elles s'appuient sur les modes de fautes, et peuvent donc être traduites dans toutes les approches de modélisation décrites dans le chapitre 2. Toutefois, à ce stade, l'ensemble d'observables *OBS* n'a pas été défini pour les approches à base d'évènements. Cela implique que les définitions de signature, et toutes les définitions ci-dessus, ne tiennent pas.

La définition de l'ensemble des observables *OBS* pour les approches à base d'évènements n'est pas triviale. C'est le sujet du chapitre 4, et elle requiert l'utilisation de formalismes tels que les automates de Büchi et les ω-langages. C'est toutefois possible, et il est prouvé que les définitions données précédemment fournissent *un point de vue unifié sur la diagnosticabilité*, pour les approches mentionnées dans ce document.

Chapitre 11

La diagnosticabilité à travers les signatures de faute

Nous avons montré précédemment que les différentes approches du diagnostic à base de modèles peuvent être classées en deux catégories : les approches à base d'états et à base d'évènements.

Nous avons montré dans le chapitre 3 et résumé dans le chapitre 10 que les concepts fondamentaux de faute et mode de faute peuvent être modélisés d'une manière unifiée. Les définitions unifiées 10.2 à 10.5 sont adoptées dans le reste de ce document.

Les autres définitions 10.6 à 10.9, concernent la diagnosticabilité. Elles reposent sur un *ensemble d'observables OBS*. Ces définitions unifient les approches à base d'états dans lesquelles une observation est une valeur pour le tuple des variables observables. Toutefois, dans les approches à base d'évènements, il est prouvé que les observations sont organisées en langages, et que le diagnostic repose sur cette structuration. Par conséquent, la nature différente des observations appelle à d'autres développements avant d'appliquer ces définitions « unifiées »aux approches à base d'évènements.

Cette partie résume le coeur de la contribution de la thèse. D'abord, l'ensemble des observables *OBS* est défini pour les approches à base d'évènements, et les définitions résultantes de signature et diagnosticabilité sont données. Il est prouvé que les définitions du chapitre 10 unifient *toutes* les approches.

Ensuite, le problème de la complexité introduit par le grand nombre de paires de modes de fautes est adressé. Le concept de mode de faute partiel, inspiré du concept de diagnostic partiel défini dans les approches DX, est introduit. L'application des signatures de fautes aux modes de faute partiels ouvre la porte

à une approche algorithmique efficace pour l'analyse de la diagnosticabilité. Une implantation de cet algorithme est décrite.

Finalement, le concept de signature de faute est étendu, dans le cas des approches à base d'états, à n'importe quelle propriété qui définit un ensemble d'états. Le but de cette extension est de permettre d'analyser la diagnosticabilité non seulement des fautes, mais aussi d'autres propriétés comme les préconditions de réparation, la qualité de service, etc.

11.1 Signatures pour approches à base d'évènements

Dans les approches à base d'états, les observations sont modélisées au moyen de variables observables. Dans la plupart des cas, ces variables sont discrètes : en FDI, les variables continues sont impliquées dans des test de cohérence dont les résultats sont en général booléens (succès ou échec du test). Le raisonnement de diagnostic reçoit en entrée ces variables discrètes, et fournit une explication en termes de fautes ou modes de faute. Un aspect important est que durant le raisonnement de diagnostic, chaque variable discrète a une valeur fixe, ce qui signifie que le système n'évolue pas du point de vue du diagnostic.

Dans les approches à base d'évènements, les observations sont modélisées par des évènements observables. Le processus de diagnostic reçoit en entrée la séquence de tous les évènements qui ont été observés pendant la vie du système. Cette prise en compte de l'histoire du système est la différence fondamentale entre les approches de diagnostic à base d'états et d'évènements. Elle impacte fortement le raisonnement de diagnostic, puisque dans les approches à base d'états, le processus de diagnostic n'est pas redémarré à chaque fois qu'une nouvelle observation est reçue : le processus garde en mémoire l'histoire du système, et met à jour les candidats au diagnostic lorsque de nouvelles observations sont reçues. La diagnosticabilité tient elle aussi compte de la possibilité d'évolution du système, et requiert simplement que le délai entre l'occurrence d'une faute et de ses symptômes caractéristiques soit bornée.

La définition d'un ensemble d'observables est fortement liée aux durées de vie relatives des observations et du processus de diagnostic. Dans les approches à base d'états, puisque le diagnostic est effectué sur une unique observation, il ne peut pas utiliser des relations entre deux observations. Les observables peuvent par conséquent être rassemblés en un simple ensemble. Dans les approches à base d'évènements, le diagnostic est basé sur une observation finie du système, durant laquelle plusieurs évènements observables apparaissent. Le diagnostic est basé sur les séquences d'évènements observées, mais à chaque nouvel évènement observable, la séquence évolue. Considérer l'*ensemble* de toutes les séquences d'évènements observables n'est pas suffisant pour la diagnosticabilité. Il est également nécessaire de considérer le *langage* des évènements observables, puisque la diagnosticabilité est liée à la notion d'extension d'une

séquence d'évènements.

La solution adoptée ici repose sur un point de vue original. Les concepts d'*observation* et d'*observable*, qui n'étaient pas clairement distingués jusqu'à présent, sont clarifiés.

Dans les systèmes à base d'évènements, une observation est une séquence finie d'évènements ; toutefois, n'importe quelle observation peut potentiellement être continuée, étendue. Donc les observables, c'est-à-dire les observations potentielles, son considérés comme étant les continuations infinies des séquences finies d'évènements observables. Lorsque deux observables infinis sont différents, ils diffèrent à un index fini, donné, i. Une observation finie de longueur supérieure à i est alors suffisante pour distinguer ces deux observables infinis. Nous distinguons alors l'*ensemble des observations*, issu de la projection du langage L_{sys} sur les évènements observables, de l'*ensemble des observables OBS* qui contient des séquences infinies d'évènements observables.

Le chapitre 4 montre comment les automates de Büchi et les ω-langages permettent de définir formellement l'ensemble *OBS* pour les approches à base d'évènements. Il est prouvé que la définition 10.9 de la diagnosticabilité des approches à base d'états est équivalente à la définition classique de la diagnosticabilité des approches à base d'évènements.

11.2 Signatures pour modes de fautes partiels

Cette section décrit comment le travail d'unification décrit précédemment nous a conduit à une nouvelle caractérisation de la diagnosticabilité qui nous permet d'effectuer son analyse de manière efficace.

Le concept de signature a été appliqué jusqu'à présent aux modes de fautes. Nous définissons en outre les *modes de faute partiels*, et leur adaptons la définition de signature. Il est montré comment le calcul des signatures des modes de faute partiels permet de diminuer le nombre de comparaisons de signatures pour l'analyse de la diagnosticabilité.

Nous adoptons ici la représentation par variables des modes de faute (définition 10.5), initialement donnée dans les approches à base d'états. En utilisant des variables de mode binaires, la traduction en représentation par ensembles est directe, ainsi que nous l'avons montré en section 9.

Les concepts de mode de faute partiel et sa signature sont indépendants du modèle. Ils sont appliqués dans un contexte distribué, à l'aide de modèles à base de contraintes. Un algorithme pour le calcul des signatures et de leur intersection est présenté, basé sur l'algorithme de diagnostic défini en [Pucel *et al.*, 2007, Ardissono *et al.*, 2005, Console *et al.*, 2007]. L'analyse de diagnosticabilité basée sur les signatures de modes de faute partiels est alors effectuée.

Un *mode de faute* est relatif au système tout entier : il décrit le comportement de *tous* les composants du système. Dans la représentation par variables, il est représenté par une affectation à toutes les variables de mode. Le diagnostic consiste à décider dans quel mode de faute le système se trouve. Notre approche est basée sur l'analyse des *modes de faute partiels*, qui sont relatifs à *certains* des composants du système.

Définition 11.1 (Mode de faute partiel) *Un* mode de faute partiel *est défini en affectant une valeur à* certaines *des variables de mode du système. Un mode de faute partiel qui affecte* toutes *les variables de modes est un* mode de faute.

La portée *d'un mode de faute partiel pfm est l'ensemble des variables de mode affectées par pfm. Il est noté* Sco*(pfm).*

Le rang *d'un mode de faute partiel est la cardinalité de sa portée.*

Deux modes de faute partiels pfm_1 et pfm_2 sont alternatifs *si et seulement si ils ont la même portée mais sont différents.*

Un mode de faute partiel pfm_1 raffine *un autre mode de faute partiel pfm_2 si et seulement si* $\text{Sco}(pfm_2) \subset \text{Sco}(pfm_1)$ *et* $pfm_1 \Rightarrow pfm_2$.

Par exemple, $m_1 = ok \wedge m_2 = ok$ et $m_1 = ok \wedge m_2 = ab$ sont alternatifs, tandis que $m_1 = ab \wedge m_2 = ok$ raffine $m_1 = ab$.

Raisonner sur des modes de faute partiels nous permet de raisonner sur certaines parties du système, sans tenir compte du reste. En fait, deux modes de faute partiels alternatifs décrivent des situations de faute différentes dans un sous-système, soit deux ensembles de situations de faute du système. Définir les signatures pour les modes de faute partiels nous permet d'analyser la diagnosticabilité en commençant par des sous-parties du système.

Définition 11.2 (Signature de mode de faute partiel) *La* signature *d'un mode de faute partiel est l'union des signatures de tous les modes de fautes qui le raffinent.*

$$Sig(pfm) = \bigcup_{f \in \mathcal{F}_{sys} \,|\, f \text{ raffine } pfm} Sig(f)$$

Dans certains formalismes, il est possible de calculer les signatures pour des modes de faute partiels. Ceci peut être très utile pour l'analyse de diagnosticabilité.

Définition 11.3 (Discriminabilité, Diagnosticabilité) *Deux modes de faute partiels sont* discriminables *si et seulement si leurs signatures sont disjointes.*

Un mode de faute partiel est diagnosticable *si et seulement si il est discriminable de toutes ses alternatives.*

Un système est diagnosticable *si et seulement si tous les modes de faute partiels sont diagnosticables.*

Une conséquence des définitions ci-dessus est que si deux modes de faute partiels pfm_1 et pfm_2 sont discriminables, alors tous les modes de fautes raffinant pfm_1 sont discriminables de tous les modes de fautes raffinant pfm_2. En comparant les signatures d'une paire de modes de fautes partiels, nous obtenons des informations sur un grand nombre de modes de fautes.

11.3 Signatures d'ensembles d'états

Cette section résume une généralisation de la diagnosticabilité, non seulement aux modes de faute et modes de faute partiels, mais à n'importe quel ensemble d'états d'un système à base d'états. Dans un tel système, une propriété correspond généralement à un ensemble d'états. Cette généralisation nous permet donc d'analyser la diagnosticabilité, par exemple, des préconditions de réparation, ou d'un niveau de qualité de service.

Le diagnostic et l'analyse de diagnosticabilité sont des domaines de recherche matures, et le problème de leur intégration dans un outil de supervision général reçoit un intérêt croissant [wsdiamond, 2005 2008, Cordier *et al.*, 2007]. La diagnosticabilité des préconditions de réparation est étudiée dans [Cordier *et al.*, 2007] par l'introduction du concept de *macrofaute*, et nous montrons que notre approche est une généralisation de la solution qui y est décrite.

Notre définition généralisée est exprimée par rapport aux états au lieu des observables et des signatures. Cet aspect original fournit une caractérisation duale de la diagnosticabilité, ce qui peut être utile dans certains contextes dans lesquels l'ensemble des observables est plus difficile à caractériser que l'ensemble des états.

Définition 11.4 (Bloc diagnosticable) *Soit $=_{OBS}$ la relation d'équivalence définie sur l'ensemble des états SD par :*

$$\forall s_1, s_2 \in SD, s_1 =_{OBS} s_2 \Leftrightarrow s_1 \text{ et } s_2 \text{ mènent à la même observation}$$

Chaque classe d'équivalence de $=_{OBS}$ est un bloc diagnosticable *du système. L'ensemble des blocs diagnosticables du systèmes est l'ensemble quotient de SD par $=_{OBS}$.*

Définition 11.5 (Diagnosticabilité) *Une propriété ou l'ensemble correspondant $S \subseteq SD$ est diagnosticable si et seulement si S peut être construit par union de blocs diagnosticables.*

Il est montré dans le chapitre 6 que cette définition élargit la définition unifiée 10.9. De plus, la définition d'une propriété supplémentaire, nommée *préemptabilité* permet de comparer et de généraliser l'approche utilisant des macrofautes décrite en [Cordier *et al.*, 2007].

11.4 Application et aspects algorithmiques

Cette section résume l'application de l'approche de diagnosticabilité définie dans la section 11.2 aux web services, ou plus généralement aux architectures orientées service (SOA). Le contexte des architectures orientées service est particulièrement contraignant. La distribution du système, la grande modularité et la privauté de certaines données rendent la supervision difficile. Notre implantation utilise des affectations partielles, ce qui nous permet de répondre aux contraintes de distribution et de privauté, et il est montré que notre implantation offre une modularité suffisante pour s'intégrer dans de telles architectures.

Conclusion

Dans cette thèse, un point de vue unifié sur la diagnosticabilité a été développé, à partir d'une comparaison des approches correspondant aux différents formalismes de modélisations utilisés dans ce domaine de recherche. Les notions d'ensemble d'observables et de signature de faute ont été étendues aux approches à base d'évènements, ce qui nous a permis d'exprimer la diagnosticabilité avec une définition formelle unifiée. Cette définition exprime l'idée que deux combinaisons de fautes différentes ne doivent pas pouvoir mener à une observation commune, sous peine de rendre le système non diagnosticable. Cette unification de la diagnosticabilité et des concepts sous-jacents ouvre de nombreuses perspectives. Les contributions utilisées dans le cadre d'un formalisme particulier peuvent être adaptées à d'autres formalismes. En particulier les concepts de mode de faute partiel et de bloc diagnosticable, définis pour les approches à base d'états, peuvent être adaptés aux approches à base d'évènements, et sont susceptibles de permettre des généralisations équivalentes par rapport aux approches classiques de diagnosticabilité.

Le problème du diagnostic basé sur plusieurs modèles avec différents formalismes a été adressé dans [Ressencourt, 2008], et illustre le besoin pour de telles applications. L'unification des définitions de diagnosticabilité est un pas important vers l'analyse de la diagnosticabilité utilisant des modèles multiples. La partie la plus difficile du diagnostic à base de modèle a souvent été considérée comme la phase de modélisation du système [Hamscher *et al.*, 1992]. La possibilité d'utiliser différents modèles est très prometteuse pour faciliter cette phase de modélisation. Deux modèles différents peuvent être utilisés afin de modéliser des aspects différents d'un même composant. De nombreuses contraintes peuvent influencer le choix d'un formalisme de modélisation, et ce travail fournit une plus grande liberté dans le choix du modèle, ainsi qu'un cadre unifié pour l'analyse de diagnosticabilité. Ceci est particulièrement vrai dans le cas des systèmes hybrides qui couplent des modèles à base d'états et d'évènements. Les approches qui s'intéressent à de tels systèmes devraient bénéficier directement de la diagnosticabilité unifiée.

La définition unifiée de la diagnosticabilité est indépendante du modèle sous-jacent, et se concentre sur la relation entre les fautes et les observables. Ce point de vue de haut niveau nous a permis d'adapter efficacement un algorithme de diagnostic à base de propagation de contraintes pour l'analyse de la diagnosticabilité, en introduisant le concept de mode de faute partiel. Cette approche

nous permet de raisonner sur des sous-parties du systèmes, ainsi que de diminuer de manière significative le nombre de comparaisons de signatures. C'est à notre connaissance une approche nouvelle, sans équivalent dans le monde des approches à base d'états ou d'évènements. Le concept de mode de faute partiel et ses propriétés ont été appliqués à une approche distribuée à base d'états, montrant la dualité avec le diagnostic à base de contraintes [Console *et al.*, 2007] dont le raisonnement est basé exclusivement sur des affectations partielles. Nous affirmons qu'il est possible d'adapter cet algorithme aux approches à base d'évènements en adoptant le formalisme des ω-langages décrits dans le chapitre 4. Nous avons appliqué cette approche de diagnosticabilité dans le contexte des architectures orientées services, et l'avons implanté. Cette implantation est hautement modulaire en réponse aux contraintes des architectures orientées service, et fournit une interface originale bénéficiant des propriétés des modes de faute partiels, qui permet au concepteur de naviguer dans les résultats de l'analyse.

L'application du diagnostic à base de modèle et de l'analyse de la diagnosticabilité aux systèmes logiciels est encore un domaine d'application jeune. Ce champ d'applications ouvre de nombreuses connexions avec le domaine de la sécurité informatique, dans lequel le diagnostic est la plupart du temps ignoré au bénéfice de réactions aux symptômes qui peuvent être inadaptées. Avec le besoin croissant de systèmes logiciels complexes, de telles applications sont susceptibles d'entraîner un engouement dans le monde de la recherche et de l'industrie. Le problème du diagnostic dans l'environnement des réseaux de communications a reçu de l'intérêt dans [Pencolé and Cordier, 2005], au niveau du matériel et des protocoles de signalisation. Dans cette thèse, le niveau applicatif des réseaux de communication est considéré. Ces deux applications sont complémentaires et leur intégration dans un environnement général de diagnostic de réseaux communicants est une perspective intéressante. Les réseaux adaptatifs, réseaux ad hoc et les réseaux de capteurs offrent également des contextes d'application intéressants pour le diagnostic à base de modèles.

L'analyse de diagnosticabilité se déroule pendant la phase de conception, et sert à fournir un retour au concepteur du système, afin d'optimiser le système ou son modèle pour le diagnostic. Le problème d'intégrer la diagnosticabilité dans un outil général d'assistance à la conception est aussi adressé dans cette thèse. Dans le contexte des systèmes à base d'états, la définition unifiée a été généralisée afin de prendre en compte n'importe quel ensemble d'états. Les ensembles d'états peuvent caractériser des propriétés comme les fautes, ce qui nous ramène aux approches classiques, mais peuvent aussi caractériser d'autres types de propriétés, comme les préconditions de réparation, les niveaux de qualité de service. Une approche algorithmique qui permet de vérifier ces propriétés est une perspective importante. L'adaptation aux approches à base d'évènements, en utilisant le formalisme des ω-langages, devrait mener à une comparaison intéressante avec l'approche à base de motifs de supervisions [Jéron *et al.*, 2006] ainsi qu'avec l'approche utilisant la logique temporelle linéaire [Jiang and Kumar, 2002], puisque ces approches permettent aussi d'analyser non seulement les fautes, mais toute propriété qui peut être exprimée dans un certain langage. Ce travail fournit une base pour appliquer les techniques d'analyse de diagnosticabilité à l'analyse de diverses propriétés.

C'est un pas vers un outil de validation qui assure qu'un système est adapté à la supervision dès la phase de conception. Le but final est de fournir des outils permettant de concevoir plus facilement des systèmes plus fiables.

Bibliography

[Ardissono *et al.*, 2005] Liliana Ardissono, Luca Console, Anna Goy, Giovanna Petrone, Claudia Picardi, Marino Segnan, and Daniele Theseider Dupré. Enhancing web services with diagnostic capabilities. In *Proceedings of the 3rd IEEE European Conference on Web Services*, Växj Sweden, 2005.

[Autio and R.Reuter, 1998] K. Autio and R.Reuter. Structural abstraction in model-based diagnosis. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI'98)*, pages 269–273, Brighton, UK, 1998.

[Baldoni *et al.*, 2004] Matteo Baldoni, Cristina Baroglio, Alberto Martelli, and Viviana Patti. Reasoning about interaction protocols for web service composition. In *Proc. of 1st Int. Workshop on Web Services and Formal Methods, WS-FM 2004, volume 105 of Electronic Notes in Theoretical Computer Science*, pages 21–36. Elsevier Science Direct, 2004.

[Bayoudh *et al.*, 2006] M. Bayoudh, L. Travé-Massuyès, and X. Olive. Hybrid systems diagnosability by abstracting faulty continuous dynamics. In *Proceedings of the 17th International Workshop on Principles of Diagnosis DX'06*, Burgos, Spain, 2006.

[Bayoudh *et al.*, 2008] M. Bayoudh, L. Travé-Massuyès, and X. Olive. Coupling continuous and discrete event system techniques for hybrid system diagnosability analysis. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI2008)*, pages 219–223, Patras, Greece, 2008.

[Biswas *et al.*, 2006] Santosh Biswas, Dipankar Sarkar, Siddhartha Mukhopadhyay, and Amit Patra. Diagnosability analysis of real time hybrid systems. In *Proceedings of the IEEE International Conference on Industrial Technology, ICIT'2006*, pages 104–109, Mumbai, India, December 2006.

[Busi *et al.*, 2005] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Towards a formal framework for choreography. In *WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, pages 107–112, Washington, DC, USA, 2005. IEEE Computer Society.

[Chittaro and Ranon, 2004] Luca Chittaro and Roberto Ranon. Hierarchical model-based diagnosis based on structural abstraction. *Advanced Engineering Informatics*, 155(1-2):147–182, 2004.

[Chittaro *et al.*, 1993] Luca Chittaro, Giovanni Guida, Carlo Tasso, and Elio Toppano. Functional and teleological knowledge in the multimodeling approach for reasoning about physical systems: A case study in diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1718–1751, November/December 1993.

[Console *et al.*, 2007] Luca Console, Claudia Picardi, and Daniele Theseider Dupré. A framework for decentralized qualitative model-based diagnosis. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI'07*, pages 286–291, Hyderabad, India, January 2007.

[Cordier *et al.*, 2004] Marie-Odile Cordier, Philippe Dague, François Lévy, Marcel Staroswiecki, and Louise Travé-Massuyès. Conflicts versus analytical redundancy relations: a comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives. *IEEE Transactions on Systems, Man, and Cybernetics Part B*, 34(5):2163–2177, October 2004.

[Cordier *et al.*, 2006] Marie-Odile Cordier, Louise Travé-Massuyès, and Xavier Pucel. Comparing diagnosability in continuous and discrete-event systems. In *Proceedings of the 17th International Workshop on Principles of Diagnosis, DX'06*, pages 55–60, 2006.

[Cordier *et al.*, 2007] Marie-Odile Cordier, Yannick Pencolé, Louise Travé-Massuyès, and Thierry Vidal. Self-healability = diagnosability + repairability. In *Proceedings of the 18th International Workshop on Principles of Diagnosis, DX'07*, pages 265–272, Nashville, TN, USA, May 2007.

[Daigle *et al.*, 2008] Matthew Daigle, Xenofon Koutsoukos, and Gautam Biswas. An event-based approach to hybrid systems diagnosability. In *Proceedings of the 19th International Workshop on Principles of Diagnosis DX'08*, Blue Mountains, Australia, September 2008.

[DAML,] DARPA Agent Markup Language Program DAML. Owl-s: Semantic markup for web services. http://www.daml.org/services/owl-s/1.1/overview/.

[Darwiche, 1999] Adnan Darwiche. Utilizing device behavior in structure–based diagnosis. In *In Proceedings of the 16th International Joint Conference on Artificial Intelligence, IJCAI'99*, pages 1096–1101, Stockholm, Sweden, August 1999.

[Dechter, 2003] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[Dressler and Struss, 2003] Oskar Dressler and Peter Struss. A toolbox integrating model-based diagnosability analysis and automated generation of diagnostics. In *Proceedings of the 14th International Workshop on Principles of Diagnosis, DX'03*, 2003.

[Dubuisson, 2001] Bernard Dubuisson. *Diagnostic, intelligence artificielle et reconnaissance des formes*. ic2 productique. Hermes Science Publications, 2001.

[Eder *et al.*, 2006] Johann Eder, Marek Lehmann, and Amirreza Tahamtan. Choreographies as federations of choreographies and orchestrations. In *ER (Workshops)*, pages 183–192, 2006.

[Fourlas *et al.*, 2002] G.K. Fourlas, Kostas J. Kyriakopoulos, and N. J Krikelis. Diagnosability of hybrid systems. In *Proceedings of the 10th Mediterranean Conference on Control and Automation, MED'2002*, Lisbon, Portugal, 2002.

[Friedrich *et al.*, 2005] Gerhard Friedrich, Georg Gottlob, and Wolfgang Nejdl. Formalizing the repair process — extended report. *Annals of Mathematics and Artificial Intelligence*, 11(1-4):187–201, april 2005.

[Frisk, 2000] Erik Frisk. Residual generator design for non-linear, polynomial systems - a gröbner basis approach. In *Proceedings of IFAC Safeprocess*, 2000.

[Haar *et al.*, 2003] Stefan Haar, Albert Benveniste, Eric Fabre, and Claude Jard. Partial order diagnosability of discrete event systems using petri nets unfoldings. In *Proceedings of the 42nd IEEE Conference on Decision and Control (CDC)*, Hawaii, USA, December 2003.

[Hamadi and Benatallah, 2003] Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In *ADC '03: Proceedings of the 14th Australasian database conference*, pages 191–200, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[Hamscher *et al.*, 1992] Walter Hamscher, Luca Console, and Johan de Kleer. *Readings in model-based diagnosis*. Morgan Kaufmann Publishers Inc, 1992.

[Isermann, 2005] Rolf Isermann. Model-based fault-detection and diagnosis — status and applications. *Annual Reviews in Control*, 29(1):71–85, May 2005.

[Jéron *et al.*, 2006] Thierry Jéron, Hervé Marchand, Sophie Pinchinat, and Marie-Odile Cordier. Supervision patterns in discrete event systems diagnosis. In *Proceedings of the 8th International Workshop on Discrete Event Systems, (WODES'06)*, pages 262–268, Ann-Arbor (MI, USA), July 2006.

[Jiang and Kumar, 2002] Shengbing Jiang and Ratnesh Kumar. Failure diagnosis of discrete event systems with linear-time temporal logic fault specifications. *IEEE Transactions on Automatic and Control*, 49(6):934–945, June 2002.

[Jiang *et al.*, 2001] Shengbing Jiang, Zhongdong Huang, Vigyan Chandra, and Ratnesh Kumar. A polynomial algorithm for testing diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 46(8):1318–1321, August 2001.

[Krysander *et al.*, 2008] 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*, 38(1):197–206, January 2008.

[Lind, 2003] Morten Lind. Making sense the abstraction hierarchy in the power plant domain. *Cognition, Technology & Work*, 5(2):67–81, June 2003.

[Lopéz-Varela, 2007] Carmen G. Lopéz-Varela. *Détection et diagnostic basés cohérence*. PhD thesis, LAAS-CNRS, Université de Toulouse, 2007.

[Meinel and Theobald, 1998] Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.

[Nyberg, 2002] Mattias Nyberg. Criterions for detectability and strong detectability of faults in linear systems. *International Journal of Control*, 75(7):490–501, May 2002.

[OASIS, 2004] Organization for the Advancement of Structured Information Standards OASIS. Universal description, discovery and integration v3.0.2 (uddi). http://uddi.org/pubs/uddi_v3.htm, October 2004.

[OASIS, 2007] Organization for the Advancement of Structured Information Standards OASIS. Web services business process execution language version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, April 2007.

[Pencolé and Cordier, 2005] Yannick Pencolé and Marie-Odile Cordier. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence Journal*, 164(1–2):121–170, 2005.

[Pencolé, 2005] Yannick Pencolé. Assistance for the design of a diagnosable component-based system. In *17th IEEE International Conference on Tools with Artificial Intelligence*, Hong-Kong, November 2005.

[Peterson, 1981] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, 1981.

[Polycarpou *et al.*, 1997] Marios M. Polycarpou, Arun T. Vemuri, and Amy R. Ciric. Nonlinear fault diagnosis of differential/algebraic system. In *Proceedings of IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes*, pages 510–515, 1997.

[Pucel *et al.*, 2007] Xavier Pucel, Stefano Bocconi, Claudia Picardi, Daniele Theseider Dupré, and Louise Travé-Massuyès. Diagnosability analysis for web services with constraint-based models. In Gautam Biswas, Xenofon Koutsoukos, and Sherif Abdelwahed, editors, *Proceedings of the 18th International Workshop on Principles of Diagnosis, DX'07*, pages 360–367, Nashville, TN, USA, May 2007.

[Pucel *et al.*, 2008] Xavier Pucel, Louise Travé-Massuyès, and Yannick Pencolé. Another point of view on diagnosability. In *Proceedings of the 19th International Workshop on Principles of Diagnosis, DX'08*, Blue Mountains, Australia, September 2008.

[Puig *et al.*, 2005] V. Puig, J. Quevedo, T. Escobet, and B. Pulido. On the integration of fault detection and isolation in model based fault diagnosis. In *Proceedings of the 16th International Workshop on Principles of Diagnosis (DX'05)*, pages 227–232, 2005.

[Ramírez-Treviño *et al.*, 2004] Antonio Ramírez-Treviño, Elvia Ruiz-Beltrán, Israel Rivera-Rangel, and Ernesto López-Mellado. Diagnosability of discrete event systems: a petri net based approach. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '04)*, pages 541–546, New Orleans LA (USA), April 2004.

[Ressencourt, 2008] Hervé Ressencourt. *Diagnostic hors-ligne à base de modèles : Approche multimodèle pour la génération automatique de séquences de tests ; Application au domaine de l'automobile*. PhD thesis, LAAS-CNRS, Université de Toulouse, 2008.

[Rintanen and Grastien, 2007] Jussi Rintanen and Alban Grastien. Diagnosability testing with satisfiability algorithms. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 532–537, January 2007.

[Sampath *et al.*, 1995] Meera Sampath, Raja Sengupta, Stephane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete event system. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, September 1995.

[Sampath *et al.*, 1996] Meera Sampath, Raja Sengupta, Stephane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete event system. *IEEE Transactions on Control Systems Technology*, 4(2):105–124, March 1996.

[ten Teije *et al.*, 2004] A. ten Teije, F. van Harmelen, and B. Wielinga. Configuration of web services as parametric design. In E. Motta, N. Shadbolt, A. Stutt, and N. Gibbins, editors, *Proceedings of the 14th International Conference, EKAW-2004*, number 3257 in Lecture Notes in Artificial Intelligence, pages 321–336, Whittleburry Hall, UK, October 2004. Springer Verlag. ISBN 3-540-23340-7.

[Travé-Massuyès *et al.*, 2006a] Louise Travé-Massuyès, Marie-Odile Cordier, and Xavier Pucel. Comparing diagnosability in cs and des. In *Proceedings of the 6th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, Beijing, China, August 2006.

[Travé-Massuyès *et al.*, 2006b] Louise Travé-Massuyès, Teresa Escobet, and Xavier Olive. Diagnosability analysis based on component-supported analytical redundancy relations. *IEEE Transactions on Systems, Man, and Cybernetics Part A*, 36(6):1146–1160, November 2006.

[van der Aalst *et al.*, 2006] Wil M. P. van der Aalst, Marlon Dumas, Chun Ouyang, Anne Rozinat, and H. M. W. Verbeek. Choreography conformance checking: An approach based on bpel and petri nets. In Frank Leymann, Wolfgang Reisig, Satish R. Thatte, and Wil M. P. van der Aalst, editors, *The Role of Business Processes in Service Oriented Architectures*, volume 06291 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), S chloss Dagstuhl, Germany, 2006.

[W3C, 1999] World Wide Web Consortium W3C. Xsl transformations (xslt) version 1.0. http://www.w3.org/TR/xslt, November 1999.

[W3C, 2004] World Wide Web Consortium W3C. Xml schema part 0: Primer second edition. http://www.w3.org/TR/xmlschema-0/, October 2004.

[W3C, 2005a] World Wide Web Consortium W3C. Web service semantics - wsdl-s. http://www.w3.org/Submission/WSDL-S/, November 2005.

[W3C, 2005b] World Wide Web Consortium W3C. Web services choreography description language version 1.0. http://www.w3.org/TR/ws-cdl-10/, November 2005.

[W3C, 2006] World Wide Web Consortium W3C. Extensible markup language (xml) 1.1 (second edition). http://www.w3.org/TR/xml11/, August 2006.

[W3C, 2007a] World Wide Web Consortium W3C. Soap version 1.2 part 1: Messaging framework (second edition). http://www.w3.org/TR/soap12-part1/, April 2007.

[W3C, 2007b] World Wide Web Consortium W3C. Web services description language (wsdl) version 2.0 part 1: Core language. http://www.w3.org/TR/wsdl20/, June 2007.

[Wen and Jeng, 2004] YuanLin Wen and MuDer Jeng. Diagnosability of petri nets. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 4891–4896, The Hague, Netherlands, October 2004.

[Wikipedia,] The free encyclopedia Wikipedia. Docbook. http://en.wikipedia.org/wiki/DocBook, permanent link: *http://en.wikipedia.org/w/index.php?title=DocBook&oldid=256309262*.

[wsdiamond, 2005 2008] Web services diagnosability, monitoring and diagnosis. http://wsdiamond.di.unito.it/, 2005–2008. European Project IST-516933.

[Yoo and Lafortune, 2002] Tae-Sic Yoo and Stéphane Lafortune. Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Transactions on Automatic Control*, 47(9):1491–1495, September 2002.

Un point de vue unifié sur la diagnosticabilité

Mots-clé : Diagnosticabilité, Diagnostic, Raisonnement à base de modèle, Détection et identification de défaillances, Tolérance aux fautes.

Résumé : Le problème du diagnostic de défaillances à base de modèle dans les systèmes complexes a reçu un intérêt croissant durant les dernières décennies. Ce problème doit être pris en compte dès la phase de conception du système, au moyen de l'analyse de la diagnosticabilité. La diagnosticabilité est la propriété d'un système consistant à exhiber des symptômes différents pour un ensemble de situations de défaillances anticipées. Plusieurs approches ont été développées basées sur différents formalismes de modélisation, toutefois les raisonnements menant à la diagnosticabilité sont très semblables dans toutes ces approches. Cette thèse développe une comparaison des approches existantes et établit une définition unifiée de la diagnosticabilité. Une nouvelle approche pour l'analyse de diagnosticabilité, basée sur les modes de faute partiels, est décrite et implémentée dans le contexte des architectures orientées services, plus précisément sur des web services. Une nouvelle généralisation de la définition de la diagnosticabilité à n'importe quel ensemble d'état est présentée, qui permet de prendre en compte de nouveaux types de propriétés, comme les préconditions de réparation, ou la qualité de service. Ces travaux ouvrent des perspectives pour le raisonnement de diagnosticabilité indépendant du modèle, pour la diagnosticabilité basée sur d'autres types de modèles, ainsi que pour l'intégration du diagnostic dans un outil de surveillance plus général. Le diagnostic et l'analyse de diagnosticabilité des systèmes logiciels est encore un domaine jeune, et ouvre de nombreuses connections avec le domaine de la sécurité informatique.

Another point of view on diagnosability

Keywords : Diagnosability, Diagnosis, Model based reasoning, Fault detection and identification, Fault Tolerance.

Abstract : The problem of model-based fault diagnosis in complex systems has received an increasing interest over the past decades. Experience has proved that it needs to be taken into account during the system design stage, by means of diagnosability analysis. Diagnosability is the ability of a system to exhibit different symptoms for a set of anticipated fault situations. Several approaches for diagnosability have been developed using different modelling formalisms., yet the reasoning for diagnosability analysis is very similar in all these approaches. This thesis provides a comparison of these and a unified definition of diagnosability. An original approach for diagnosability analysis, based on partial fault modes, is described and implemented in the context of service oriented architecture, more precisely on web services. An original generalization of the definition of diagnosability to any set of system states is presented, that accounts for many kinds of properties, like repair preconditions or quality of service. This work opens perspectives for model independent diagnosability reasoning, diagnosability based on other types of models, and in integrating diagnosis into a general purpose supervision tool. Model-based diagnosis and diagnosability of software systems is still a young applicative domain, and opens many connections with the software safety engineering domain.