FAULT DIAGNOSTICS BASED ON TEMPORAL ANALYSIS

Michal Knotek

UNIVERSITÉ JOSEPH FOURIER-GRENOBLE

and

BRNO UNIVERSITY OF TECHNOLOGY

THESIS

for obtaining the degree of

DOCTEUR DE L'UNIVERSITÉ JOSEPH FOURIER (Ph.D.)

IN AUTOMATIQUE - PRODUCTIQUE

prepared at *Laboratoire d'Automatique de Grenoble*
and Department of Control and Instrumentation, FEEC BUT

presented and sat by

Michal KNOTEK

7 September 2006

FAULT DIAGNOSTICS BASED ON TEMPORAL ANALYSIS

Sylviane Gentil	Professor at INPG Grenoble, FR	President
Nouredinne Zerhouni	Professor at ENSMM Besançon, FR	Reviewer
Zineb Simeu-Abazi	Assoc.Professor at UJF, FR	Thesis director
František Zezulka	Professor at FEEC BUT, CZ	Thesis director
Miroslav Švéda	Professor at FIT BUT, CZ	Reviewer
Ivo Provazník	Professor at FEEC BUT, CZ	Examinator

Contents

Acknowledgements

I would like to thank my supervisors for proposing such an interesting and challenging mission with many various paths to be traversed, many things to learn. Thank my czech director František Zezulka for support of my work and possibility to work on interesting academic projects. Je remercie Zineb Simeu-Abazi pour m'avoir accueilli au LAG et pour m'avoir donne la possibilite de faire la recherche au sein du laboratorie d'automatique connu au niveau mondial. Merci pour les discussions nombreuses quelle m'aide trouver le contenue de cette thèse.

Then, I am grateful to Hassan Alla for putting his interests in our work and sharing his experience, ideas and solutions to hot issues. I am glad for the experience to cooperate with working group SEDEP at LAG.

Thanks my friends and colleagues who were around me. Thanks Michal for you hospitality and support during defence, Vašek for your language and LaTeXcorrections. Thanks Ondřej , my college and friend from BUT for his small and big help on the Czech side. Thanks Khaled to share kindly our office. Thanks my Grenoble friends for spending funny time on the French side: Carmen, Melha, John and Lina, Salvador and others.

Thanks my buddy Jarda and my others mountains friends for good time on the mountains which were important aspect during researching period.

Thanks my Aikido teachers and friends, François and Gaëtan and Isabelle. You were important part of my life in Grenoble.

My deep thanks to my Bara for being with me, to my parents for the support which gave me during this period.

Keywords

fault diagnosis, fault detection, fault isolation, fault identification, time discrete-event systems, timed automata, modeling formalism, model-checking, backward time analysis, diagnoser, verification, correctness

Summary

Fault diagnosis plays a crucial role in protecting life and property, and in increasing operational time and productivity. A significant part of the engineering work in real-world systems (up to 80% is some cases) is dedicated to testing and diagnosis. Solving diagnostic problems for complex systems is a complicated task requiring a systematic approach. Fault diagnosis is one part of supervision systems. Fault diagnosis' purposes are fault detection and fault isolation. These phases are not always indissociable.

We propose a timed discrete event approach to fault diagnosis problem. This approach is applicable to systems that fall in the class of time discrete-event systems, which we call timed system. A system where correctness not only depends on the logical order of events but also on their timing. Many technological processes, for example communication or batch processes can be considered as timed systems. We employ temporal parameters to extend qualitative model with quantitative time. In this thesis, the fault diagnosis problem is studied in the context of timed automata. Timed automata represents powerful and very expressive tool for obtaining a realistic model. Timed automata model is also suitable for diagnostic analysis.

Our aim is to design an observer for a given plant, such that this observer can detect and isolate errors in the behaviour of the plant. We call this observer a "diagnoser". In many cases, a fault changes the timing of the output sequence. In these situations, timing information can be used to increase the accuracy and speed of fault diagnosis.

Hence, we assume a diagnoser building in the form of timed automaton. Built diagnoser will be called $\mathcal{D}$-diagnoser. This diagnoser is built independently of system component model. Diagnoser is built by integration of faultless and faulty behaviour. It is supposed that faulty and faultless behaviour is known, *i.e.* temporal parameters of dynamical behaviour can be obtained by process simulation. Our interest is directed towards finding faulty transitions. Faulty transition is considered as a fault signature, where

time plays important role to distinguish between faultless and faulty trajectory. The time diagnosability is defined as an ability to detect and isolate the fault in a finite time. Time aspect of diagnosability is expressed by two parameters. The first one expresses delays between the fault occurrence and ability to detect the fault and second one between occurrence and fault isolation. In this context, we show fault detection and isolation algorithm.

Thus, the methodology for modeling a complex physical system in a TDES framework is presented. Approach based on component is proposed for modeling a complex system. Each component is modeled by timed automaton. Fault presence can be easily described in the respective component (timed automaton model). We suppose that one component is a controller. Composition of all components together, makes "synchronous product" called dynamic global model $\mathcal{G}$. This composition creates a language of all possible evolutions. Model-checker[1] is useful tool for component based modeling and its analysis. Thanks to the model-checker, on-line evolution of the global dynamic model is obtained.

The model checking method is used for formal verification of built diagnoser. We propose a method in which the system model (dynamic global model $\mathcal{G}$) is compared with the diagnoser built ($\mathcal{D}$-Diagnoser). The correctness of the diagnoser is understood in such way, that the diagnoser should announce a fault if fault has occurred. This verification is done by model-checking method, when the property of fault implication is checked.

In the model-checker context, we treat the case of "Alarm treatment": an alarm signal is received and fault cause (origin) is searched. The methodology of this approach is shown using a backward time algorithm, where possibles traces are explored to find the coherent trace.

We use a simple real-world neutralisation batch process to demonstrate our modeling and diagnosis approach. Each development is illustrated at the end of the chapter by one example of application allowing a better comprehension of theoretical approaches.

As one of the perspectives, the problematics of quantised system is introduced. Diagnosis of quantised system based on the idea that continuous (or hybrid) system can be viewed as TDES at a higher level of abstraction (Chapter 5). The timed diagnosis approach of quantised system is illustrated on heat exchanger application.

[1]Model checker UPPAAL is used in this thesis.

Resumé

Dans le domaine de la sûreté de fonctionnement, le diagnostic joue un rôle primordial dans l'amélioration de la disponibilité opérationnelle des équipements. Dans les systèmes industriels, une part importante (jusqu'à 80%) est consacrée à la maintenance, test et diagnostic. Pour des systèmes complexes, la résolution des problèmes liés au diagnostic et d'une manière générale de la supervision nécessite la mise en oeuvre d'une approche générique. Le diagnostic concerne les deux phases indissociables de détection et de localisation. Dans cette thèse nous proposons une approche dynamique de diagnostic pour les systèmes à événements discrets. L'approche proposée basée sur l'exploitation du temps, est applicable à tout système dont l'évolution dynamique dépend non seulement de l'ordre des événements discrets mais aussi de la durée des tâches associées comme pour les processus de communication ou les processus batch. Dans cette thèse, le diagnostic des fautes est réalisé grâce à l'implémentation d'un modèle basé sur l'utilisation des automates temporisés. L'objectif est de concevoir un observateur pour un système donné, qui permet de détecter et localiser les éventuelles défaillances du procédé. Cet observateur est appelé "diagnoser". Une défaillance est constatée lorsque le séquencement temporel en sortie est incorrect. Nous présentons donc les différentes étapes de la démarche de diagnostic : la construction du diagnoser, la vérification du modèle ainsi qu'une l'application de la démarche sur un exemple réel avec son extension aux systèmes hybrides.

CHAPTER 1

Introduction

This chapter introduces various works on fault diagnosis. First, fault diagnosis is presented in the context of supervision. Different purposes correspond to different levels of supervisory system. Our focus will be fault detection and isolation, which are covered by the terms diagnosis or diagnostics. Fault diagnosis problem has been treated in different domains and contexts. The methods oriented at automatic control are presented. Our next focus is the discrete-event systems (DES) with time consideration, in so called Timed DES (TDES). The formulation of the problems is proposed, aims of dissertation are stressed. The objective of the work is an extension of an existing diagnostic approach: to take temporal information into account, so called timed diagnoser. Throughout this thesis, the main principles will be shown on an illustrative example. We have chosen the batch neutralisation process, which will be described and treated in the end of chapters. Finally, thesis outlook is presented.

Ce chapitre présente de différents travaux sur le diagnostic de défauts, étape incontournable dans la supervision des systèmes. Dans nos travaux, on s'intéresse au diagnostic et plus particulièrement aux phases de détection et d'isolation. Le problème de diagnostic de fautes touche différents domaines industriels et les méthodes varient suivant la nature des données que l'on exploite. Nos travaux concernent les systèmes à événement discret (SED) avec la prise en compte des contraintes temporelles, soit SED temporisés. L'objectif du travail est une extension d'une approche diagnostic existante avec la prise en compte de la variable temporelle, appelé diagnoseur (ou diagnostiqueur) temporisé. Les principaux développements seront détaillés dans ce rapport avec une illustration de la démarche sur un processus de neutralisation. Une conclusion et des perspectives de développement termineront ce document.

1.1. Fault diagnosis

The problem of failure diagnosis has received considerable attention in the domain of reliability engineering, automatic control and computer science. The increasingly stringent requirements on performance and reliability of complex technological systems have necessitated the development of sophisticated and systematic methods for diagnosis of system failures. Each community treats the diagnosis problem differently and this diversity of approaches makes the problematics of fault diagnosis rather huge and complex. The proof of the high requirement on safety system is international standard IEC 61508, titled "Functional safety of electrical/electronic/programmable electronic safety-related systems"[**IEC00, IEC05**] . Now, we introduce fault diagnosis in the context of supervision system.

1.1.1. Supervision system. Figure 1.1.1 illustrates the overall system architecture which contains fault diagnosis module. This architecture is two-level; at the lower level, it represents the system itself with the controller (or set of controllers). The upper level consists of the supervisor, which performs the task of control and coordination of the low level controllers, fault diagnosis and system reconfiguration/fault recovery. The interface between the two layers conveys information like occurrences of observable events in the system to the supervisor and communicates the commands issued by the supervisor to the system. [**SSL+95**]

1.1.2. Fault diagnosis purposes. The fault diagnosis represents one part of the supervision level. Supervision denotes the highest level of the integrated automation structure shown on a figure 1.1.2. The low level represents the instrumentation, which in turn represents choice and implementation of sensors and actuators. The control level works on the level of instrumentation, to which it sends the commands and receives sensor readings. The level of monitoring denotes visualisation and state observation of the system. The highest architecture level represents a supervision system which serves to control planning, fault diagnosis and control reconfiguration.

The fault diagnosis problem consists of the following phases:

Detection: Determination of faults present in a system and the time of detection

Isolation,identification: Determination of kind, location and time of the fault in question. It follows fault detection.

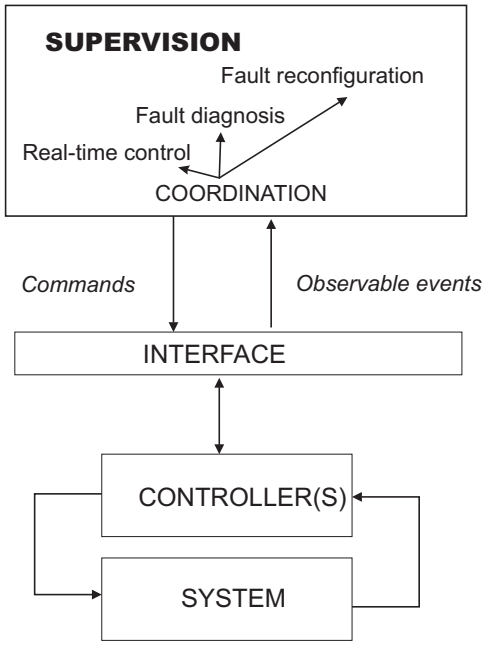


FIGURE 1.1.1. Supervision system

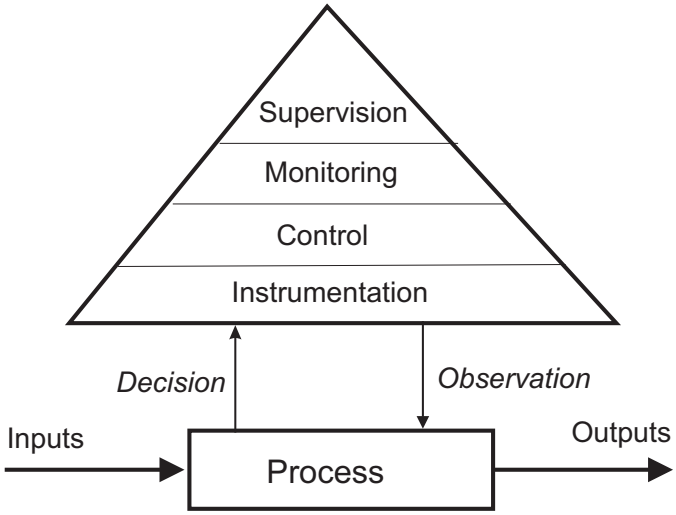


FIGURE 1.1.2. Integrated automation

Estimation: Determination of the size and behaviour of the fault over time.

Accommodation: A corrective action (reconfiguration or a change in the operation of the system) that prevents faults from propagating into undesired end-effect.

FIGURE 1.2.1. Fault diagnosis method

Our definition of fault diagnosis (diagnostics) denotes fault detection and fault isolation (identification). In this thesis, we will use both expressions (isolation and identification) with the same meaning. The expression fault isolation comes from control system community. The fault identification is a term used rather in the computer science domain.

1.2. Fault diagnosis methods

The wide variety of proposed schemes and methods can be divided into i) FDI methods based on mathematical model[1] ii) expert systems (DX) and other Artificial Intelligence based methods [2] iii) discrete-event systems (DES's) methods.

Fault Detection Isolation (FDI) methods. The FDI (quantitative) methods require analytical models of the physical process, which allow the comparison of sensor measurements with the predicted model, see *e.g.* [**MR00**, **Ise84**]. The first step of FDI procedures is to design a set of residuals that reflects the discrepancy between the actual behavior of system and the expected behavior given by its model. The second step is to design a decision procedure which is intended to separate cases in which discrepancies are caused by errors, uncertainties or measurement noises. These quantitative methods are also used for the diagnostic of hybrids systems (see *e.g.*[**CMS03**])

Diagnostic Expert systems (DX). Expert systems use the experience and knowledge of experts (stored as a set of rules) and an inference engine to diagnose failures. Expert systems are particularly useful in cases where it is hard to obtain a model for the plant. [**PD00**, **FLMM00**, **?**]

[1] community FDI = Fault Detection and Isolation
[2] community DX = Diagnostic eXpert system

Discrete-event system methods (DES). This class of the system describes an evolution of system states by changing the discrete quantity. Discrete-event system methods are model-based methods. Models describe system in different states and also its evolution by changing the states in corresponding transitions. DES systems naturally belong to qualitative modeling models. The behaviour is encoded by a DES modeling language. The faultless (and also faulty) model implemented in the DES tool represents qualitative sequences of events along with the transitions, see *e.g.*[**BKLS03, LSS01, PH00, SPVJ02, ZKW99**].

Fault-tree analysis seems to be the most popular approach. Fault trees can be synthesized automatically. However, they have a limitation: it is difficult to incorporate information about ordering and timing of event in a fault tree. [**Zad99**]

Continuous variable systems can often be viewed as DES act a higher level of system abstraction. The states of DES reflect normal and failed status of the system components. The major advantage of this approach is that it does not require detailed in-depth modeling of the system to be diagnosed and hence is ideally suited for diagnosis of systems which are difficult to model.

1.3. DES and TDES framework

The discrete-event system is described by the following properties [**Phi01**]:

- time set $\mathcal{T} = \mathbb{Z}$
- state space, a finite set of discrete states
- $\mathcal{U}$; set of inputs and outputs that is finite, also called input alphabet
- the transition map, defined as the next-state transition map

Note that for discrete-event systems there is no notion of time and the time set $\mathcal{T}$ is only used for ordering of the events. For description of DES (and later TDES) systems, we will use so called automaton (and timed automaton). Formal description is given later, in Section 2.1, now let we carry on with the overview of DES.

In general, discrete-event systems are systems in which the dynamics is event-driven (as opposed to time driven) and for which (at least some of) the variables required to describe the dynamics are discrete.

- Automata [**HU79**],
- Petri nets [**Rei85**],

- Linear systems in the (max, +)-algebra [**Ols93**],
- Markov chains [**Fre71**]

The discrete-event systems differ from continuous systems because they are not adequately modeled through differential or difference equations. On the other hand, temporal information included in the model has a relation to these differential or difference equations; more details can be found in the Section 5.1.

1.3.1. Timed System. We focus on the model in which time is taken into account (timed plant model). A timed plant represents a system where the time parameters are taken into account and play an important role. Naturally, the controller's commands influence timing of the controlled system. Therefore the diagnostic system will itself implement the time. First, possible ways of incorporating time into the model are overviewed. In this thesis, we restrict our focus to timed automata. This system description tool is suitable for modeling and analyzing in timed contexts.

Correctness of hybrid and real time systems (timed systems, in our terminology) not only depends on the logical order of events but also on their timing. This problematics is given intensive research activity in both communities: Control Theory and Computer Science.

1.3.2. ***System behavior.*** Model of the system behavior describes system trajectories, *i.e.* the time evolution of the system variables. Thus, three basic features are to be considered in modelling processes: variables, time and constraints. [**BKLS03**]

Variables. Quantitative variables have their values in a subset of real numbers, while qualitative variables take their values in a given finite set of symbols, which may be ordered or not. A symbol is often associated with each segment of the partition, *e.g.* small, medium, large.

Time. In discrete time systems, the classical time representation is a fictive time event "tick", which corresponds to clock passing discretely, as set of positive integers. By refining the computational methods in TDES, the most expressive time variable takes its values in the set of real numbers (dense time). A general conclusion of this comparison is that dense time is is strictly more expressive than discrete time[**Tri98**].

Constraints. Only controlled systems which verify Markov property are considered. This means that the whole past history of the system until time t can be summarized by the values of a set of variables at time t. The trajectories are such that the evolution of the state at time t only depends on the value of the state and input at time t. The evolution of the system is described through a set of constraints which apply to system variables. According to different descriptions of variables and of time, the constraints are modeled differently:

Continuous-variable: The evolution of continuous variables (whose values are in the set of real numbers) can be described in continuous or in discrete time. Continuous time descriptions basically use algebraic and differential equations and a transfer function (Laplace transform). Discrete time descriptions are useful when computer-controlled systems are considered, since the data are sampled at a constant rate by the system clock. They basically use algebraic and difference equations and transfer functions (based on z-transform).

DES: The evolution of qualitative (or symbolic) variables is best described by using discrete-event models such as automata, Petri nets or set of rules. Such models will be of interest in our research. Fuzzy variables (and models) can be used when it is wished to avoid abrupt transitions from one qualitative value to another one.

Hybrid: In many real life systems, continuous and qualitative variables co-exist. For example, an on-off temperature control system would be described by a continuous model. The time evolution of such a system is described not only by the temperature (which is a continuous variable) but also by heating mode (on/off), which is a qualitative one.

In this thesis, we restrict fault diagnosis to the DES model, precisely the TDES model. We describe the system by timed automaton with dense time consideration. Continuous or hybrid system can be regarded as quantised (TDES) and can also be described by timed automata; we consider this a promising. This approach is not the core of this thesis, but as there exists research effort in this area, such extension should be mentioned. Quantised systems will be briefly introduced now.

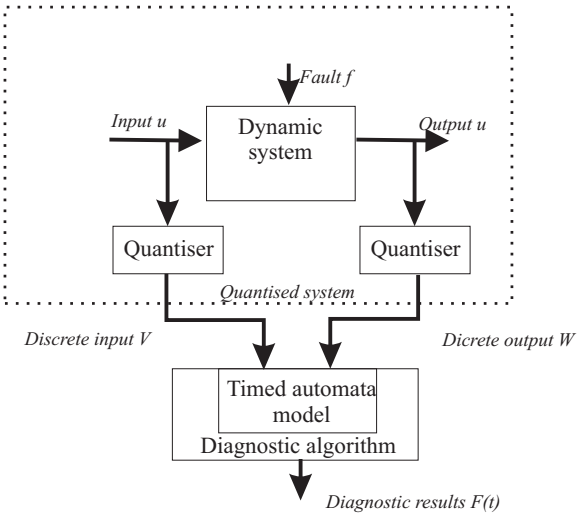
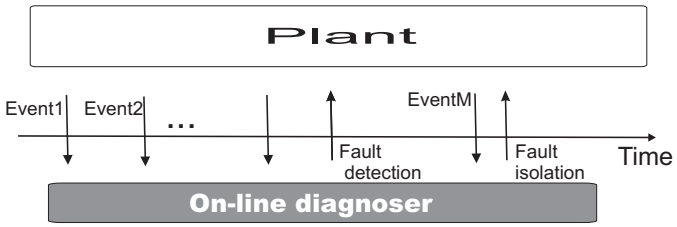
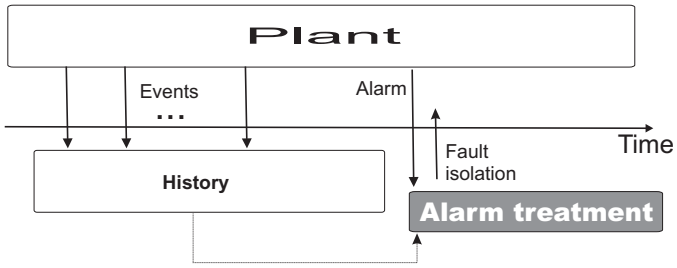


FIGURE 1.3.1. Quantised systems



(a) On-line diagnosis



(b) Off-line diagnosis

FIGURE 1.3.2. On-line and off-line diagnosis

1.3.3. Quantised system. The most important difference of quantised systems as opposed to classical continuous-variable systems is the fact that input and output signals of the dynamical system under consideration are not restricted to real values (Figure 1.3.1). Signals may be discrete values like a switch which has only binary values 1 ("on") or 0 ("off"); [**LSS01**]. The problematics of diagnosis in quantised system is treated in Chapter 5

1.3.4. On-line vs. off-line techniques. Diagnoser for on-line and off-line are different implemented according to the different purposes.

According to the different requirements for diagnostics system, we distinguish two techniques: On-line and Off-line. On-line diagnosis works by event observation in real time, see Figure 1.3.2.(a). The off-line is started with an external alarm signal, therefore we will call it "alarm treatment case", see Figure 1.3.2.(b). Each technique has its specifics. The designer of fault diagnosis system must consider the system requirements to choose one of them.

On-line detection and identification. The aim of on-line diagnosis is to generate a diagnostic statement from the observation to decide if there are faults in a process and also identify them.

Alarm Treatment[3]. Let us consider a fault detection system in the plant; we call it external alarm. This external alarm reports the fault to avoid a critical state and physical damage. The alarm denotes fault detection or isolation mechanisms. Alarm can be a physical sensor or some other means of fault detection (FDI methods, DX detection system, etc.). To avoid confusion of fault detection techniques realised by different families, we will consider an alarm fault detection system realised by an additional sensor. Alarm increases the reliability of system because the sensors failures can be also taken into account. Fault is detected by an alarm, the next phase fault isolation (identification) must be explored. The task of alarm treatment can be also in alarm confirmation or alarm masking.

In the two approaches, the dynamic evolution is exploited by using the time parameter. For the both cases it is same principle: looking for a coherent automaton run, although by different algorithms. The main differences are as follows (summarized in table 1):

- The model for alarm treatment is extended with alarm final states
- The implementation for the on-line mode works with real-time events, the alarm treatment employs a history of stored events.
- The algorithm for the on-line case is a state estimation and the off-line cases are based on the research of coherent paths by backward time analysis.

[3]We will use term "alarm treatment" instead of off-line diagnosis or off-line isolation. We understand that alarm treatment belongs to off-line case.

	On-line	Alarm treatment
events occurrence	On-line observing	Store to history
aim	Detection and isolation	Isolation
alarm produces	diagnoser	sensor or other FD system
algorithm	State estimation	Backward time analysis
activation	initial state synchronisation	by alarm signal
model	normal, faulty	with alarm states
method	by observation	by verification

TABLE 1. On-line and off-line differences

- The on-line algorithm works on-line from a synchronized initial state. The backward time algorithm is executed after the reception of the alarm signal.

1.4. Problem formulation

A timed system represents a system of which behaviour depends on timing constraints. This definition being too general, we restrict our attention mainly to timed systems for which timing constraints are critical, that is, the correctness of the system depends on them. Examples of such systems include traffic controllers, chemical-reaction controllers, real-time operating systems and so on.

Failure of a timed system can have catastrophic consequences, therefore it is crucial to ensure its correctness and reliability of its fault diagnosis mechanisms.

Our subject-matter is fault diagnosis of timed system by timed automata. We restrict our discussion to passive, process diagnosis: *i.e.* the system under supervision is operational and the fault detection system does not use test inputs for diagnosing failures but relies on incoming signals.

Following definitions of problems express the different treated problems. In this thesis, we try to bring an answer to various questions concerning the fault diagnosis[4].

PROBLEM 1.4.1. *Modeling formalism:* how to model the normal (faultless) and faulty behaviour of a given timed system.

This topic is treated in two chapters: We propose to model the system by timed automaton that is a suitable tool for modeling and analysing the

[4]For each problem, typical questions about the problem are formulated.

behaviour of a timed system. Timed automaton integrates quantitative and qualitative parameters, which makes the modeling powerful and expressive. The timed automaton phenomenon will be introduced below, along with relevant formal definitions.

Later, in Section 4.2, a systematic manner for modeling complex and large systems is described. A system can be decomposed into descriptions of faultless and faulty behaviour. Fault integration into the faultless model can be easily extended. The composition (parallel product) of timed automaton components will create *dynamic global model* $\mathcal{G}$. This composition creates language of all possible timed traces of the system (infinite). This model is useful for diagnoser verification.

PROBLEM 1.4.2. *Time-diagnosability:* How does a system behave in the presence of a fault? How a fault is propagated? Does exist any temporal delay between the occurrence-detection and isolation?

We treat diagnosability as detectability and isolability of the system. Detectability is the ability to detect a considered fault. Not every system is detectable and also isolable within finite delay. A diagnostic system that can detect faults but not distinguish them is detectable and non-isolable. This part is treated in Section 3.2.

PROBLEM 1.4.3. *Diagnoser building:* how to construct an observer (called $\mathcal{D}$-diagnoser) for a given system. Diagnoser analyses the observed behaviour to decide upon presence of a fault and also identifies the fault.

The diagnoser will be built using knowledge of fault propagation. The fault will be detected by changing the timing or changing the event sequence. Fault propagated by changed timing can cause invariant violation. To avoid deadlock of the system, the diagnoser will be extended with faulty transitions. Diagnoser $\mathcal{D}$ is supposed in the form of a timed automaton. Building of $\mathcal{D}$-Diagnoser is motivated by the fact, that diagnoser in the automaton form is easily implemented in the industrial system like PLC. The algorithm is detailed in Section 3.3.

PROBLEM 1.4.4. *Model checker for diagnosis:* We treat the problem how use a model checker for the purposes of fault diagnosis.

In *model checker concept*, diagnostic mechanism uses knowledge of the global dynamic model. $\mathcal{G}$ model is calculated on-line as a composition

of timed automata components. The diagnostic mechanism is checking of respective property in the dynamic global model $\mathcal{G}$. Our property to check is reachability of faulty state. Realisation of this concept has the difficulty of memory and time consumption for the verification task and also with model-checker implementation. Hence, illustration of alarm treatment case is proposed.

To verify some property for complex system (verification of reachability of faulty state) takes time (approx. seconds, minutes) which was a constraint for real-time implementation. According to the promising reduction and state-space representation, actual researchers interest in computer science community try to make this approach attractive for the on-line diagnosis problem (See UPPAAL TRON in [**LMN04b, LMN04a**]).

PROBLEM 1.4.5. *Check of the diagnoser correctness:* For a given system model $\mathcal{G}$ and diagnoser $\mathcal{D}$, check whether diagnoser's results of detection and identification are correct for all faulty runs of global model (runs that contain a faulty event f_i). Practically, it means that the diagnoser announces the fault if and only if the fault has already occurred. The safety property is following implication: The presence of the fault implies the diagnoser's faulty result. The problem is viewed as a model-checking problem. We check a common property of global system model $\mathcal{G}$ and $\mathcal{D}$-diagnoser. If the results of verification are negative, the rebuilding of the diagnoser or system must be done. The formal verification of correctness leads to avoiding incorrect diagnosis results. The result of the verification is a verdict whether the diagnoser is well-designed or not. The verification is detailed in Section 4.4.

Quantised system: Application of a diagnosis approach on TDES model for a given continuous (quantised) system.

This problem is treated only partially in this thesis. Our aim is to show that existing research on modeling and diagnosis of continuous system by TA model is promising. We demonstrate the diagnosis on a real application of heat exchanger system. Chapter 5 deals with this problematics.

1.5. The state of the art

Fault diagnosis has been studied by different communities and in different contexts, *e.g.* see [**SSL+95, BLPM00, Fra87, Ise84**] and citations therein. First, we mention the fault diagnosis in discrete event system,

where the diagnoser observes only an event sequence. Then, we restrict our focus to fault diagnosis of timed systems modeled by TDES tool, where the sequence of events and time are taken into account.

1.5.1. Fault diagnosis in DES. The fault diagnosis problem is studied in different domains; therefore, terminology has not stabilized. The research is still very lively, which is also due to increasing requirements on the safety of systems.

The task of fault diagnosis is to generate a diagnostic statement from the observation or from the history in order to decide if there are faults in a process and also to identify them.

The problem of fault diagnosis was studied first *without time* aspect; the diagnostic result was searched by analyzing a logical order of events (events sequence). In the context of discrete event system, the problem of fault diagnosis has been subject of intensive research since 1970s and there are several good books about this subject. One of the key contributions to fault diagnosis in DES are [**SSL$^+$95, SSL$^+$96**], where notions like diagnoser and diagnosability were introduced in the systematic manner. System behavior is modeled as a regular language and it is represented by a finite state machine. The global model denotes synchronous product of their components. The systematic way for detection and isolation of failure events use diagnosers: the diagnoser is normally built from the FSM model of the system. Usually, the diagnoser serves two purposes: i) on-line detection and isolation of failures and ii) off-line verification of the diagnosability properties of the system. More details about classical diagnoser building are given in the Section 1.5.1.1. The authors introduce the notion of Diagnosability as a possibility to detect the fault with a finite delay. Our effort is directed at extending this base approach, in which time aspect is not included to create a diagnoser, into an approach where time parameters are taken into account.

REMARK 1.5.1. Fault-diagnosis is also related to the controller-synthesis problem, introduced for DES in [**RW87**]. The problem has been studied for timed and hybrid models as well (see *e.g.* [**BGBDS04, Tha00**]).

1.5.1.1. *Classical diagnoser (no temporal constraints).* Diagnoser approach is described in [**SSL$^+$95**]. The component based modeling describes

the system model by N finite state automata $G_i = (Q_i, \sum_i, \delta_i, x_{0i})$, where δ_i is the transition function, with no temporal constraints.

In the first step, the model of whole system $G = (Q, \sum, \delta, x_0)$ is built using a composition operation on automata. Some of the events in $\sum$ are observable ($\sum_0$), their occurrence can be observed. The rest denotes set of unobservable events, $\sum_f \subset \sum_{uo}$.

The hypothesis is that G_i has no unobservable cycle (*i.e.* a cycle with only unobservable events) and that the failures are permanent.

The aim of the diagnosis is fault detection and identification based on the observed events. In order to solve this problem, the state model is directly converted into a diagnoser.

The diagnoser, as defined in [**SSL$^+$95**], is a deterministic finite state machine $D = (Q_d, \sum_d, \delta_d, x_{0d})$ where the set of events $\sum_d$ is a set of observable events of the system ($\sum_d = \sum_o$). Each state of the diagnoser contains possible states of the system and additionally, for each state is associated a label. It includes all the faults that have to occur to reach this state. Labels denote diagnostic results; they have following meaning:

- $l = \{N\}$: no failure has occurred
- $l = \{A\}$: some fault may or may not have occurred. None can be identified (A for ambiguous).
- $l = \{F_{i_1}, F_{i_2}, \ldots, F_{i_k}\}$: at least one fault from the set of fault has occurred.

Building the transition function δ_d is described in [**SSL$^+$96**]. It is a recursive process. The classical diagnoser concept is based on the finite state machine (automaton), where the transition function has no temporal constraints.

1.5.2. Fault diagnosis in Timed Discrete-Event System (TDES).

1.5.2.1. *Discrete time.* [**Zad99**] deals with a timed model of DES also called timed discrete-event system (TDES). The author assumes that the plant under control can be modeled as TDES. A TDES is a finite-state automaton containing an event which is tick of global clock in its event set. The tick is assumed to be observable. TDES are used in supervisory control systems, TDES models are obtained from activity transition graphs (ATG) of the subsystems. Design of a fault detection system has, in the worst case, exponential complexity. A model reduction scheme with polynomial time complexity is introduced to reduce the computational complexity of

the design. The standard diagnoser provides updates of system's condition after generation of any new output symbol or any clock tick. However, the number of states corresponding to this approach will be very large due to the incorporation of timing information.

[**ZKW99**] proposes an alternative approach in which the process of updating the estimation of system's condition is performed only when a new output symbol is generated. The update process is based on output symbols generated and on the number of clock ticks occurring between then. No updating at clock tick is required in this method.

1.5.2.2. *Dense time.* The problem of fault diagnosis in the context of dense-time automata has been also studied by Tripakis in [**Tri02**]. Tripakis shows i) how to analyse the timed automaton to check whether a given timed plant is diagnosable or not ii) how to construct a diagnoser as an on-line algorithm. The plant is supposed to be diagnosable if any pair of faulty/non-faulty behaviours can be distinguished by their projections to observable behaviours. Diagnoser, in this approach, is not a finite state machine (timed automaton) built from the system model, but it is regarded as an algorithm which treats the given system model (timed automaton).

The algorithm for checking the diagnosability works as follows: First, a special parallel product of automaton $\mathcal{A}$ is built. This product generates all pairs of runs of $\mathcal{A}$ that yield the same observations, yet one is faulty while the other one is not. Then, it is shown that $\mathcal{A}$ is diagnosable iff the product cannot generate a non-zeno pair which contradicts the necessary condition for Δ-diagnosability.

1.5.2.3. *Template languages approach.* [**HC94**] uses a timed automaton model without clocks, but where time interval is associated with discrete states. They propose template monitoring as a technique for distributed fault diagnosis, where templates are sets of constraints on the occurrence time of events.

[**PH00**] continues in a framework for modeling discrete event processes. This framework, called condition templates, allows modeling of processes in which both single-instance and multiple-instance behaviors are exhibited concurrently. A single-instance behavior corresponds to a trace from a single finite-state process, and a multiple-instance behavior corresponds to the timed interleavings of an unspecified number of identical processes operating at the same time. The template framework allows for modeling of correct

operation for systems consisting of concurrent mixtures of both single and multiple-instance behaviors. This representation is used in on-line fault monitoring for confirming the correct operation of a system. It is shown that templates are able to model timed languages corresponding to single and multiple-instance behaviors and combinations thereof. Templates can thus represent languages that could not be represented or monitored using timed automata alone.

1.5.2.4. *Extended diagnoser.* [**RC02**] treated the problem of timed extension for the case of alarm treatment. The theory is adjusted according to the specific needs of telecommunication networks. This work is based on diagnoser approach proposed by Sampath [**SSL+95**]. The network adapted to simulate faults is modeled, thus is transformed into a finite state automaton (diagnoser), in order to analyze the stream of alarms. This paper extends classical approach by communicating finite state machines. A major difficulty is the huge size of the global model of the system. To solve this problem, advantage has been taken to the hierarchical structure of the network. This approach was implemented and experimented on the application of switching telecommunication network.

1.5.3. Timed automata framework. The timed automata theory was introduced in [**AD94**]. [**Tri98**] deals, in his thesis, with the formal analysis of timed system. The timed formalism using timed automata is described. Analysis techniques of timed automata verification is proposed. Thus the controller synthesis problem is described and treated as parallel composition of controllable timed automata. [**BY04**] proposes the global overview of timed automata semantics.

1.5.3.1. *Model-checking approach.* [**BGBDS04**] deals with the correctness checking of the safety functioning of a PLC program in model-checker UPPAAL. Authors take an evaporator batch process as an illustrative example.

Variations of this example has been used in different papers to illustrate different purposes. [**KS98**] deals with the safety analysis for this example. They check a control code to comply with the specification and analyse the dangerous states. The model-checking is provided by the tool UPPAAL [**Luk99**].

The model-checker approach is more interesting due to its easier usage. To verify some property for a complex system (verification of reachability of a faulty state) takes an amount of time (approx. seconds, minutes), which constraints its use for real-time implementations. Because of the promising reduction and state-space representation, present researchers involved in the computer science community try to make this approach attractive for on-line diagnosis problem ([**sUT, LMN04a**]).

1.5.4. Quantised system. [**Phi01**] deals with the problematics of translation of continuous dynamics to discrete-event system. Author also deals with the problem of discretization of the continuous dynamics, controller synthesis and he also touches the problematics of fault detection and isolation by DES models.

[**HGS04**] treats the translation of the continuous dynamics into a TDES formalism. The continuous model is first approximated using intervals and then translated into a timed automaton by comparison of thresholds.

Lunze proposes a diagnostic method based on automata. In majority of publications, a stochastic automaton is used ([**LN03, BKLS03, LS01**]). Quantised system approach is shown on the example of two tank system in [**LS01**], where the modeling and diagnostic task is illustrated in detail. The extension for timed system is described in [**SFL03**], where solution of the identification task (to obtain the timed automaton model) is proposed. Diagnostic approach works on a set of time automaton models, where each fault is described by its temporal and sequential behaviour. Diagnostic task is consistency-based, a coherent automata is searched according to input/output observation ([**HLS04**]).

1.6. Aims of the dissertation

1.6.1. Our approach. Our aim is provide a global overview in fault diagnosis by timed automata. TA representation of dynamic systems by timed automata is chosen for its good modeling and analysis behaviour. We consider a system, where a fault changes its dynamics, *i.e.* the temporal behavior. In our point of view, dynamics of a system is translated to the timed parameters of TA. These parameters, along with the events, describe expressively the time behavior in normal situation and in the presence of fault as well.

We concentrate on passive diagnosis; *i.e.* the system under supervision is operational and the fault detection system does not use test inputs for diagnosing failures but relies on incoming signals.

Some examples of the internal faults are (Illustration on *e.g.* tank system):

- Process fault: Change the dynamics behaviour of the process. Such changes cannot be tolerated by the control law. Example: The tank is leaking.
- Actuator fault: Disturb the possibilities to influence the plant. They make the plant partially uncontrollable. Example: The input valve is stuck open.
- Sensor fault: Break the information link between the plant and the controller. They make the plant partially unobservable.

We consider a "drastic" failures, such as "valve stuck-closed" and "sensor short-circuited". Other types of partial faults such as drift in sensor or small changes in dynamics of actuator are not considered here, because they are difficult to model within the TDES framework.

Our diagnostic approach extends fault diagnosis approaches by Sampath [**SSL+95, SSL+96**] and Tripakis [**Tri02**] by time consideration. In the time model, a dense time framework is more convenient. Because "fictive clock tick" has an exponential complexity, therefore it does not represent a suitable method for fault diagnosis for complex systems. Dense time is represented by real value clock [**Tri02**]. We cannot use results from existing result of time diagnosis in [**RC02**]. There, the solution proposed is influenced by concrete application from telecommunication, *i.e.* the diagnoser is built according to a hierarchical structure, which is not identical to the control system structure.

The aim is to build a diagnostic system (called $\mathcal{G}$-diagnoser) for a given system. Diagnoser is build by determination of faulty transitions and must include necessary information about the dynamics of fault behaviour (time aspect). Diagnostic system as TA has advantage in easy implementation to supervision system.

On the other hand, we introduce also a model checker concept in fault diagnosis problem. Model checker has advantage in representation of TA components (dynamic global model $\mathcal{G}$), where the composition is calculated on-line. With model checker, we treat an alarm treatment diagnosis to show

the verification logic for diagnosis purposes. Model checker help us also with formal verification of built $\mathcal{D}$-diagnoser.

In model checker context, Global dynamic model $\mathcal{G}$ can be viewed as a composition of product component. State of $\mathcal{G}$ is vector of states of all components and vector of clocks. Practically, the construction of global dynamic model in the form of one $\mathcal{G}$-automaton is not needed. The reason is simple: global dynamic model represented as one timed automata is very huge. Some alternative ways of global dynamic model (viewed as parallel composition) representation exist: Difference Bound Matrices (DBM), Compact Data Structure, approximation methods. Later in this thesis, global dynamic model is calculated "virtually" by model checker software UPPAAL. On the other hand, explanation of time-diagnosability is better seen on simple automaton then some alternative mentioned representation.

1.6.2. Thesis aims. According to the described problematics, this thesis has the following aims:

(1) Modeling formalism based on the timed automata.
(2) Fault diagnosis mechanism described on timed automata (time-diagnosability)
(3) Method for diagnoser construction ($\mathcal{D}$-Diagnoser) based on the faulty transition determination (for fault detection and also for fault isolation phase).
(4) Examine a model checker for fault diagnosis purposes
(5) Verification (model-checking) of a diagnoser, comparing diagnoser and global model to verify the diagnoser correctness.
(6) Brief discussion of methods of extension for quantised system.

in short, this thesis offers a formal description of fault diagnosis problem. We try to make a bridge between the control system community and the computer science approach.

1.7. Thesis outline

The thesis is organized as follows: chapter 1 has offered an introduction to our problematics. The fault diagnosis problem in the context of supervision system was presented. Fault diagnosis takes into account the fault detection step and the fault identification (isolation) one. The fault diagnosis domain contains a large number of methods and different approaches.

According to different modeling classes, different fault diagnosis methods were overviewed. Then we focus on the formulation of treated problems with the definition of thesis objectives.

Chapter 2 proposes modeling of the system by timed automata. Timed automaton is introduced informally as well as in a formal way. Frequently used concepts are described. For quantitative and qualitative description, the component based approach is recommended. Modeling steps introduce a term "dynamic global model", shortly $\mathcal{G}$. Finally, we treat an illustrative example to show the practical results.

Chapter 3 presents fundamental diagnosis mechanisms. The notion of time diagnosability is introduced in the sense of fault detectability and fault isolability. Parameter of detectability express an ability to distinguish between faulty and faultless behaviour. Time parameter δ denotes the maximal delay between fault occurrence and fault detection. Time parameter θ is the maximal delay between fault occurrence and fault isolation. On-line algorithms are illustrated by the fault detection and isolation.

The second part of chapter 3 is dedicated to the problem of diagnoser building. Diagnoser $\mathcal{D}$ is resulting from fault diagnosis system, in the form of finite state machine (timed automaton). Diagnoser is built by determination of faulty transitions, which are viewed as faulty signatures, including temporal conditions. At the end of Chapter 3, a trivial example for time-diagnosability is treated with an example of diagnoser building.

In chapter 4, advantages of model checker are illustrated on:

(1) Complex system model based on components
(2) Diagnostic mechanism shown on alarm treatment case
(3) Formal verification of $\mathcal{D}$-Diagnoser

Complex system modeling make use of the feature that the system is described by several components. Implementing a fault into model could be easily done. Parallel composition of all TA components, including a controller, creates state space of all possibles evolutions represented by dynamic global model $\mathcal{G}$. This model is calculated in model-checker on-line; we use the model checker UPPAAL.

Usage for diagnosis purposes is shown on alarm treatment case. We introduce a algorithm "backward time analysis" which search a cause of the fault. By this analysis, the alarm signal is treated by examining the timed sequence to isolate the fault. In model checker, we use Backward

time analysis in the sense of verification of fault occurrence possibility. This approach has certain limitation which are discussed also there.

The idea of formal verification of diagnoser correctness is realised. We compare the global model of the system $\mathcal{G}$ with the built diagnoser $\mathcal{D}$ to verify the diagnosis results. We define a term well designed diagnoser for diagnoser where the correctness is verified. Correctness express an ability of the diagnoser to give correct diagnostic results.

In Chapter 5, reader can find the promising extension continuous system. There, quantised systems are viewed as TDES at the higher level of system abstraction. Timed automata model is obtained by discretization of known trajectories of continuous system. We have no aspirations at describing the problematics in detail, on the other hand, research has promising results in this domain and we would like to present the outlines in wider context. Advantages and limitations of this method are illustrated on the heat exchanger application.

Finally, in Chapter 6, we present summary of our results and discuss directions for future research.

CHAPTER 2

Modelling formalism

This chapter gives an overview of modeling formalisms oriented at timed framework. We describe timed systems using timed automata (TA). This chapter presents the basic formalism of timed automata. The next chapters will use the described timed automata semantics. Timed automaton is finite-state automaton extended with a finite number of real-valued clocks. A TA alternates between two modes of execution, letting time pass continuously, then taking a step changing its discrete state. TA represents an expressive model with consideration for both qualitative and quantitative parameters. We prefer a dense-time TA (instead of discrete-time), because it is a more natural model for physical processes operating over continuous time. Finally in this chapter, the various developments for modeling and the analysis will be illustrated on a simple didactic example.

Ce chapitre donne une vue d'ensemble des formalismes de modélisation dans le cadre temporel. Les systèmes temporisés sont modélisés par des automates temporisés (AT). L'AT est automate à état fini avec un nombre fini d'horloges à valeurs réelles. Un AT représente le séquencement discret des opérations à exécuter en tenant compte de la variable continue temps qui autorise le changement d'état discret. Cette alternance temps continu - états discrets, nous amène à définir le terme "dense time TA" pour désigner le modèle "automate temporisé à temps discret" Enfin, les différents développements pour la modélisation et l'analyse seront illustrés dans ce chapitre, sur un exemple didactique simple.

2.1. Discrete-event system framework

The fault diagnosis problem in DES represents essentially the model-based approach. That is to say, an algorithm for fault detection and isolation will analyse and compare the model with the observed behaviour. Note that

in the context of discrete event system, the problem of fault diagnosis has been subject of intensive research since 1970s and there are several good books about this subject. [**SSL$^+$95, SSL$^+$96**] can be considered as the key contributions to fault diagnosis in DES. Research of fault diagnosis is also related to the research of modeling formalism.

For discrete-event systems, there was no notion of time and the time set T is only used for the ordering of the events. By evolution of formal mathematical method, discrete-event system was extended for consideration of temporal information, resulting in a class called Timed Discrete Event System (TDES).

Before we restrict our discussion to timed systems, let us introduce finite automata and the basic notions.

2.1.1. Finite automata. The formal language theory offers the possibility to model system behavior. A formalism of finite-state machine (FSM, finite state machine) has been studied for many years and provides the qualitative way to model a system. Engineers have found that the use of finite-state automata and corresponding state transition diagrams make design and diagnostics of complex systems easier.

An automaton is a mathematical model for a *finite state machine* (FSM). A FSM is a machine that, given an input, jumps through a series of states according to a transition function (which can be expressed as a table). In the common "Mealy" variety of FSMs, this transition function tells the automaton which is the next state to go to, given a current state and a current symbol. The input is read symbol by symbol, until it is consumed completely (think of it as a tape with a word written on it, that is read by reading head of the automaton; the head moves forward over the tape, reading one symbol at a time). Once the input is depleted, the automaton is said to have stopped. Depending on the state in which the automaton stops, it's said that the automaton either accepts or rejects the input. If it landed in "accept" state, then the automaton accepts the word. If, on the other hand, it lands on a "non-accept" state, the word is rejected. The set of all the words accepted by an automaton is called the *language* accepted by the automaton.

The following terms are used very often in the automata theory:

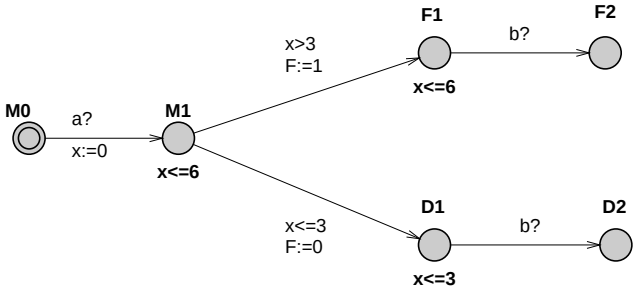


FIGURE 2.2.1. Time extension model by timed automata

Symbol: Analogy of a single letter, although it need not to be a letter. It may be any symbol, as long as it is a single token (no words here yet), and can be distinguished from other symbols.

Word: A finite string formed by the concatenation of a number of symbols.

Alphabet: A finite set of symbols.

Language: A set of words, formed by symbols in a given alphabet. May or may not be infinite.

2.2. Timed discrete-event system

A time extension is useful, since it allows to model plants with time behavior. For example, "**a** followed by **b** with a delay of 3 time units". It also allows for diagnoser to base their decision not only on the sequences of observed events, but also on the *time* delays between these events [**Tri02**].

In the Figure 2.2.1, two events are expected: **a** and **b**, in this order. If the delay between **a** and **b** is greater than 3 time units, it corresponds with the faulty behaviour. In the non-faulty behaviour, the delay is at most 3_{tu}.

All information regarding the system operation must be determined from sequencing and timing events. This idea is the same for fault diagnosis algorithms as well.

2.2.1. Semantic model: Dense versus Discrete time. Time is considered and modeled in different ways. This different implementations are related to the evolution of modeling tools. Dense time is strictly more expressive than discrete time. Perhaps the most important feature of dense time is the fact that it abstracts from a specific time quantum, since it can model arbitrarily small delays. This has the advantage that the system is

independent of the time quantum. More formal results can be found in [**Tri98**] and in citations therein.

Discrete time model. Between a number of different discrete-time model, we illustrate two following models:

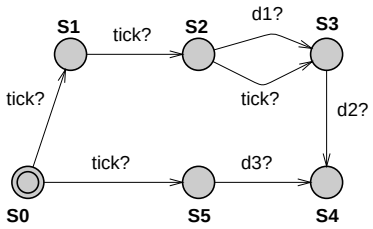
(1) *Silent event:* This model requires the time sequence to be a monotonically increasing sequence of integers; it can be easily transformed into a ordinary formal language. Each timed trace can be expanded into a trace where times increase by exactly one at each step, by inserting special "silent" event as many times as necessary between events in the original trace. Discrete time behaviours can be manipulated using ordinary finite state machine.[**SSL$^+$95, SSL$^+$96**]

(2) *Observable tick event:* Model is similar to the discrete time model, except that it only requires the sequence of integer times to be non-decreasing. The interpretation of a timed execution trace in this model is that events occur in the specified order at real-valued times, but only the (integer) readings of the actual times with the respect to a digital clock formal language. First, add to the new set of events a new one, called tick. [**Zad99, ZKW99**]

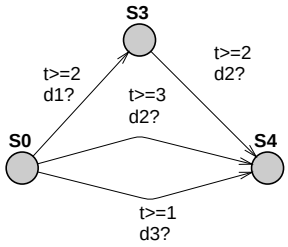
Dense time. The timed automata of Alur and Dill [**AD94**] are a popular model for time-dependent systems that extend classical finite state machines with real-time clocks. These clocks can record the passage of a time in the states, and can be used to guard the occurrence of transition. A timed automaton generates timed sequences of events. The fault diagnosis problem thus detects faulty behaviours from a given timed sequence of observable events. Authors consider both transition structures – deterministic and non-deterministic.

2.2.2. TDES descriptions languages. We restrict our study only to Timed Automata phenomenon. Examples of different qualitative model with incorporated timing constraints include timed Petri nets [**Sav01**], timed transition systems [**HC94, PH00**].

Hybrid automata are a generalization of the TA model with real variables having more general evolution laws, defined by differential equations. The fault detection and isolation problematics is treated *e.g.* [**CMS03, CSM04**].



(a) Discrete time



(b) Dense time

FIGURE 2.2.2. Discrete and dense time consideration

2.3. Timed automata

Timed automata is a theory for modeling and verification of real time systems. A timed automata [**AD94, BY04**] is a essentially a finite automaton (FSM) extended with real-valued variables. Such an automaton may be considered as an abstract model of timed systems. This expressive modeling tool offers the possibilities of model analysis like verification, controller synthesis and also fault detection and isolation.

In the original theory of timed automata [**AD90, AD94**], a timed automaton is a finite state Büchi automaton extended with a set of real-valued variables modeling clocks. Constraints on the clock variables are used to restrict the behaviour of timed automaton, and Büchi conditions are used to enforce progress properties. A simplified version, namely Timed Safety Automata, is introduced in [**HNJY94**] to specify progress properties using local invariant conditions. Due to its simplicity, Timed Safety automata has been adopted in several verification tools for timed automata *e.g.* UPPAAL [UPPAAL] and KRONOS [**Yov93, Yov97**]. In this thesis, we shall focus on the Timed Safety Automata and refer to them as Timed automata or simply automata.

By timed automata, the system is described in qualitative and quantitative manner. See example on the Fig.2.3.1, the qualitative parameters

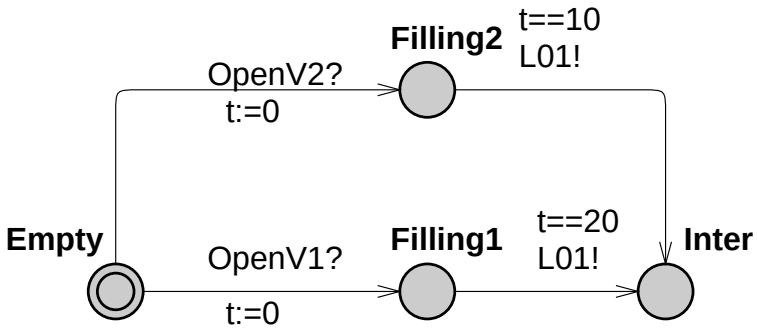


FIGURE 2.3.1. TA model includes qualitative and quantitative parameters

represents the sequence of events whereas quantitative pertains to temporal parameters.

The problem of automata analysis is considerably more difficult in the timed case than in the discrete case: in the discrete case, one deals with classical regular languages which have robust closure properties. Thus one can obtain a diagnoser by determining the model of the plant. In the timed setting, the problem is compounded by the fact that timed automata are a very expressive formalism [**AD94**].

2.3.1. Formal description. Assume a finite set of real-valued variables $\mathcal{C}$ ranged over by x, y etc. standing for clocks and a finite alphabet $\sum$ ranged over by a, b etc. standing for actions.

Clock constraints. A clock constraint is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x - y \sim n$, for $x, y \in \mathcal{C}$, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. Clock constraints will be used as guards for timed automata. We use $\mathcal{B}(\mathcal{C})$ as the set of clock constraints, ranged over by g.

We define the timed automata as follows ([**AD94, Tri02, BY04**])

DEFINITION 2.3.1. Timed automaton is a tuple $\mathcal{A} = (Q, q_0, \sum, E, I)$ where

Q is finite set of discrete states (locations)

$q_0 \in Q$ is the initial discrete state

$\sum$ is set of events

$E \subseteq Q \times \mathcal{B}(\mathcal{C}) \times \sum \times 2^C \times Q$ is set of timed edges

$I : Q \rightarrow \mathcal{B}(\mathcal{C})$ is a function associated with each discrete state Q. System can remain in the same location as long as the invariant is true. Invariants I are downwards closed, in the form: $x \leq n$, where n is natural number.

We shall write $q \overset{g,a,r}{\rightarrow} q'$ when $\langle q, g, a, r, q' \rangle \in E$.

2.3.2. Timed automata syntax and semantics. The terminology about timed automata is quite extensive. Let us define a few following concepts to simplify later text:

The semantics of timed automata is defined as a transition system where a state or configuration consists of the location and the current values of the clocks. There are two types of transitions between states. The automaton may either be delayed for some time (a delay transition), or follows an enabled edge (an action [event] transition).

States. A state is a pair (q, v), where $q \in Q$ is a location, and $v \in I(q)$ is a validation satisfying the invariant of q.

Timed transitions. The semantics of timed automata is defined as a transition system where a state or configuration consists in the current values of the clocks. There are two types of transitions between the states. The automaton may either be delayed for some time (a delayed transition), or follow a enabled edge (an action transition).

The semantics of a timed automata is a transition system (also known as a timed transition system) where states are pairs (q, v) and transitions are defined by the rules:

- $(q, v) \overset{\delta}{\rightarrow} (q, v + \delta)$ if $v \in I(q)$ and $(v + \delta) \in I(q)$ for a non-negative read $\delta \in (R)_+$
- $(q, v) \overset{e}{\rightarrow} (q\prime, v\prime)$ if $q \overset{g,e,r}{\rightarrow} q\prime$, $v \in g$,$v\prime = [r \mapsto 0]v$, $v' \in I(q')$

where $v \in g$ means that the clock values denoted by v satisfying the guard g, $v' = [r \mapsto 0]v$ means a clock assignment that maps all clocks in r to 0 (reset) and agree with v for other clock in $\mathcal{C}$.

Path. A finite or infinite path is finite or infinite sequence $q_1 \rightarrow q_2 \rightarrow \ldots$ where the q_1, q_2 are called the visited states, $q_1, q_2 \in Q$.

Timed path. A timed path over $\mathcal{A}$ if set of events $\sum$ is a finite sequence $\gamma_1, \gamma_2, \ldots$, where each γ is either an event in $\sum$ or a delay δ in Q. We require that between any two events in ρ there is exactly one delay (possibly 0). For example, if $\mathbf{a}$ and $\mathbf{b}$ are events and $\mathbf{a}, 0, \mathbf{b}, 3, \mathbf{c}$ is valid timed sequence (also called time word), while $\mathbf{a}, \mathbf{b}$ and $\mathbf{a}, 1, 2, \mathbf{b}$ are not. [**Tri02**]

Timed Action. A timed action is a pair (t, e), where $e \in \sum$ is en event (action) taken by an automaton $\mathcal{A}$ after $t \in \mathbb{R}_+$ time units since $\mathcal{A}$ has been started. The absolute time t is called a time-stamp of the event e.

Timed trace (trajectory). A timed trace is a sequence of timed observed event $\xi = (t_1, e_1)(t_2, e_2)(t_3, e_3)$. The set of visited states is called trajectory[**CGLP03**].

Timed word is accepted (or rejected). Time word is the sequence in which a real valued time of occurrence is associated with each symbol. A timed word is accepted by an automaton $\mathcal{A}$, if the automaton is landed in an accept state. If, on the other hand, it lands on a non-accept state (deadlock), the time word is rejected.

Projection (operator P). Given a (finite or infinite) timed sequence ρ and a set of events $\Sigma \subset \Sigma'$, $P(\rho, \Sigma'))$ is timed sequence obtained by erasing from ρ all events in Σ' and summing the delays between successive events in the resulting sequence. For example, if $\rho = 1, a, 4, b, 1, c, 0, d, 3, e$, then $\mathrm{P}(p, b, d) = 1, a, 5, c, 3, e$. Note that, in the definition of $\mathrm{P}(\rho, \Sigma')$, Σ' is the set of events to be erased. Also notice that $\mathrm{time}(\rho) = \mathrm{time}(\mathrm{P}(\rho, \Sigma'))$, for any ρ and Σ'.

Trajectory. A sequence of events is called trajectory of automaton $\mathcal{A}$ iff the sequence is enabled in the initial state q_0 of the automaton $\mathcal{A}$.

Time of path. If ρ is finite timed sequence, $\mathrm{time}(\rho)$ expresses the sum of delays in ρ. If ρ is infinite, then $\mathrm{time}(\rho)$ expresses the limit of the sum (possible ∞). We say that ρ is *non-zeno* if $\mathrm{time}(\rho) = \infty$. Note that a non-zeno timed sequence is necessarily infinite, although it might contain only a finite number of events

Run. A run of timed automata $\mathcal{A}$ with initial state q_0 over a timed trace $\xi = (t_1, e_1)(t_2, e_2)(t_3, e_3)$, is a sequence of transitions:
$(q_0, v_0) \xrightarrow{\delta_1} \xrightarrow{e_1} (q_1, v_1) \xrightarrow{\delta_2} \xrightarrow{e_2} (q_2, v_2) \xrightarrow{\delta_3} \xrightarrow{e_3} (q_3, v_3) \ldots$satisfying the condition $time(i) = time(i-1) + \delta_i$, for all $i \geq 1$.

The timed language $L(\mathcal{A})$ is the set of all timed traces ξ for which there exists a run of $\mathcal{A}$ over ξ.

A run is a path in the automaton graph of $\mathcal{A}$ where discrete transitions are taken infinitely often and consecutive time transitions are concatenated.

Reachable state . A state s is reachable if there exists a finite run $s_0 \xrightarrow{\delta} \xrightarrow{e_0} \ldots \xrightarrow{\delta} \xrightarrow{e_l} s_l \xrightarrow{\delta} s$, for $l \geq 0$. Let $\mathrm{Reach}(A)$ to be set of all reachable states of $\mathcal{A}$.

Deadlock. Deadlocks are states violating the discrete-progress requirements. A state s of $\mathcal{A}$ is deadlock, if there is no delay $\delta \in \mathbb{R}$ and event edge $e \in E$ such that $s \xrightarrow{\delta} \xrightarrow{e_l} s'$.

Well-timed TA. $\mathcal{A}$ is well-timed if for all $s \in \mathtt{Reach}_A$, where there is a non-zeno run of $\mathcal{A}$ starting at s.

Parallel composition. A network of timed automata $\mathcal{A} = (\mathcal{A}_1 \parallel \ldots \parallel \mathcal{A}_n)$ is a collection of concurrent timed automata $\mathcal{A}_i$ composed by a parallel composition. Furthermore, we give a semantic meaning to parallely composed networks. A state of a network is modeled by a configuration $(\bar{q}, \bar{v})$. The first component is a location vector $\bar{q} = (q_1, \ldots, q_n)$, where q_i is location of automaton $\mathcal{A}_i$.

The parallel composition will be useful for dynamic global model construction. A system is divided in parts, therefore, it is convenient to be able to describe system compositionally (to this topic is dedicated Section 4.2). That is, as a set of components, their composition creates a global model. The components are executed in parallel and communicate in a certain way. Our model of parallelism is based on synchronous passage of time for all components and it is interleaved by discrete actions. Communication is modeled via synchronisation. Because this construction is a time consuming and extensive process, we will use the dynamic global model for comparison with the diagnoser to verify the correctness. For this verification, the model-checking tool will be used. The model checking tool uses a different state space representation and therefore construction of a timed automata is not required. For these reasons, we do not detail the description of parallel composition and refer the reader to existing works, such as [**Tri98, Tri02, BY04**]

2.3.3. TA as Dynamic global model . We introduce a term (Dynamic) Global model which is often used in this thesis.

Network: state space created by all possible combination of discrete states. See an example on 2.3.2.(a).

Global faultless model: model for considered behaviour description; it is derived as controller projection to state space of physical system. Without fault consideration, controller makes a projection of control sequence into state space to obtain desired controllable trajectory, it is called faultless mode – 2.3.2.(b).

Global model: If a fault is integrated into a model of physical system, controller projections create also faulty trajectories – 2.3.2.(c). Global model model the system behaviour according to sequence order (trajectory). Time is taken into account in a *dynamic global model.*

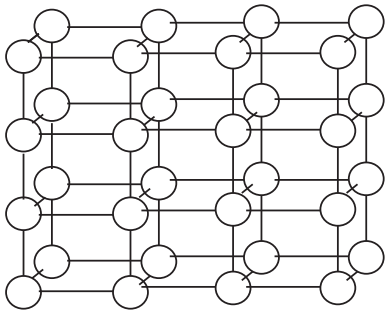
Dynamic global model (also called $\mathcal{G}$-model): is global model with time consideration. Time helps us to precise the model so that a very useful model for time diagnosis is obtained. In component logic introduced in Chapter 4, dynamic global model represents a language of component composition (TA model). Model checker uses a different form of global model representation, normally is not calculated as one global TA, $\mathcal{G}$-model is represented as *e.g.* matrices Difference Bounded Matrices (DBM). Model checker of calculated $\mathcal{G}$-model is treated in Chapter 4. For illustration purposes, in example treated in this chapter, Dynamic Global Model is shown on one global TA model.

REMARK 2.3.2. For a quantised system, which will be treated later, authors (*e.g.* [**BKLS03, LSS00**]) deals with the fault diagnosis based on coherent automaton runs. There, the automaton is not considered as a product of component-based modeling but models of quantised system. Each fault denotes different dynamics which is described by its own automaton. A set of automata makes together the system model.

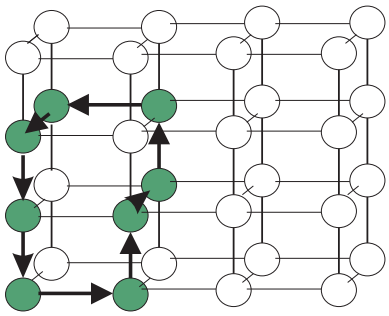
2.4. Example on modelling

Modelling with Timed automata will be illustrated on example of trivial tank system, also known as batch process. We simplify a tank system as much as possible to show described modelling technique.

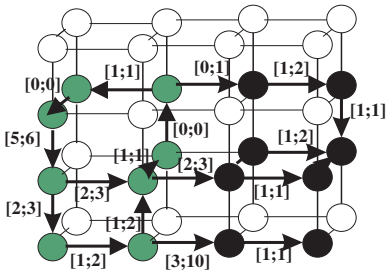
2.4.1. Tank system-instrumentation. It is considered one valve $\mathbf{V_1}$. Valve serves to fill tank. In the tank, we consider two level sensors: $\mathbf{L_1}, \mathbf{L_2}$. Their placement see at Figure 2.4.1.



(a) State space of physical system



(b) Global faultless model (faultless controller projection into state space)



(c) Dynamic global model (controller projection with fault consideration)

FIGURE 2.3.2. State space and controller projection

2.4.2. Dynamic Global model. In DES framework, we suppose discrete variables *e.g.* valve $\mathbf{V_1}$ can be *close* (0) or *open* (1). To illustrate fault modelling with TA, we consider in this example two following faults:

- $\mathbf{V_1}$ can be *stuck close* (2)
- $\mathbf{V_1}$ can be *stuck open* (3)

It means that valve $\mathbf{V_1}$ can be in one of forth states: 0,1,2,3.

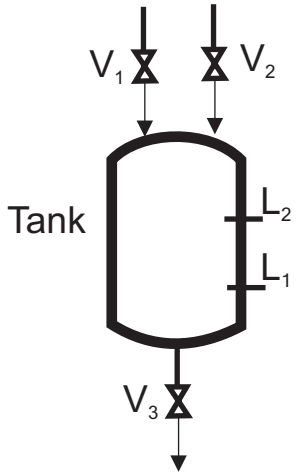


FIGURE 2.4.1. Schema of tank system

Thus, let us describe states of the tank according to sensors states: empty tank (00)[1], intermediate (01) and full (11).

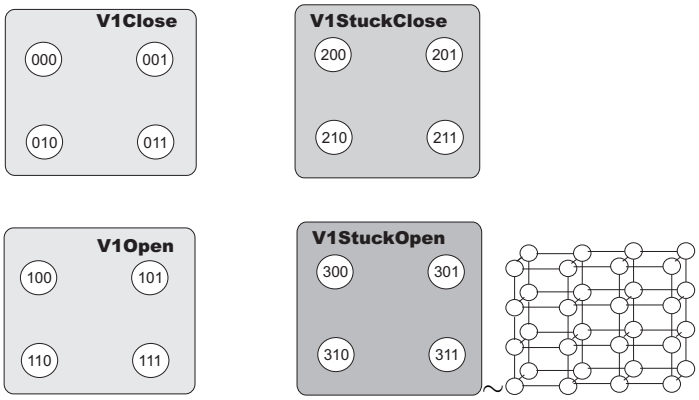
The combination of all possible states together creates the state space. State space describes all possible evolutions. See states $000, 001, \ldots, 311$ at Figure 2.4.2. This composition is called also network (Compare with Figure 2.3.2.(a).)

Apply control sequence on the state space (projection) gives path which present a faultless behaviour. In Figure 2.4.2.(a), see that system starts from the state 000 ($\mathbf{V_1}$ close, $\mathbf{L_1}, \mathbf{L_2}$ is 0). By opening the $\mathbf{V_1}$ is system state changed to 100. When sensor $\mathbf{L_1}$ is reached, the system state is changed to 101. Thus, 111 corresponding with state that $\mathbf{L_2}$ is also reached. Our controller closes $\mathbf{V_1}$, therefore system ends in the state 011.

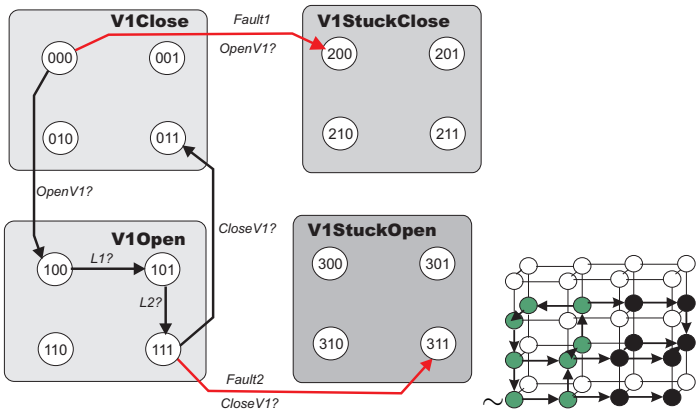
Fault behaviour for $\mathbf{V_1}$ stuck close is modeled by unobservable event $\mathbf{Fault_1}$ which changed state 000 to 200. We have sent a signal to actuator, therefore the supposed state denotes 100 but the real state is 200. Also for fault $\mathbf{V_1}$ stuck open is modeled by unobservable event $\mathbf{Fault_2}$: from state 111 to 311.

The model with all considered faulty and faultless evolution is called Global model ($\mathcal{G}$). Global model incorporated information from plant, sensor, actuator and controller. Global model for complex system can be represented in model checker will be described in Section 4.3.

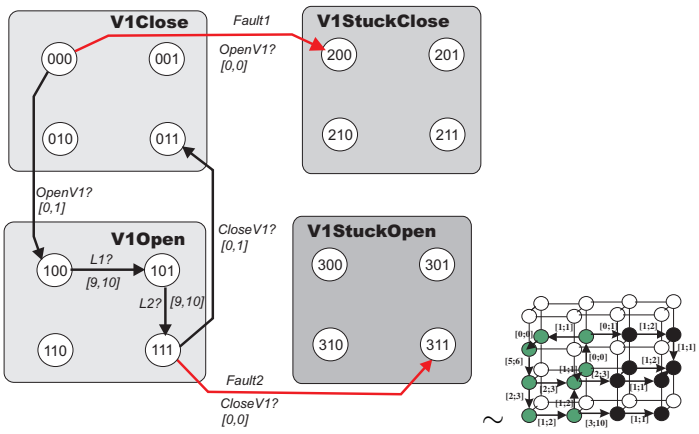
[1]sensors $\mathbf{L_1}$ and $\mathbf{L_1}$ indicate 0



(a) States composition creates state space



(b) Global model



(c) Dynamic global model

FIGURE 2.4.2. Global model

For diagnosis purposes of timed diagnosis, the last step is most important. Obtained global model must be extended by temporal parameters described the dynamic of the real system *e.g.* by opening the valve, $\mathbf{L_1}$

is reached normally at 10_{tu}, etc. This extended model by time is called "dynamic global model" and this model denotes base for time diagnosis.

Hence, time is taken into account to help us distinguish which state could be reached. This fact is important to manage the diagnosability of the system. See time is associated with transitions in Figure 2.4.2.(c).

In the next chapter, we explain how to build dynamic global model (diagnoser), how time plays important role in diagnosability. In Chapter 4, the dynamic global model will be calculated on-line in model checker software. Model checker represents effective way of complex system representation.

2.5. Chapter conclusion

We model the system by timed automata. It is a suitable tool for modeling and analysing the behaviour of timed system. Timed automaton integrates quantitative and qualitative parameters, which makes the modeling powerful and expressive.

We have introduced a "dynamic global model" as a time model describing all possible evolutions according to given controller. The faults are implemented itself.

CHAPTER 3

Timed Diagnosis

In this chapter, the problematics of fault diagnosis of timed system is treated. Our objective is to design a diagnostic system, called shortly diagnoser. Diagnoser is a model-based diagnosis approach within discrete-events system framework. We suppose the diagnoser in the form of timed automaton.[1] We restrict our focus only to timed system. The necessary formal definition for fault detection and isolation (diagnostic) is treated according to the notion of *time diagnosability*. Presence of the fault is defined with reference to the faulty run of automaton, the property detectability represent an ability of fault detection in the system. For an isolation aspect of diagnosis, the property isolability is proposed, which is defined as ability in the system to isolate the fault. A delay between the fault occurrence and detection is described by temporal parameter δ and delay between the fault occurrence and its isolability is characterized by θ. This chapter deals with a diagnoser building. We suppose that diagnoser is built by determination of faulty and faultless behaviour. The diagnoser faultless mode is described first, then the faulty transitions are searched. The diagnoser building and time diagnosability are illustrated by trivial examples in the end of this chapter.

Ce chapitre traite, la problématique du diagnostic des fautes de systèmes temporisés. Notre objectif est de concevoir un système diagnostic, appelé "diagnoseur". La méthode utilisée est basée sur l'approche modèle. Sachons que nous nous limitons aux systèmes temporisés, le diagnoser est alors représenté par un automate temporisé. Dans ce contexte de diagnostic de fautes, une définition formelle de la détection et d'isolation est proposée en accord avec la notion du "diagnosability". La présence d'une faute correspond à l'exécution d'un état

[1] The diagnoser based on composition of TA component is treated in following Chapter 4.

défini comme défectueux de l'automate. La détectabilité est une propriété qui correspond à la capacité de détection de défaut dans le système. Pour la phase d'isolation, une propriété est défini correspondant à la capacité d'un système à isoler (localiser) le défaut. Les performances du diagnostic sont quantifiées à travers différents paramètres : un paramètre de détection: il représente le retard de détection, c'est le temps écoulé entre l'occurrence d'une défaillance et la détection de cette défaillance ; un paramètre d'isolation : il représente le temps écoulé entre l'occurrence d'une défaillance et la localisation de cette défaillance désigné par thêta. Ce chapitre traite la construction du diagnoser. Le diagnoser est basé sur le modèle representant les différents modes de fonctionnement (fonctionnement normal et défectueux). Le mode normal du diagnoser est décrit en premier, ensuite, le modèle est complété par la recherche des différentes transitions menant à des états de fonctionnement défectueux. La construction du diagnoser et les notions de diagnosability sont illustrés par des exemples à la fin de ce chapitre.

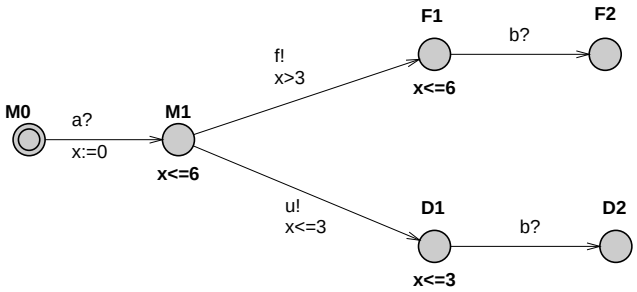
3.1. Diagnoser based on time

Our diagnostics approach belongs to passive diagnostic families. We use model-based techniques on the discrete model. In this chapter, we suppose the timed system is modeled by timed automata.

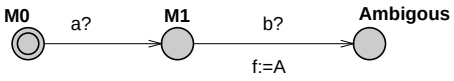
In DES, the fault is often viewed as an event in the system. By fault diagnosis, this faulty event is searched. The common approach in DES changes the problem from the event based fault diagnosis problem to fault estimation problem. We estimate which state could be reachable by given sequence of observable events.

Our objective is to design a diagnostic system, called shortly diagnoser. The diagnoser is an observer for a given system, such that this observer can detect and also identify faults in the behaviour of the system. The diagnoser method is model-based: it is assumed that behaviour of the system is well modeled.

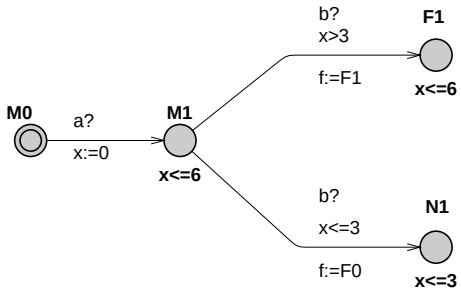
3.1.1. Time extension for diagnosis. Our work extends the classical diagnoser approach by taking time into account. The classical diagnoser automaton involves qualitative representation. Time extends the representation also in a quantitative manner.



(a) Model of timed system



(b) Classical diagnoser



(c) Extended timed diagnoser

FIGURE 3.1.1. Principal idea

For illustration, let us consider the model of timed system given in Figure 3.1.1.(a) (using the automaton formalism). The model represents the following behaviour: suppose two observable events $\mathbf{a}$ and $\mathbf{b}$. The faultless behaviour is modeled by an unobservable event $\mathbf{u}$. We consider the faultless behaviour of the system if event $\mathbf{b}$ occurs up to three seconds after $\mathbf{a}$. If the event $\mathbf{b}$ occurs later than 3_{tu} afterwards, it present a faulty behaviour.[2] On Figure 3.1.1.(b), a classical diagnoser without time consideration is shown. It is evident that is not possible to distinguish faulty and faultless behaviour. Figure 3.1.1.(c) shows the diagnoser with time incorporated. The consideration of time allows us to investigate the dynamics of the system manifested through temporal distances between events and in this way detect and thus identify the fault.

[2]This illustrative example can correspond to the following system: after receiving the $\mathbf{a}$, a valve is opened and the level L_1 is reached within $3s$, which is announced by sensor L1. If the level is not reached, the system is under the fault f_1.

The advantage of using temporal information can be seen through observability. The diagnosis based on sequence observes the occurrence of event. By adding the time constraint, we force the observability. This forcing means that event occurrence and time guard the occurrence of event. The fact that event appears in a defined time τ_2 is naturally more precise than the simple fact that "event appears". Diagnoser analyses times of incoming events and compares a model with real event timed sequence. Diagnoser guards coherent behavior, *i.e.* automaton run, which accept the timing sequence. The fault behavior is also absent of event in defined time, like in Figure 3.1.1.

We will show a time advantage also for alarm treatment case as follows: Let us consider an alarm signal at time t. Then, we search a cause by exploitation of event sequence. We can find the same event sequence (projection, or path) for the different trajectory of physical system. Here, the time plays important role and can help us to distinguish the different cause (different physical trajectory). The different physical trajectory can be difference of time, formally by timed path.

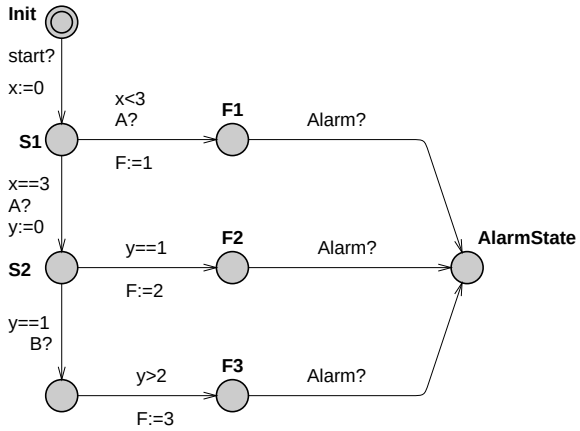
3.2. Time-Diagnosability

In this Section, we focus on time-diagnosability. Informally, diagnosability describes fault detection and isolation ability of a system. Knowledge of system faulty evolution in a fault presence helps us to illustrate phases of fault detection and isolation used in a diagnoser. The diagnosability will be treated and defined on timed automaton runs, which describes the considered system behaviour. The model of system behaviour is considered to be identical with global dynamic model, it means all possible evolutions are known - without and with fault presence.

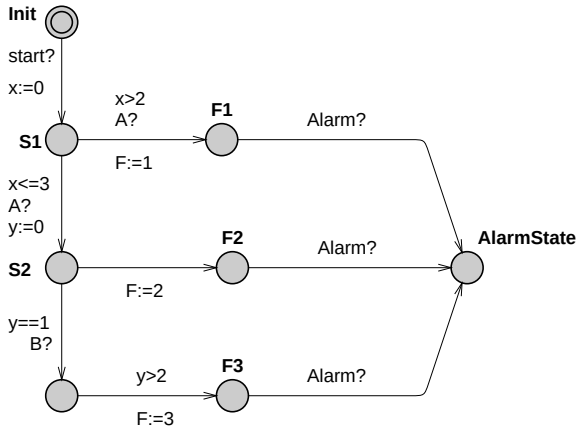
The results of time diagnosability will be illustrated on example.

3.2.1. Semantics of time-diagnosability. The term of diagnosability was introduced in [**SSL$^+$95**]. Here, the diagnosability has no direct connotation of time. Roughly speaking, a language L is said to be diagnosable if it is possible to detect (within a finite delay) occurrences of certain distinguished unobservable failure events.

[**Zad99, ZKW99**] introduces the term of *time-diagnosability*. The sense of diagnosability stays the same as previously, only the time aspect is formulated by extension of an integer $T \geq 0$ such that following both



(a) diagnosable case



(b) b. non diagnosable case

FIGURE 3.1.2. Principal idea (alarm treatment case)

the occurrence of the failure and initialization of the diagnoser, f_i can be detected and isolated in at most T clock ticks.

For dense time, [**Tri02**] proposed the Δ-*diagnosability*, where the integer time T is replaced by Δ; the meaning remains the same.

FDI community uses a term *detectability* and isolability instead of diagnosability. Detectability means the ability to detect a considered fault. The *isolability* is related to the ability to isolate faults. Not every system is detectable within finite delay and also isolable within finite delay. A diagnostic system that can detect faults but not distinguish them is detectable and non-isolable. Our diagnosability will cover both facts: detectability and isolability; for both of them, we define the time parameter. If detectability

and isolability is guaranteed in the finite time, we say the fault f in plant is *diagnosable.*

REMARK 3.2.1. The diagnosability in DES is treated as an ability to distinguish different modes. Frequently, to distinguish the faultless mode from the faulty mode. It means diagnosability in the sense of detectability.

3.2.2. Fault detection. The system behaviour is described as an automaton run in time. The faultless behaviour is defined by the notion of a non-faulty run. We define the non-faulty run as every run accepted by automaton describing the faultless mode.

DEFINITION 3.2.2. *Non-faulty run.* Let $\mathcal{N}$ be the automaton modeled the faultless behaviour. The automaton run is called non-faulty (or faultless), if the timed trace $\xi(t)$ in the time t is accepted by automaton $\mathcal{N}$ to reach the state q_i. The t denotes global time of the system.

The time aspect of the faulty run notion is fundamental. For the same time trace but different time, the runs can be different.

LEMMA 3.2.3. *Faulty run has temporal dependency. For given observed timed trace $\xi(t)$, let t_1 be time of timed trace $\xi(t)$ accepted by automaton $\mathcal{N}$. Let t_2 be a time of timed trace $\xi(t)$ rejected by automaton $\mathcal{N}$. The times t_1 and t_2 must be different.*

To obtain the smallest (or biggest) t, for which the automaton run is rejected in time t, *i.e.* fault is detected, denotes techniques of fault detection, which will be described later. Our aim will be diagnoser, which covers all faulty transition, for every rejected automaton.

The automaton $\mathcal{G}$ is dynamic global model of the system, where the faulty trajectories are included. A fault is viewed here as an unobservable event $e_i = f$. This unobservable event changes the sequence or timing (in the detectable case). The respective incoherent (changed) event in time is observed by the diagnoser so that it can detect and identify the fault. Parameter δ expresses the maximal time to detect the fault.

DEFINITION 3.2.4. *Faulty run (detectable) [1].* If the automaton run is not non-faulty then is called faulty.

According to the definition, faulty run is rejected by faultless automaton. On the other hand, we expect that the faulty run is acceptable in the global model $\mathcal{G}$ and also in the diagnoser automaton $\mathcal{D}$.

This definition is based on the fact that the fault detection is based on guarding the faultless mode. Any violation is considered as a fault. The notion of δ-faulty run is explained on the global system model, where all considered faults create the language of all possibles runs. Otherwise said, in the the global model, the fault is represented by an unobservable event. Time of path from fault occurrence to fault detection is defined as follows:

DEFINITION 3.2.5. *δ-faulty run.* A automaton run $\rho = \gamma_1, \gamma_2, \ldots$ is called faulty if for some $i = 1, 2, \ldots$, $\gamma_i = f$. Let j be the smallest i such that $\gamma_i = f$ and let $\rho' = \gamma_j, \gamma_{j+1}, \ldots$. The fault is δ-detectable if for some $\delta \in Q$, $\mathrm{time}(\rho') \geq \delta$; then we say that at least δ time units pass after the occurrence of f in ρ.

Detection in $\mathcal{N}$ is based on the violation of an invariant. Therefore, the time δ represents the value of invariant. The invariant is expected in the form $t \leq I(q)$. δ equals to the maximal value of the $I(q)$. The knowledge of the invariant violation will be included in the $\mathcal{D}$-diagnoser. The invariant violation can be associated with the fault occurrence in the diagnoser. This violation represents a transition which has not been satisfied within the time δ. The unsatisfied transition can be translated in such a way that the expected event (or condition as consequence of event) does not appear. This involves one of the principles of diagnoser building: faulty transitions will be an invariant violation in the form of absence of some event (absence of true condition). Without the additional faulty transition, the faultless automaton is deadlock because of invariant violation.

This absence of event can be implemented by timer mechanism: timer checks the variable state in the maximal invariant time and if the transition is not true then the diagnoser must announce a fault.

Fault is detectable if automaton $\mathcal{N}$ of faultless behaviour rejects the the timed sequence in time t.

In $\mathcal{G}$ model[3], the fault will be detected with δ delay of its occurrence.

DEFINITION 3.2.6. Fault Δ-*detectable:* Let $\mathcal{G}$ be a global model over Σ with sets of observable/unobservable events $\Sigma_o, \Sigma_{uo} \subseteq \Sigma$. For any two finite

[3]Identically in $\mathcal{D}$-Diagnoser which will be introduced later

runs ρ_1, ρ_2, where for some $i = 1, 2, \ldots$, ρ_1 includes the faulty unobservable event f_i and ρ_2 is non-faulty, fault is Δ-fault detectable in $\mathcal{G}$, if ρ_1 is Δ-faulty and ρ_2 is non faulty then $P(\rho_1, \Sigma_{uo}) \neq P(\rho_2, \Sigma_{uo})$.

3.2.3. Fault isolation. Informally, fault is isolable if faults project their behaviour differently, *i.e.* observed traces are different. We define the isolable faulty run on a global system automaton.

DEFINITION 3.2.7. *θ-faulty run.*

Let $\rho = \gamma_1, \gamma_2, \ldots$ be δ-faulty run, with $\gamma_i = f$ for some $i, j = 1, 2, \ldots$. If there does not exist any other faulty run ρ_2 containing another fault $\gamma_j = f_k$ with the same observed projection $P(\rho_1, \Sigma_{uo})$, the run is called isolable. Let j be the smallest i such that $\gamma_i = f$, and let $\rho' = \gamma_j, \gamma_{j+1}, \ldots$.

The fault is θ-isolable if for some $\theta \in Q$; if $\mathrm{time}(\rho') \geq \theta$, then we say that at least θ time units pass after the occurrence of f for isolate the fault in ρ.

The fault is isolable implies that is detectable automatically. The time parameters δ and θ can be different, but satisfying the constraint $\Delta \leq \Theta$.

DEFINITION 3.2.8. *Θ-isolability:* Let there be a timed automaton $\mathcal{G}$ with the faulty set $\Sigma_f = F_1 \cup F_2 \cup \ldots \cup F_m$ with $m > 1$ and $\Sigma_f \subset \Sigma_{uo}$, . Fault is *isolable* in $\mathcal{G}$ if for any two runs ρ_1, ρ_2 (where $\rho_1 = \gamma_1, \gamma_2, \ldots$ is the faulty run containing the fault f_i, $\gamma_i = f_j$ and ρ_2 is the faulty run containing fault f_j, $\{f_i, f_j\} \in F$, $f_i \neq f_j$) $P(\rho_1, \Sigma_u) \neq P(\rho_2, \Sigma_u)$.

The fault is Θ-isolable if for some $\theta \in Q$ and $\rho' = \gamma_j, \gamma_{j+1}, \ldots$ is true that $\mathrm{time}(\rho') \geq \Theta$; then we say that at least Θ time units pass after the occurrence of f for isolate the fault in ρ_1.

Automaton is diagnosable if every fault is diagnosable. In the case of non-diagnosability, the system must be reconfigured. It means that within the existing information (instrumentation), the system is not able to diagnose the fault. Some manner of system reconfiguration must be performed, *e.g.* an additional sensor.

3.2.4. Fault Diagnosability. To describe a fault in a given system, we use the notion of diagnosability with two time parameters. We described the detectability first, followed by an isolability condition.

DEFINITION 3.2.9. If the fault is Δ-detectable and Θ-isolable, where Δ *and* Θ are finite number, we call the fault is Δ-Θ diagnosable in the plant.

LEMMA 3.2.10. *The time of fault detection is equal or less then time of isolation:* $\Delta \leq \Theta$.

3.3. Diagnoser building

3.3.1. State space of faulty evolution. The system is in a state considered. The state space is divided between the states corresponding with the phase of fault detection as follows:

$Q = Q_{normal} \cup Q_{faulty}$,

where $Q_{faulty} \subset Q_{detected} \cap Q_{isolated} \cap Q_{alarm}$

Thanks to the existing definition, we can define phases of fault diagnosis. These definitions will be based on the fact that event-based fault diagnosis problem will be changed to state based problem.

We consider two types of states, where faulty states are marked according to diagnosis phase.

Normal: Normal functioning mode corresponds with the state space of all possible trajectories (timed sequences) without the presence of fault event (f_i). This mode is also called faultless.

Faulty: If a fault occurs in the system, the system trajectory leaves the faultless state space and is moved to the faulty state space. This mode transition is often unobservable, therefore we model it as an unobservable event f. The faulty state space distinguishes the states according to phase of fault diagnosis.

Detected: The set of states, where the faulty behaviour is detected.

Isolated: In the diagnosable case, after Θ time units, the fault is isolated. This mode contains the state space of trajectories (timed words) accepted by $\mathcal{G}$-model. Informally, if we isolate the fault, the timed sequence (trajectory) has the states in the isolated state space. $Q_{isolated} \subset Q_{detected}$, which means that if the faulty state is **Isolated**, it is also detected.

Alarm: Alarm state space contents the final states, which are associated with the alarm announcement. In these final states, the system is supposed to be stopped. We define the alarm state independently on the others, because it depends on the placement and realisation of alarm system.

The described state space is valid generally for system evolution. The state space of dynamic global model $\mathcal{G}$ suits these descriptions perfectly.

After occurrence of a fault, the system keeps good functioning mode until some violation is observed (detected). As we said, the occurrence of the event is modeled by an unobservable event. This unobservable event is not included in a $\mathcal{D}$-Diagnoser. The observation of the fault is related to state **Detected**, which are implemented in $\mathcal{D}$.

The state space of the faultless automaton $\mathcal{N}$ contains only normal states. Faulty state space is not included.

EXAMPLE 3.3.1. Figure 3.3.1 shows one part of the $\mathcal{D}$-Diagnoser. According to our definitions of state space, the diagnoser contains the following state spaces:

faultless states space contains states:
Q_{normal}={N_INIT, N_WAIT2V1, N_WAIT2L1, N_L1REACH}
faulty state space contains states:
Q_{faulty}={FD_LONGFILLING, FI_V1STUCKCLOSE, FI_SENSORL1, ALARM_OVERFLOW}
detected state space contains:
$Q_{detected}$={FD_LONGFILLING}
isolated state space contains:
$Q_{isolated}$={FI_V1STUCKCLOSE, FI_SENSORL1}
alarm state space contains:
Q_{alarm}={ALARM_OVERFLOW}

3.3.2. Diagnoser building by invariant violation. We propose an algorithm of diagnoser building, which is described in Section 3.3. At Figure 3.3.2, the process is illustrated. According to an existing instrumentation, dynamics of the system is identified. By this temporal knowledge of system evolution, a Diagnoser is built Then, the diagnosability check can be executed. In our case, the diagnosability check is done within diagnoser building. If the diagnosability is not reached, instrumentation must be reconfigured to obtain a model with different projection. The correctness of the $\mathcal{D}$-Diagnoser built is verified in next Chapter 4 for its correctness. The time-diagnosability checking is also shown there.

3.3.2.1. *Global model transformation.* A diagnoser can be built off-line from the global model $\mathcal{G}$ of the system. When the system is modified, the elementary models are updated; consequently, a new global model and a

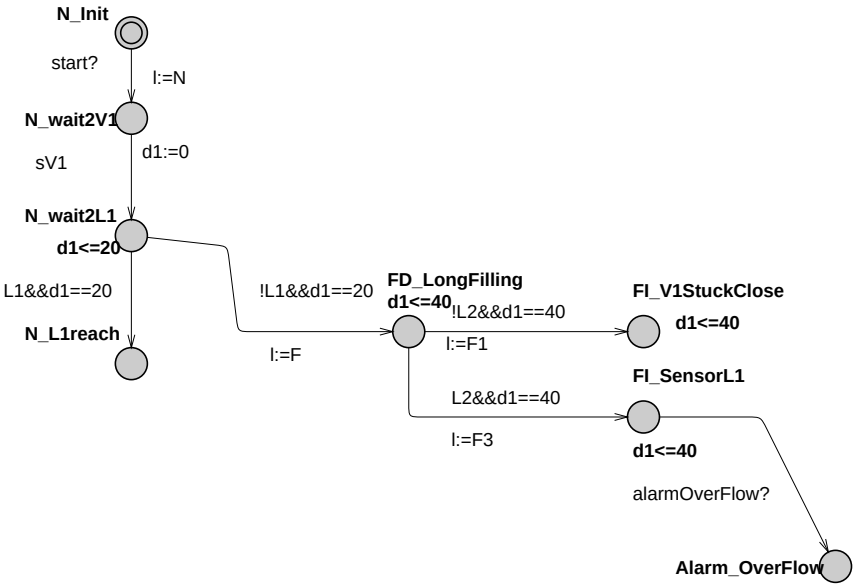


FIGURE 3.3.1. Faulty evolution

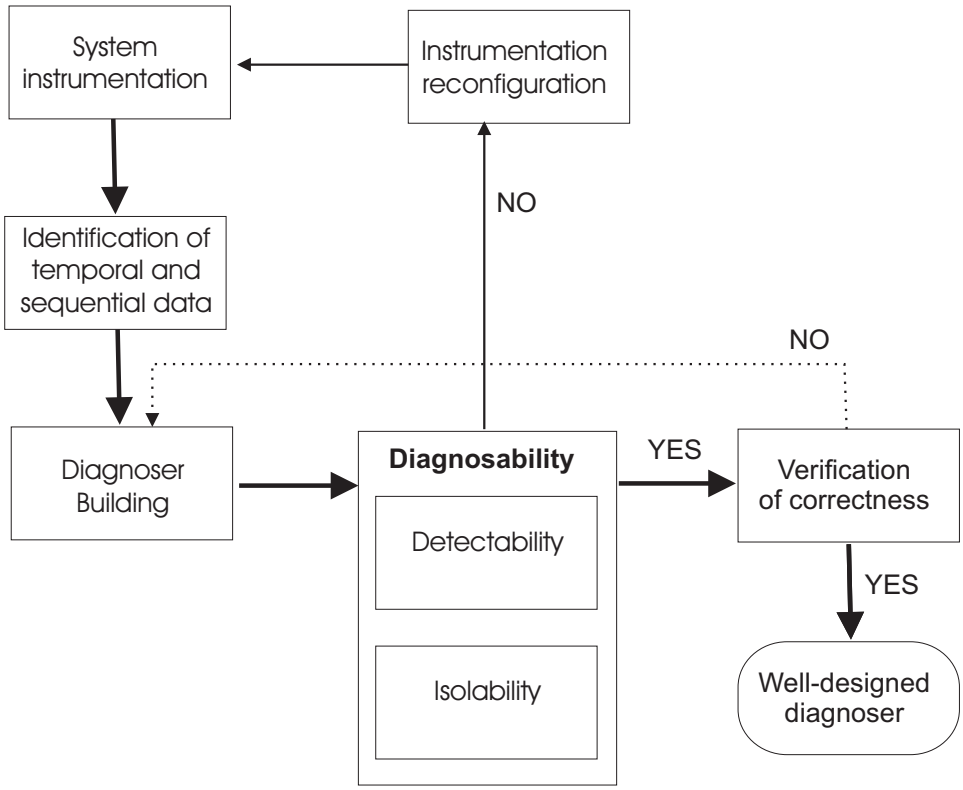


FIGURE 3.3.2. Iterative diagnoser building

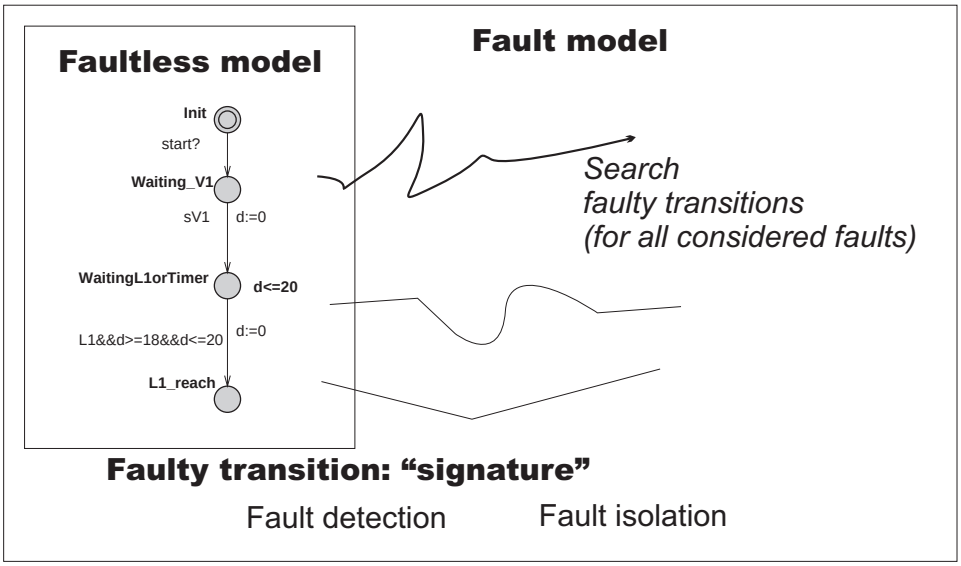


FIGURE 3.3.3. Diagnoser building

new diagnoser are built, which makes this approach generic. This way has the disadvantage of state space explosion and the diagnoser synthesizing. A state space reduction mechanism must be implemented in order to obtain a model of reasonable size to be implemented [**CGLP03**].

3.3.2.2. *Our approach.* We propose an alternative way of diagnoser building: we build the diagnoser by simulation, directly in reduced size. The building process proposed is based on the two aspects of a fault in TDES: fault changes the timing or event sequence. This difference in timing is crucial for our diagnostic approach. As we know from Section 3.2, the presence of the fault can be observed within a time delay. The presence of the fault changes the occurrence of some event in time; it can also cause an expected event not to occur. Therefore, we will diagnose the fault from the observed “signature” (in the sense of changed timing or event sequence).

Details about building follow.

3.3.3. Algorithm for diagnoser building. The design of diagnoser is a systematic extension of faultless model.

We start with the faultless model. For each considered fault, the faultless model will be extended for faulty evolution by determination of faulty transitions and their future trajectory.

3.3.4. Formal definition of diagnoser. Diagnoser is defined as a timed automaton $\mathcal{D}$, where E is the transition function, with temporal constraints.

The hypotheses are that $\mathcal{D}$ has no unobservable cycle (*i.e.* a cycle with only unobservable events) and that failures are permanent.

The aim of the diagnosis is fault detection and identification based on observed events. In order to solve this problem, the state model is directly converted into a diagnoser.

The diagnoser is a deterministic finite state machine $\mathcal{D} = (Q, q_0, E, \Sigma, I)$ where the set of events Σ is set of observable actions of which occurrence can be observed (with the time consideration). Each state Q of the diagnoser contains a label l. The label includes all the faults that have to occur so that this state is reached. Labels are diagnostic results. Labels have the following values:

- $l = \{N\}$: no failure has occurred,
- $l = \{F\}$: a fault has occurred, but there exists more than one candidate. Such label corresponds with the phase of fault detection.
- $l = \{F_{i_1}, F_{i_2}, \ldots, F_{i_k}\}$: at least one fault from the set of faults has occurred. Label in this form means fault isolation.

3.3.4.1. *Diagnoser of faultless behaviour.* As a first step, we built the diagnoser of faultless mode. We consider a monotask sequence of events. In this case, the diagnoser of the faultless mode is a copy of control sequence command. As we mentioned, our diagnoser approach takes temporal information into account. Therefore, the diagnoser is not only the copy of program sequence but the temporal information about task duration is added.

The faultless model can serve for fault detection where no model for fault is needed: every violation against the model is considered a fault. Such diagnoser can be implemented as a diagnoser based on guarding the good functioning mode. See Section 3.4.1.1 for more details about this implementation.

3.3.4.2. *Diagnoser of faulty behaviour.* The transition for fault detection must be semantically different from the faultless transition, which causes deterministic behaviour of the diagnoser. The next step will be an exploration of the leaving transitions from faultless mode. Diagnoser is based on observation of changed behaviour. Let us suppose the δ-faulty

run. Informally, it says that fault can be observed in time[4] δ. The path up to the fault detection will be common with the faultless path. Fault leaves the faultless mode in time $t = \delta$.

The occurrence of the fault is naturally an unobservable event. The diagnoser does not contain unobservable events but only observable ones. It is the reason of the part of faulty trajectory between the occurrence and fault detection not being included in the diagnoser.

3.3.4.3. *Proposed algorithm.* The algorithm for the proposed diagnoser building works as follows:

Let us consider the fault set $\Sigma_f = \{f_{i_1}, f_{i_2}, \ldots, f_{i_k}\}$.

For each considered fault, we find the interleaving transitions E from faultless mode. To this transition, we add the label of considered fault as an affectation. If there exists the same interleaving transition, we change the label of this transition to $l = \{F\}$.

For the same interleaving transitions, let us explore the faulty behaviour by determination of different faulty transitions until the system reaches the final deadlock or final alarm state.

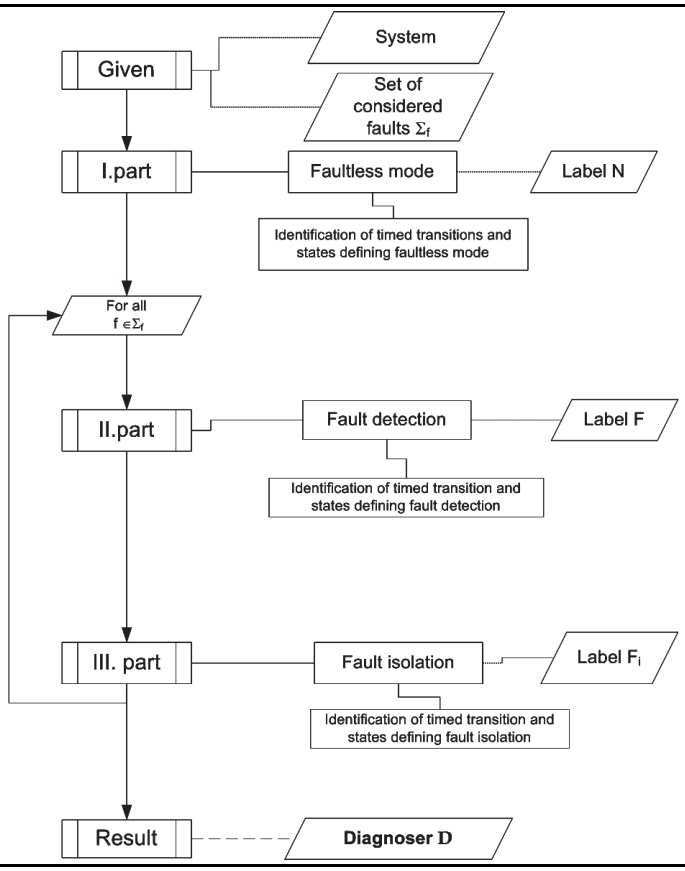
According to our definition of fault detection, fault is detectable if the projection from the faultless model is different. In another words, if the observation of events and timing is the same for faultless and faulty mode, the fault is not detectable. If we find a non-detectable fault, the system conception must be explored and changed to take non-detectable faults (e.g add sensor to force the detectability) into account.

The same rule is valid for the fault isolation, because the reasoning is the following: we can isolate the fault if the observed projection is different. For non-isolable faults, the reconfiguration must be done as well.

3.3.5. Diagnosability check. The diagnosability check is succeeded directly by building the algorithms for detectability (as the second step) and for isolability (as the third step). If we find a faulty transition E, it means that the projections of faultless and faulty behaviour are different, the fault can be detected. If we find a timed event to distinguish these faults in the fault trajectory, that is, if we assure different projection, we also assure the fault isolability.

[4]according to the respective clock which are attached to the time observation

Algorithm 1 Diagnoser building



3.4. Diagnoser implementation

This section will illustrate diagnoser implementation. Firstly, the fault detection technique based on guarding of faultless model will be described. There, we would like to violate the detection mechanism based on algorithm guarding the faultless regime. The faultless model is static model obtained by identification of control program. Then, the fault isolation algorithm is proposed.

3.4.1. Fault detection. The first level of fault diagnosis is the phase "fault detection". In practise, this phase could implement different diagnostics means: voting logic, supervisory program, additional instrumentation. Our approach is based on observation: we expect some supervisory program (PLC, DCS, ...) for fault detection. We focus on algorithm based on timed automata model (diagnoser). Now let us explore the phase of fault detection more in details.

Fault detection works informally as follows: We observe *sequence and time* of incoming events from the plant. Observation is compared with the model of faultless behaviour. The fault is announced:

(1) If an event comes and is not coherent with the model.

(2) If expected event does not come in defined time by the model.

This knowledge about fault aspects is integrated in following algorithms for fault detection and for fault isolation.

3.4.1.1. *Fault detection algorithm.* An algorithm works as follows: After initialization, the main loop of fault detection is started. In the loop is waited for event interruption or timer interruption[5]. The timer represents checking of invariant. If any event comes (event interruption), the received event (timed word) is examined for automaton acceptation or rejection. If is accepted, new state is reached and for its new timer is set up. If the time word is rejected it means an incoherent event is received. If timer finish without any event incomes this case corresponds with the absence of an event in the time. Both are significant cases for faulty behaviour and an alarm is started, the diagnoser result of fault detection is set to one.

3.4.2. Fault Isolation. The modeling of faultless mode has been extended for purpose of fault identification. We must get to know how the fault is propagated, how the fault changes observed timed events. Therefore the diagnoser building searched faulty transitions, *i.e.* deviation from the faultless mode. The faulty transition corresponds to fault observation that means changed time or event parameter. Diagnoser building was treated in detail in the Section 3.3.

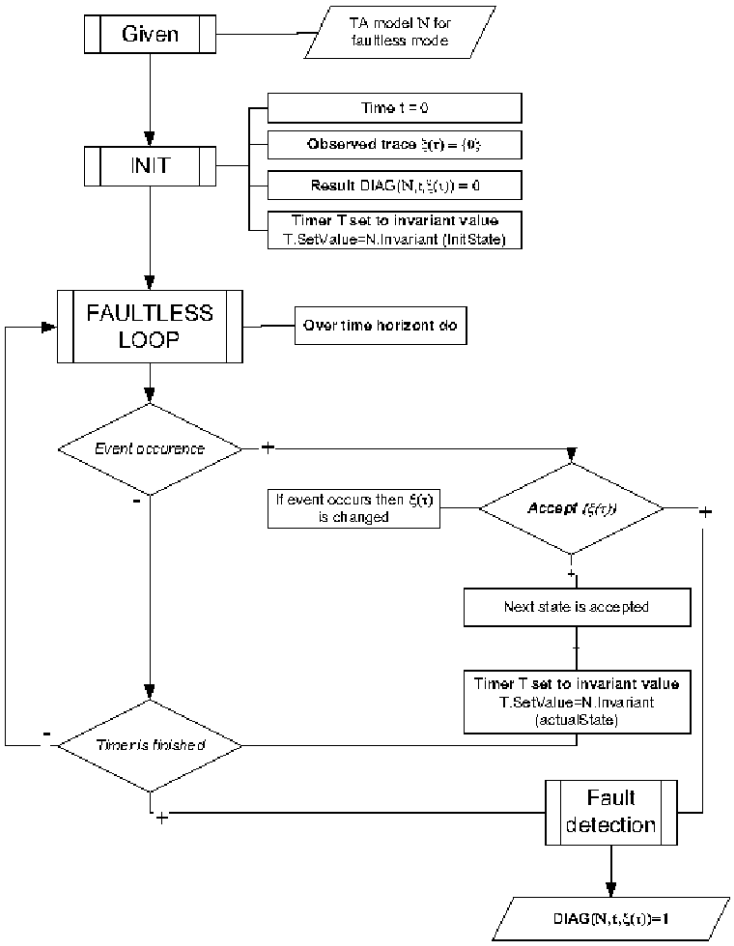
Let us remind that considered faults are related to the control or physical condition, *i.e.* according to control sequence, we want to change the state of the actuator, sensor and it stays in the same position. The time drift is not considered[6].

The fault isolation algorithm uses the same techniques as fault detection: timer invariant checking and test of automaton acceptation. We have two main differences: 1) the model for fault isolation is extended by faulty transitions and therefore 2) algorithm is also extended to check faulty set

[5]For the timer, it is used following notation: set value of the timer is written as "T.SetValue", for announce that timer has finished "T.Finish".

[6]Time drift of fault occurrence means e.g. Valve is open normally and after some time interval is stuck close.

Algorithm 2 On-line fault detection



of candidate to guarantee that algorithm gives the Isolation results in the finite time.

The model for fault isolation must cover the trajectory for respective fault: leaving the faultless mode and the transitions for fault isolation. One transition can be fault detection and isolation also. On the another hand, the diagnoser can include states corresponding to the faulty state space evolution between these transitions. The alarm states considered are associated with transitions of receiving an alarm signal. System in the alarm state is considered to be stopped and no events are awaited.

3.4.3. Diagnoser as Sequential Function Chart (SFC). For automatic control community, the classical representation of a diagnoser is in

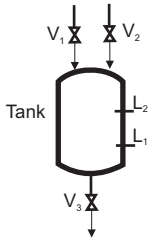


FIGURE 3.5.1. Tank instrumentation

the form of finite state machine (*e.g.*[**ZKW99, RC02**]). For timed system, a timed automata description language is used.

The diagnoser in the form of timed automata employs the labels for fault detection and isolation. The labels can be seen as observed effects of unobservable fault events .

In this community, supervision system, like the industrial programmable automata (PLC), is commonly considered. The language of industrial automata is close to the formalism of timed automata, therefore the result of design is often timed automata and additional algorithms are not needed.

The diagnoser can be implemented from an existing TA model directly. Timed automata can be translated to the sequential language *i.e.* SFC, Grafcet (IEC-61131-3) and then easily implemented into supervision system. The disadvantage can be with the memory and processor time requirements to observe the events.

3.5. Example on Timed Diagnosis

The aim of this example is to provide a practical illustration. Illustrative examples accompany the theory results and different modeling steps. We chose a small batch neutralisation process. The process represents a mixture of two ingredients in one tank to obtain final product; filling in tank must respect a given order.

Instrumentation: Neutralisation batch process consists in one tank that is equipped by two level sensors and three valves. Valve $\mathbf{V_1}$,$\mathbf{V_2}$ as input valves, output valve $\mathbf{V_3}$. The Figure 3.5.1 shows a sensor placement of two level sensors.

3.5.1. Control. The following production sequence takes place: First phase is preparation of the chemical product. Firstly, the valve $\mathbf{V_1}$ is open, an ingredient 1 flows into tank 1. When the tank level $\mathbf{L_1}$ is reached, the

valve $\mathbf{V_1}$ is closed and $\mathbf{V_2}$ is opened. Then, the tank level $\mathbf{L_2}$ is waited for. After the positive edge of sensor $\mathbf{L_2}$, the valves $\mathbf{V_2}$ is closed. We finish with our example here.

Control sequence:

(1) S_0: When the process is initialized, tank should be empty.
(2) S_1: First, valve $\mathbf{V_1}$ is open, an ingredient 1 flows into tank 1.
(3) S_2: If level $\mathbf{L_1}$ is reached then valve $\mathbf{V_1}$ is closed and $\mathbf{V_2}$ is opened.
(4) S_3: If Level $\mathbf{L_2}$ is reached then $\mathbf{V_2}$ is closed.

3.5.2. Considered faults. Let Σ_f be a set of considered faults $\Sigma_f = \{f_1, f_2, f_3\}$, where

f_1**:** Fault valve $\mathbf{V_1}$ *being stuck close.* Practically it means, that tank stays in the initialized state. Controller waits to event $\mathbf{L_1}$ which can not occur because of the stuck valve.

f_2**:** Fault valve $\mathbf{V_1}$ *being stuck open.* This fault can physically cause an overflow.

f_3**:** The third considered fault is sensor $\mathbf{L_1}$ which stays in close position. It means when the level $\mathbf{L_1}$ is reached, this sensor does not indicate it.

3.5.3. Faultless model. Let us define automaton $\mathcal{N}$ which describes a faultless behaviour. In our monotask case automaton is control sequence extended by temporal information describing the faultless mode.

3.5.4. δ-detectability. As it was defined in time detectability, fault can be detect if projection (observation) of faulty and faultless behaviour is different.

f_1**:** For fault f_1, the projection into faultless $\mathcal{N}$ model causes the violation of first transition: $\mathbf{L_1}$ is reached in 20s. If f_1 occurs, the transition is not satisfied and therefore we say that the fact of transition $\mathbf{L_1}$ missing in the required time indicates fault f_1. The difference between fault occurrence and fault detection is 20_{tu}, so we say f_1 is 20-detectable. We have no other observation to detect this fault earlier.

f_2**:** Projection of f_2 behaviour violates the $\mathcal{N}$-automaton also by missing the event $\mathbf{L_1}$. It means the same missing event can be caused by more sources: f_1 or f_2. The time of occurrence of fault f_2 can be

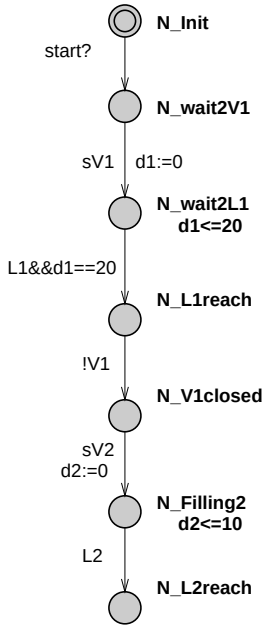


FIGURE 3.5.2. Faultless model

practically somewhere between the initialized state and $\mathbf{L_1}$ being reached. Delta was defined as maximal time, therefore we count with the worst case that fault occurs immediately after initial state. We say f_2 is 20-detectable.

Practically, it is reasonable that the faultless mode employs margins of minimal and maximal faultless time. Delta is then influenced by their difference. Margins give more robustness, on the other hand, we focus on simple model to show diagnostic mechanism. Hence we work without margins.

3.5.5. θ-isolability. One can see that detectability for both considered fault has the same projection. Our diagnostic approach is passive, so we can infer the results only from observation. According to the definition of Isolability, there exists an event $\mathbf{L_2}$ which differs in both projections. If $\mathbf{L_2}$ appears in defined time, it isolates fault f_2, because f_1 can produce no other event. Hence, we found that f_1,f_2 is 40-isolable.

3.5.6. Diagnoser building. Algorithm of $\mathcal{D}$-Diagnoser building will be shown on faults f_1,f_2,f_3.

From the beginning, the program produces signal for opening the valve $\mathbf{V_1}$, but the valve stays in the closed position. This fault caused the system

to be blocked because the controller waits for the sensor $\mathbf{L_1}$, which can be never reached in the faulty case $\mathbf{V_1}$ being stuck closed.

It must be mentioned that without the time aspect, we can not detect this fault.

(1) In the first step, we search a faultless model. (see 3.5.3.(a). The faultless functioning mode is described by temporal behaviour that in our case means that *i.e.* the sensor $\mathbf{L_1}$ is normally reached in time τ_{tu}. See this temporal implementation in the Figure 3.5.3.(a), where from the state `N_wait2L1`, state `N_L1reach` is normally reached at time 20_{tu}. If the time of sensor event passes over this interval or is less, it is considered a faulty behaviour. This faulty "signature" will be imprinted in the diagnoser as a transition leaving the faultless mode.

(2) Next, in second step, for all considered faults, we search for faulty trajectories:

   (a) For fault f_1 ($\mathbf{V_1}$ being stuck close), an invariant of the state `N_wait2L1` is violated, therefore we formulate the faulty transition as the absence of the event $\mathbf{L_1}$ in time $t = 20_{tu}$: $\mathtt{L1\&\&d_1} == \mathbf{20}$.

   (b) For given fault f_2 ($\mathbf{V_1}$ being stuck open), the fault is detected by changed timing[7] of clock watching the filling process 2 (second ingredient). Fault is detected in time $d_2 = 8_{tu}$, *i.e.* in global time $t = 28_{tu}$. This fact will be modeled by new transition which comes from state `N_Filling2` and goes to state `FD_ShortFilling2`. The transition is formulated $\mathtt{L2\&\&d_2} == 8$, *i.e.* in time $d_2 = 8_{tu}$, the $\mathbf{L_1}$ is true. See Figure 3.5.3.(b) to see this transition. $f_l = \{F_2\}$

   (c) Fault f_3 has the same faulty transitions as fault f_1. The label for faulty state `FD_LongFilling` is : $f_l = \{F\}$.

(3) In the third step: search an event for isolating the fault.

   (a) f_1 and f_3: These faults can be isolated by event $\mathbf{L_2}$ appears at time $t = 40_{tu}$. For f_1, the transition is formulated as absence of the event $\mathbf{L_2}$ as $\mathtt{L1\&\&d_1} == 40$ and for f_3 as the event

[7]The command for closing the Valve is executed but valve stays in open position. Then, command continues with opening valve $\mathbf{V_2}$. In presence of fault, two valves are opened instead of one, so it is natural that the system reaches sensor $\mathbf{L_2}$ earlier than for faultless mode.

presence: $\mathtt{L1\&\&d_1} == 40$. See the faulty transitions which check the state of the sensor **L2** in appropriate time at Figure 3.5.3.

(b) f_2: The faulty transition for fault detection f_2 represents also transition for fault isolation. From considered set of faults, there is a concurrent fault with the same fault detection, therefore there is no need for additional transition for fault isolation. Transition $\mathtt{L2\&\&d_2} == 8$ express fault detection and isolation.

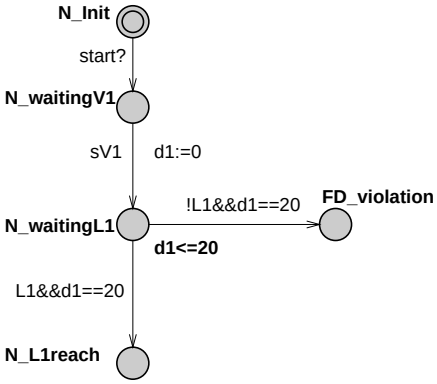
The final diagnoser built is shown on the Figure 3.5.3. As was mentioned in the previous section dealing with time diagnosability, the fault f_1 and f_3 has 20-40-diagnosability. Fault f_2 has 8-8-diagnosability and this case is good example of fault where the time δ is equal to θ.

3.5.7. Fault detection algorithm. Now, we will demonstrate the proposed fault detection algorithm. In this example, we use an interval for faultless mode description. Interval description is common manner to increase a diagnostics robustness. Interval usage can be implemented also in other examples, but there we prefer a readability and simplest to explain a diagnostic mechanism.

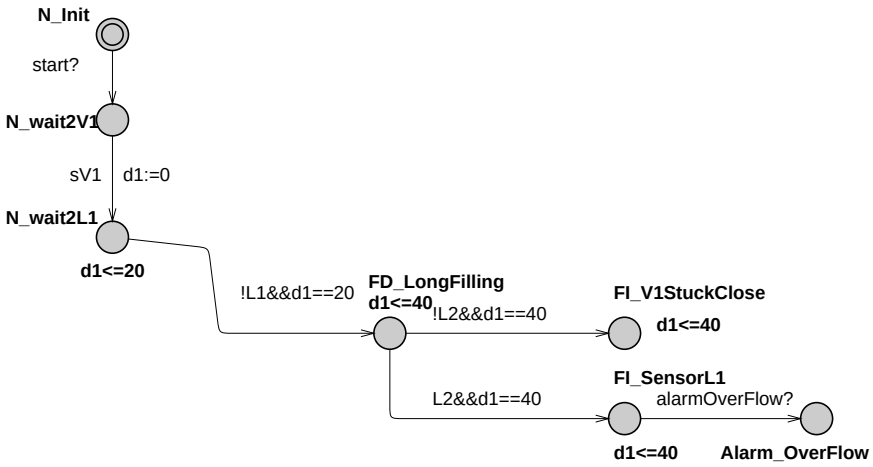
At Figure 3.5.4 you see the faultless model, automaton $\mathcal{N}$.

- We start in time $t = 0_{tu}$ with synchronization signal **start**, we reach the state $\mathbf{N\_waitingV1}$.
- Then we are waiting for the event $\mathbf{senV1}$, *i.e.* the control command for opening the $\mathbf{V1}$. We receive this event within an insignificant delay, timed trail is $\xi(t) = (start, 0)(senV1, 0)$[8].
- After this event, we stay in the state $\mathbf{N\_waitingL1}$, the algorithm sets up the timer according to its invariant value of $T.SV = 20_{tu}$.
- If the event $\mathbf{L_1}$ is received within time $t = 20_{tu}$, the timed trail is accepted and algorithm continues in the faultless mode. Let us suppose that the received timed trail in time $t = 5_{tu}$ is $\xi(t) = (start, 0)(senV1, 0)(L_1, 5)$, *i.e.* that the sensor $\mathbf{L_1}$ changed the value from zero to one at $t = 5_{tu}$. This timed trail (or time path) is rejected and the algorithm detect the fault. This faulty case can have an explanation of noisy signal from the sensor $\mathbf{L_1}$.

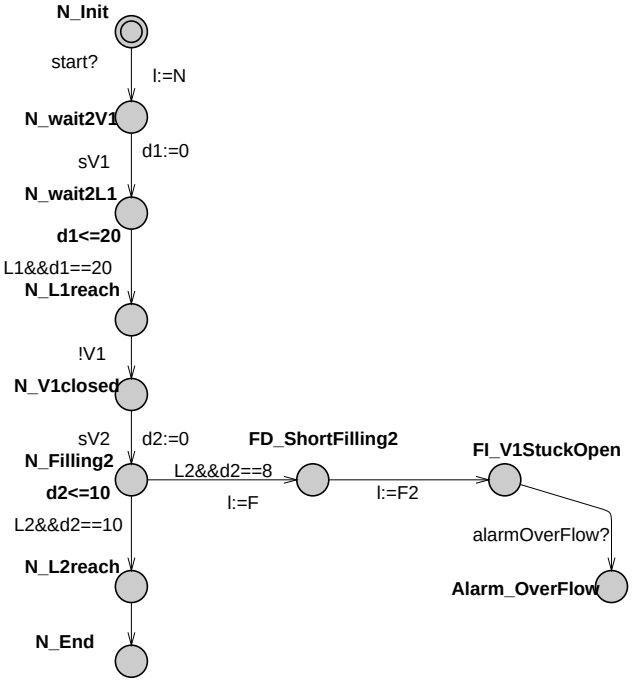
[8]Practically, there is a very small delay between the signal start and signal **senV1**. But this delay is produced by electric circuit of control system and it is not significant for diagnostic task.



(a) Transition for fault detection



(b) Fault transition for isolation (f1,f3)



(c) Transitions for fault isolation (f2)

FIGURE 3.5.3. Diagnoser building

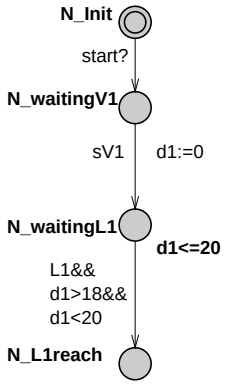


FIGURE 3.5.4. Model of the faultless behaviour.

- If any event does not come before $t = 20_{tu}$, the timer is finished and fault is detected.

3.5.8. Fault isolation algorithm. The fault isolation algorithm will be shown at this point, continuing with the previous example for fault detection. We have reached the state **waitingV1** and we consider faulty state detected by invariant violation in time $t = 20_{tu}$. Now, let us explore the phase of fault isolation. See the Figure 3.5.5, where the faulty state space to capture the faulty behaviour is described (for the considered fault). The detection of the fault is represented by faulty flash **!L1 AND x==20**. The state N_LongFilling is reached, timer is set to the value $T.SetValue = 20_{tu}$. If the event L_2 appears in the time $t = [38, 40]_{tu}$ the fault f_3 is indicated (Faulty sensor L1). If the event L_2 does not appear, the considered fault is f_2 (V1StuckClose). If any another event appears, an unknown fault is detected in the system.

3.6. Chapter conclusion

The diagnoser was formally described, the notion of time diagnosability has been treated. This notion has been shown to have two distinguishable meanings: fault detectability and fault isolability. The additional time parameters, δ-detectability and θ-isolability, denote delays of the respective phase. It means that after a fault occurrence f, at most δ time units pass before the fault is detected and at most θ time units pass before the fault is isolated.

The diagnoser was built using knowledge of fault propagation. The fault is not always detected in the same time as its occurrence. The changed

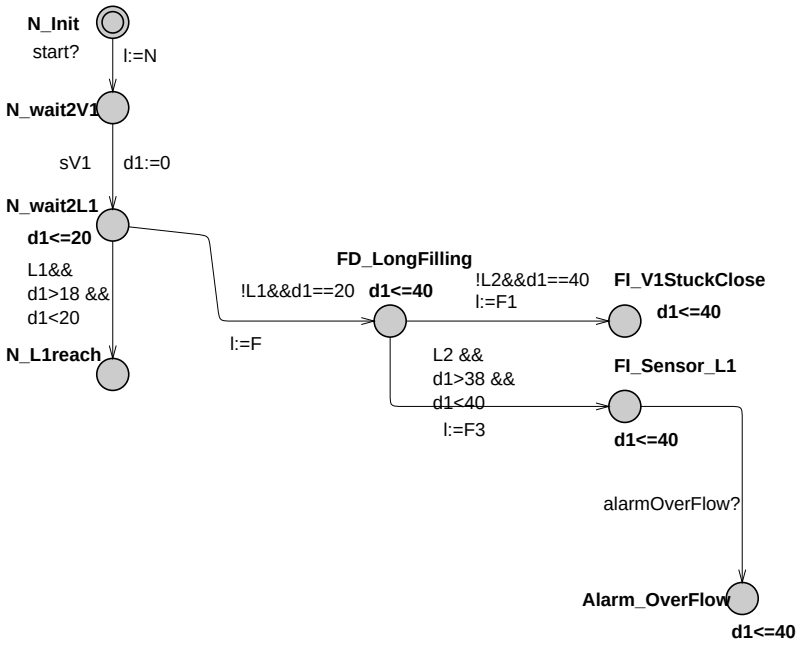


FIGURE 3.5.5. Fault isolation

behaviour is observed as a) changed timing (events come sooner or later) b) changed event sequence (an unexpected event occurs). The principle of the changed behaviour is the same for the phase of fault detection and that of fault isolation. Diagnoser building starts with exploration of faultless mode. For the faulty behaviour, faulty transitions must be explored and added to the diagnoser. Fault propagated by a delayed event (changed timing) caused invariant violation of the faultless diagnoser.

The diagnoser $\mathcal{D}$ (as timed automaton) was built and is considered a reduced copy of the global model $\mathcal{G}$. Fault diagnosis algorithm makes a projection of observation on the model, the aim is to find the faulty trace. Diagnoser model must contain all traces for all considered faults.

The proposed algorithm for diagnoser building studied time diagnosability; it also regarded as a diagnosability check.

Two algorithms were described: i) on-line fault detection ii) on-line fault isolation. The first algorithm worked for a faultless time automaton $\mathcal{N}$, where any violation was translated into a fault detection. The second, on-line fault isolation, was based on the diagnoser $\mathcal{D}$.

CHAPTER 4

Timed diagnosis based on model checking

Model-checking approach is based on property verification of the model. Our model will be composition of timed automata models describing the system. Final model includes all considered faulty behaviour, therefore the model is called global dynamic model (shortly $\mathcal{G}$). The property to verify is formulated according to diagnosis purposes.

Model checker is used for fault diagnosis and contributes in different level:

Modelling: effective complex system modelling

Diagnosis: diagnostic mechanism based on model-checking (we illustrate an alarm treatment case)

Verification: formal verification of diagnoser construction correctness: comparison of $\mathcal{D}$ against $\mathcal{G}$.

To describe a system in the systematic and sophisticated manner, we introduce the component-based approach. This approach has the advantage with an easy and systematic way to model the real system. The fault is easily implemented as an extension of faultless TA model. Respective modifications can be done directly in the component model. Component based modeling consists in a collection of TA executing in parallel. All components together make a synchronous product (dynamic global model $\mathcal{G}$). The dynamic global model represents a time language of all possible time traces. It must be mentioned that the component of controller is considered as one of the components.

We use model-checker also for the fault diagnosis purposes. The property to check if the fault has occurred in the system is formulated. Model checking in the on-line mode has the disadvantage in time (and memory) consumption of each property verification. For complex system, the number of property to check could be a big number. We show the model checker in the context of alarm treatment. We have received an alarm signal; which fault origin caused the alarm must be investigated.

Finally, we deal with formal verification of diagnoser correctness. The idea is to verify a diagnoser (or diagnostic algorithm) which was built without knowledge of global dynamic model. This diagnoser used for the verification is called $\mathcal{D}$-Diagnoser. There are many different possible construction of a diagnoser, of which one we introduced in the previous chapter. Dynamic global model ($\mathcal{G}$ model) is considered as etalon, where every considered faulty evolution is included. The model checker serves as comparator of model $\mathcal{G}$ based on component description against the diagnoser $\mathcal{D}$ built. The property to check is informally: whether every fault that occurs in the system ($\mathcal{G}$) is also announced by the diagnoser ($\mathcal{D}$).

L'approche Modèle-checking est basée sur la vérification des propriétés du modèle. Notre modèle correspond à un assemblage des différents modèles d'automates temporisés décrivant les modes de fonctionnement du système. Le modèle final prend en compte les comportements défectueux de chaque élément qui le constitue. Ce modèle ainsi obtenu s'appelle le "modèle dynamique global" (modèle $\mathcal{G}$). L'approche Modèle-checking est intéressante dans la mesure où elle permet d'une part la modélisation de systèmes complexes et d'autre part la vérification de la validité du modèle. En effet, la modélisation du système est obtenue grâce à la représentation de l'AT de chaque composant. Cette approche a l'avantage de modéliser un système complexe de façon simple et systématique grâce aux modèles AT des différents sous systèmes. L'adjonction des modèles de faute permet de compléter ce modèle pour obtenir le modèle $\mathcal{G}$. Nous l'approche modèle-checking peut être également utilisée pour le diagnostic de défaut. Cependant, cette approche nécessite un temps vérification important. Cette approche est illustrée dans le contexte du traitement d'alarme sur un exemple didactique. A la réception du signal d'alarme, l'exploitation des trajectoires nous permet d'isoler les fautes. Le modèle une fois obtenu appelé « $\mathcal{D}$-Diagnoseur », doit être validé par une vérification formelle de l'atteignabilité des différents états de faute. Cette validation est basée la comparaison de deux modèles d'un même système obtenue de

façon différente. Ainsi, le modèle global dynamique (modèle $\mathcal{G}$), considéré comme "modèle de référence", est comparé au $\mathcal{D}$-Diagnoseur. Le modèle diagnoseur est validé si aucune discordance n'apparaît.

4.1. Model checking overview

The model checker concept come from computer science domain. Following rows introduce bases from a formal system theory. Then is explained how we accomodate his tool for fault diagnosis purposes.

4.1.1. Model checking. Model checking is a method to algorithmically verify formal systems. This is achieved by verifying if the model, often deriving from a hardware or software design, satisfies a formal specification. The specification is often written as temporal logic formulas. Model checking is most often applied to hardware designs. The model is usually given as a source code description in an industrial hardware description language or a special-purpose language. Such a program corresponds to a finite state machine, *i.e.* a directed graph consisting of nodes (or vertices) and edges.

A set of atomic propositions is associated with each node, typically stating which memory elements are one. The nodes represent states of a system, the edges represent possible transitions which may alter the state, while the atomic propositions represent the basic properties that hold at a point of execution.

Formally, the problem can be stated as follows: given a desired property, expressed as a temporal logic formula φ, and a model M with initial state q_0, decide if $M, q_0 \models \varphi$.

Model checking tools face a combinatorial blow up of the state-space, commonly known as the state explosion problem, that must be addressed to solve most real-world problems. There are several approaches to combat this problem.

Symbolic algorithms avoid ever building the graph for the FSM. Instead they represent the graph implicitly using a formula in propositional logic. Then, they use binary decision diagrams or SAT solvers (Boolean satisfiability problem) to perform the graph search. A partial order reduction can be used (on explicitly represented graphs) to reduce the number of independent interleavings of concurrent processes that need to be considered.

Model checking tools were initially developed to reason about the logical correctness of discrete state systems, but have since been extended to deal with real-time and limited forms of hybrid systems.

4.1.2. Temporal logic. In model checking we speak about temporal logic formula. Here is an introduction about temporal logic.

In logic, the term temporal logic is used to describe any system of rules and symbolism for representing, and reasoning about, propositions qualified in terms of time. Consider the statement: "I am hungry." Though its meaning is constant in time, the truth value of the statement can vary in time. Sometimes the statement is true, and sometimes the statement is false, but the statement is never true and false simultaneously.

In a temporal logic, statements can have a truth value which can vary in time. In a temporal logic we can then express statements like "I am always hungry", "I will eventually be hungry", or "I will be hungry until I eat something". Temporal logic has found an important application in formal verification, where it is used to state requirements of hardware or software systems. For instance, one may wish to say that whenever a request is made, access to a resource is eventually granted, but it is never granted to two requests simultaneously." Such a statement can conveniently be expressed in a temporal logic.

4.1.2.1. *Computational tree logic (CTL).* Computational tree logic (CTL) is a temporal logic. It is often used to express properties of a system in the context of formal verification or model checking.

It uses atomic propositions as its building blocks to make statements about the states of a system. CTL then combines theses propositions into formulas using logical operators and temporal operators.

Logical operators. The logical operators are the usual ones: $\neg, \vee, \wedge, \rightarrow$ and $\leftrightarrow$. Along with these operators CTL formulas can also make use of the boolean constants true and false.

Temporal operators. The temporal operators are the state operator following:

- $\mathtt{A}\Box\varphi$ – *All*: φ has to hold on all paths starting from the current state.
- $\mathtt{E}\Diamond\varphi$– *Exists*: there exists at least one path starting from the current state where φ holds.

4.2. Complex system modeling

For complex systems, a sophisticated approach is needed for system modeling and analysing. One of the major advantages is its easy modification. Model of the complex system means final synchronous composition of all components; it will be described later. What is important is that any desired modification in component causes the modification of final product. This makes this approach attractive. We avoid the difficulty related to rebuilding the diagnostic system, which can be error-prone.

In practice, it can still be difficult to apply component-based approach for industrial size systems. The reason is huge memory usage needed to explore state-space of the network of TA. For this case, we illustrate complex system modeling on alarm treatment problem and for verification of $\mathcal{D}$-diagnoser. Dynamic global model can be compared with the examined diagnoser to verify its correctness.

As it was mentioned in Section 2.3.3, components of physical system make together a state space called network. The component of controller give us trajectories in the network, we call it "global model". The dynamic global model $\mathcal{G}$ describes all possibilities of faultless and considered faulty system evolution with time consideration.

4.2.1. System decomposition. The component-based modeling facilitates the complexity of system behaviour description. The system to be diagnosed is decomposed into components describing:

- plant behaviour
- controller behaviour
- (diagnoser[1]) behaviour

Our focus is given to physical and controller behaviour. Physical behaviour can be divided into the components according to its purpose from control theory (see the system component in Figure 4.2.1). The physical behaviour is given by i) actuator ii) plant and iii) sensor.

Parallel composition (synchronized product) of all physical components makes together the state space of all possible traces of the system (language of automata $L(\mathcal{A})$). This state space can be rather large, according to the

[1]For Diagnoser verification, $\mathcal{D}$-diagnoser will be added as one additional component. It serves for verification of dynamic global model of the system. This topic will be treated later, in Section 4.4.

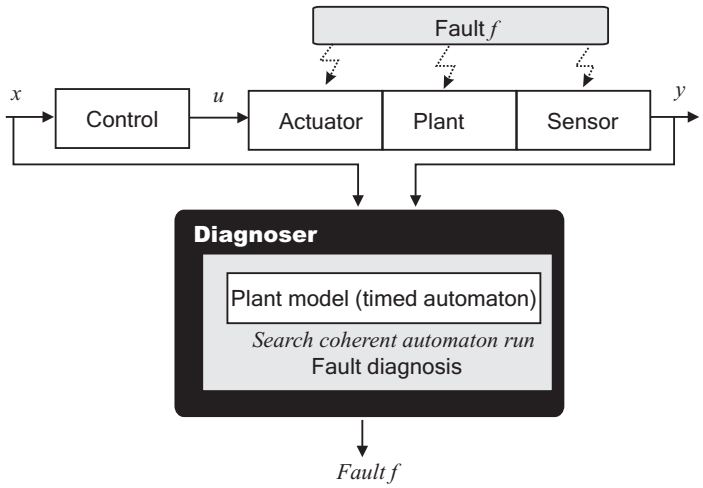


FIGURE 4.2.1. System decomposition

component quantity. We suppose that a controller can be modeled as one or more timed automata. Thus, composition of physical behaviour with the controller reduces the language generated by $\mathcal{G}$-model significantly. It means that from all possible evolutions of the system, only the trajectories accessible by the controller are chosen (also in the presence of faults).

4.2.1.1. *Plant behaviour.* Each component can be modeled by timed automata. It is not needed that all components depend on the time.[2] The physical behaviour must respect the reality of interconnection between components. For this reason, synchronisation events are used to model a realistic physical behaviour.[3]

4.2.1.2. *Control behaviour.* We model also the controller behaviour. It must be possible to model to controller by a timed automaton. For the reason of controller legibility, we consider the controller in the form of sequential program (the standard known as sequential function chart, abbreviated SFC, as an equivalent of GRAFCET). Generally, it is possible to model another kind of control like in [**BGBDS04**], but it stays out of this thesis' scope.

[2]Consider *e.g.* the component $\mathbf{V_1}$ that has no timed transitions. On the other hand, the component tank **T1** has timed constraints which describe the dynamic behaviour of the system.

[3]Actuator having received a synchronized event of changing the state, the synchronized event is be sent to the physical system and there, the synchronous event for sensor component is produced.

4.2.2. Dynamic global model as a parallel composition. Suppose that the system to be diagnosed has N individual components: typically, these components consist of equipments and controller. First, we build TDES models for these components. Let $\mathcal{A} = (Q, q_0, \Sigma, E, I)$ refer to timed automata model of i-th component. The states in Q_i and the events in Σ_i correspond to the normal and faulty behaviour of the component.

Global model of the system will be a synchronous composition of components models $A_i, i = 1, \ldots, N$. Some of the events in the global system Σ are observable (Σ_0). The rest is a set of unobservable events, including faults $\Sigma_f \subset \Sigma_{uo}$. The global model represents a language of all possible time traces.

The problem to calculate the global model is time and memory consuming. Therefore, model checking tools use the DBM (Difference Bounded Matrices) representation of state space. The global model is built on-line to verify the respective property.

According to the large state space representation, our diagnoser is not built from the global model. We use the global model to verify the correctness of designed diagnoser.

4.2.3. Faulty implementation in component model. Adding fault to the system model means extending it with faulty description for respective component. We consider that faults are non-reparable, *i.e.* permanent.

The system model without the fault is much less extensive than the faulty one. Faults normally make a lot of possible trajectories; it is therefore advantageous to use a systematic method for their description.

4.3. Alarm treatment diagnosis

In *model checker concept*, diagnostic mechanism uses the knowledge of global dynamic model. The diagnostic mechanism is checking different properties in the $\mathcal{G}$-model. Our property to check is reachability of faulty state. Realisation of this concept has difficulty with memory and time consumption of the verification task; also with property formulation and model-checker implementation. On the other hand, the major advantage is in using global dynamic model represented in model-checker.

We illustrate the model-checker approach for alarm treatment diagnosis. Aim of the alarm treatment is a fault isolation. Alarm signal is received and then investigation of the fault origin (source) is executed. We call this

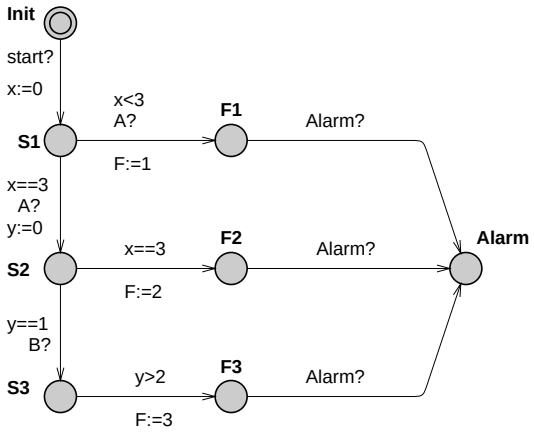


FIGURE 4.3.1. Backward Time Analysis - Principle

method of investigation backward time analysis. We retrace the automaton from faulty state to initial state and we search the coherent run which isolates the fault. An illustration of alarm treatment case is proposed in Section 4.5.

4.3.1. Backward time analysis. We define a *backward time analysis* as a method for backtracking an automaton run to obtain one coherent trace which signifies a fault. The aim is seen as follows: for given system model, observation and alarm signal, analyse the model to find the faulty cause (fault isolation). BTA retraces the automaton from the final alarm state back to an initial state. This set of timed path is examined to identify a fault.

4.3.1.1. *Principle.* See automaton graph with fault model in figure 4.3.1. This automaton graph represents a system model. The faultless states are ($\mathbf{Init}$, $\mathbf{S_1}$, $\mathbf{S_2}$, $\mathbf{S_3}$). Faulty flash leave these faultless states for faulty states $\mathbf{F_1}$, $\mathbf{F_2}$, $\mathbf{F_3}$. We consider one alarm state $\mathbf{Alarm}$.

Backward time analysis explores the return path from the final state $\mathbf{Alarm}$. We obtain three paths (sequences). Only by sequence consideration, one cannot distinguish the faults $\mathbf{F_1}$ and $\mathbf{F_2}$. We must employ the time. Let us see that fault can be identified by knowledge of time (path length). Time of alarm denotes *time of path*. By this time of path consideration, we obtain three different projections, which is condition necessary and sufficient for fault identification.

4.3.2. Diagnostic mechanism. Backward time analysis corresponds to reachability checking: For given alarm time and stored history, what fault has occurred in the system? Model checker help us to find coherent runs: Coherent run has the property which satisfied a verification.

All associated fault with respective alarm $f_1, f_2, \ldots, f_n$ are checked according to formulated property: "Does exist any automaton run including an unobservable fault event f_1 (event cause reaching the state F_1)?" We rewrite in a formal verification language: $\mathtt{E}\Diamond\mathcal{G}.F_x$. It means: "it is possible to reach in our dynamic global model $\mathcal{G}$ the faulty state $\mathbf{F_x}$?"

To this general property of fault occurrence, we add time of alarm to verify: does system reach the fault state in received alarm time? Formally, $\mathtt{E}\Diamond\mathcal{G}.\mathtt{F1}$ and $\mathtt{time}{==}\tau$. Time of fault occurrence must be equal to the time of path.

See the algorithm 3, time property is checked for all inspected faults. If the result of the verification is true for more then one fault, we must explore projection and find where is the difference in the automaton run. If we find the difference, we reformulate the property and re-execute the verification. If the model is diagnosable, verification gives us one result, *i.e.* one isolated fault.

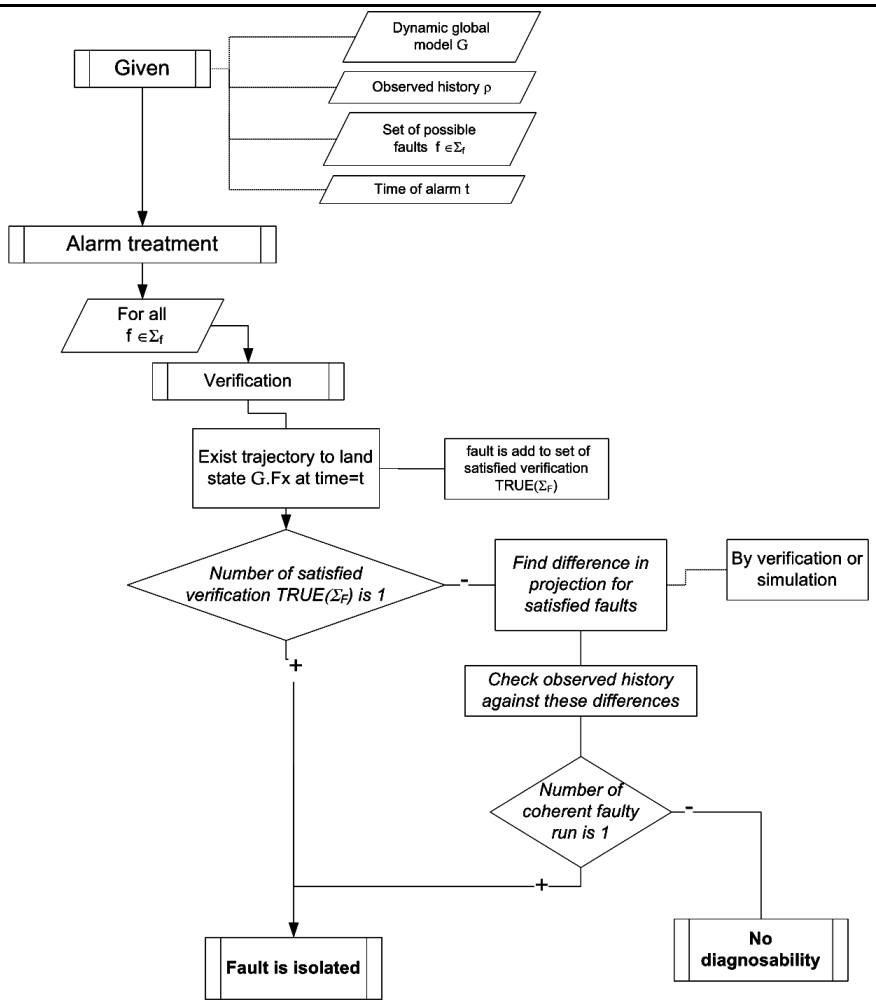
REMARK 4.3.1. Model checker is based on property formulated using reachability of some state (state based). On the other hand, fault is viewed as an unobservable *event*. Therefore, the property of fault occurrence is transformed from event-based to equivalent state-based form: property is formulated as reachability of faulty *state*.

4.4. Formal verification

In this section, we provide a formal verification of diagnoser ($\mathcal{D}$-diagnoser) built in the previous chapter 3. The idea of formal verification is to check a $\mathcal{D}$-diagnoser behaviour according to dynamic global model ($\mathcal{G}$-model). From this, the proposed verification can be done for diagnoser built in a different way, not necessarily as we have done it. Formal verification can be useful for checking existing implemented diagnostic system as well.

As we have mentioned, verification makes a comparison of designed diagnoser against the dynamic global model. This verification can be easily done

Algorithm 3 Backward time analysis



in some existing model-checking tool. The formal verification and simulation can be very useful for diagnoser design and re-design. The verification can discover the system trajectory which is not included in $\mathcal{D}$-diagnoser.

The manner of component modeling is useful for formal verification of the model and diagnoser. Diagnoser will be one of the components. By formal verification, we will check properties of the model. Properties to check are proposed: Is the diagnoser able to capture all faults according to the specification? Is it possible that diagnoser announces the fault without the occurrence of the fault in the physical system?

This chapter deals with the part of verification of the results obtained in previous chapters. The system model $\mathcal{G}$ will be compared with the designed diagnoser $\mathcal{D}$ to decide upon the diagnoser correctness.

We understand the correctness of diagnoser as follows: The diagnoser is *well-designed* (correct) if it announces a fault if and only if the fault already exists in the system model. The fault in the system model represents an unobservable event which can cause the deviation from faultless behaviour (in detectable case). Diagnoser detects this changed behaviour through an observation change (event and time). Detection of changed behaviour is used to detect and isolate the fault. This kind of verification is helpful in order to check the diagnoser before its implementation.

We use the model-checking tool for formal verification of diagnoser correctness UPPAAL. An example of batch process is modeled and simulated to verify the theoretical results in Section 4.5.

4.4.1. Correctness verification. This correctness verification is motivated by recent research about model-checking methods to verify the correct functioning of the control program, *e.g.* [**BGBDS04**]. We would like to extend the idea of model-checking to the area of fault diagnosis design. Our aim is to provide a formal diagnoser verification.

Figure 4.4.1 shows the schematic graph of diagnoser verification. Verification compares reachable states in global model $\mathcal{G}$ and diagnoser $\mathcal{D}$. If the correctness properties are satisfied for the diagnoser $\mathcal{D}$, we say that diagnoser $\mathcal{D}$ is well-designed. If a property is violated, the diagnoser $\mathcal{D}$ must be modified. Modification of diagnoser can be motivated by taking into account some evolution of the global model $\mathcal{G}$ which is not included in the diagnoser $\mathcal{D}$. This modification can be also related to additional information needed to reach a diagnosability. Practically, it means modify the system instrumentation and install additional sensor.

Correctness verification denotes the last step of diagnoser design, where the properties of correctness are checked. Correctness will be checked by following properties:

- verification of fault implication
- verification of δ-detectability and θ-isolability

4.4.2. Fault implication. The verification of reachability belongs to the verification of safety properties, which are of the form "something bad will never happen". Reformulated in a positive way, "something good is invariably true". Let φ be a state formulae. We say that φ should be true in all reachable states with the path formula $\mathtt{A}\Box\varphi$.

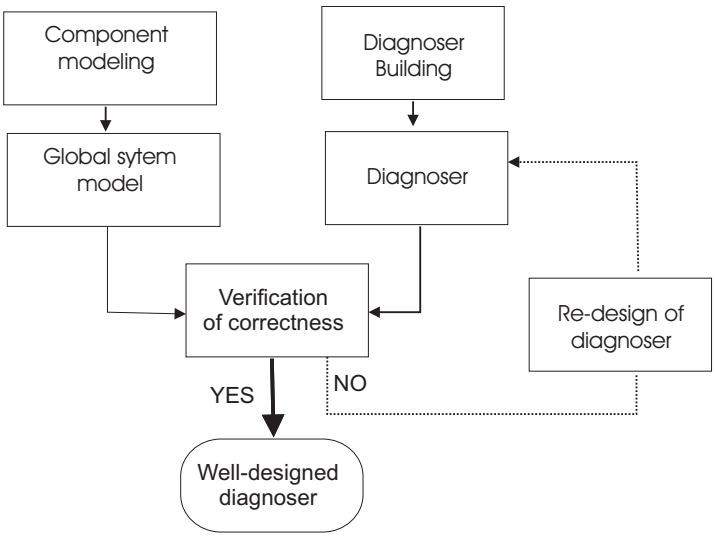


FIGURE 4.4.1. Verification of diagnoser correctness

The verification in this case is formulated:

- Diagnoser $\mathcal{D}$ should announce every fault already occurred in $\mathcal{G}$.

It will be translated into a reachability problem: a state being reachable in $\mathcal{G}$ implies a state being reachable in $\mathcal{D}$. States to verify belong to the faulty state space, $\mathcal{D}$.`FaultyState1` $\in Q_{\mathcal{D},faulty}$, $\mathcal{G}$.`FaultyState1` $\in Q_{\mathcal{G},faulty}$. Hence, the property to verify is formulated as follows:

$A\Box\mathcal{D}$.`FaultState1 imply` $\mathcal{G}$.`FaultState1`

DEFINITION 4.4.1. *Well-designed diagnoser.* For a considered set of faults $F = \{f_1, \ldots, f_m\}$, where m is quantity of considered fault, and its corresponding global model $\mathcal{G}$, which is composed by timed automata components $\mathcal{G} = \mathcal{G}_1 \parallel \mathcal{G}_2 \parallel \ldots \parallel \mathcal{G}_i$ and we suppose that component of controller and diagnoser are included. If the automaton run of dynamic global model $\mathcal{G}$ contains an unobservable faulty event f_k then diagnoser must announce the result of occurrence of fault f_k. And *vice versa*: every diagnoser faulty run f_i must be implied by occurrence f_i in global model. If these properties are valid for all considered faults from Σ_F, we say that diagnoser is *well-designed*.

REMARK 4.4.2. The fact that $\mathcal{G}$ contains the fault event f_k corresponds to the fact that $\mathcal{G}_i$ reaches the state $\mathcal{G}_i.F_k$. This transformation is given by checking of state; we cannot check an event occurrence in UPPAAL.

4.4.3. Reachability verification. Reachability properties ask, whether the given formula φ, possibly can be satisfied by any reachable state. To perform a reachability check can be useful through the diagnoser design. We have already shown a reachability verification in Section 4.3, where reachability verification has served as diagnosis mechanism for alarm treatment case. In this section, we are focused on the behaviour of $\mathcal{D}$-Diagnoser.

The property φ to check is: Is it possible to reach the state state $\mathcal{D}.FaultState1$ in some defined global time $time = \tau$? Formally, formulae is written:

E$\Diamond\mathcal{D}$.FaultState1 and time==τ

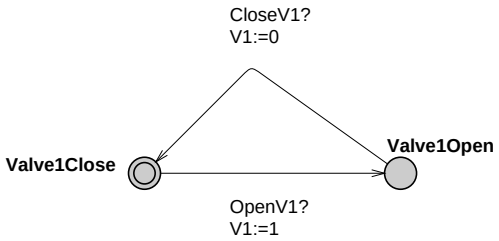
4.4.4. Verification of δ-detectability and θ-isolability. Verification of diagnosability is viewed as verification of reachability. The temporal parameter can be verified by an additional time parameter in the formulae. Our verification task contains three steps. In the first one, we must determine the time of fault occurrence. In the second step, the time of fault detection is verified; and in the third step, the time of isolation is verified.

(1) determine occurrence of fault
   E$\Diamond\mathcal{D}$.FI_F1 and $\mathcal{D}$.F1 and time==τ_1
(2) verify the time of detection δ
   (a) E$\Diamond\mathcal{D}$.FI_F1 and $\mathcal{G}$.F1 and time<$\tau_1 + \delta$
   (b) E$\Diamond\mathcal{D}$.FI_F1 and $\mathcal{G}$.F1 and time==$\tau_1 + \delta$
(3) verify the time of isolation θ
   (a) E$\Diamond\mathcal{D}$.FI_F1 and $\mathcal{G}$.F1 and time=$\tau_1 + \theta$

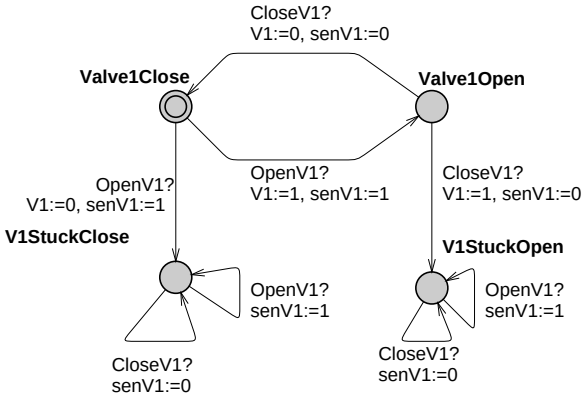
4.5. Example of model-checker

In this section, we work on example already introduced in section 3.5. First we show the component modeling. Then, the verification mechanism for alarm treatment case is proposed. And finally, the formal verification of $\mathcal{D}$-diagnoser built in previous chapter with dynamic global model is treated.

4.5.1. Component modeling. The component modeling will be detailed. We show the component model for one of the valves, where the fault implementation is considered. Then, the model for plant is presented and finally the controller description is shown.



(a) Valve 1 faultless model



(b) Valve 1 model with faults consideration

FIGURE 4.5.1. Model of a faulty behavior

4.5.1.1. *Valve* $\mathbf{V_1}$. We focus on the actuator valve $\mathbf{V_1}$, see on Figure 4.5.1.(a). Valve $\mathbf{V_1}$ in the faultless mode can be in the two states: **Valve1Close**, **Valve1Open**. The state can be changed by receiving the signal **OpenV1** or **CloseV1**, where the variable $\mathbf{V_1}$ is set according to current state ($\mathbf{V_1}$ is set to 1 for open state).

Now, we add a description of the faulty behaviour. We suppose two faults for $\mathbf{V_1}$: $\mathbf{V_1}$**stuckopen** and $\mathbf{V_1}$**stuckclose**. These states are added to the $\mathbf{V_1}$ model, along with transitions. The variable $\mathbf{V_1}$ describes the real state of the valve. The variable **senV1** is describing that controller already sent a signal to open or close the valve. See the extended fault model at figure 4.5.1.(b). The faulty behaviour corresponds to setting the variable $\mathbf{V_1}$.

4.5.1.2. *Plant.* The component $\mathbf{T_1}$ models a physical behaviour of the tank $\mathbf{T_1}$, Figure 4.5.2. Dynamics is implemented using timed transition (guard) which observes the respective clock of the component, *e.g.* from state **Empty** to **Inter** (the $\mathbf{V_1}$ is open and others valves are closed), the

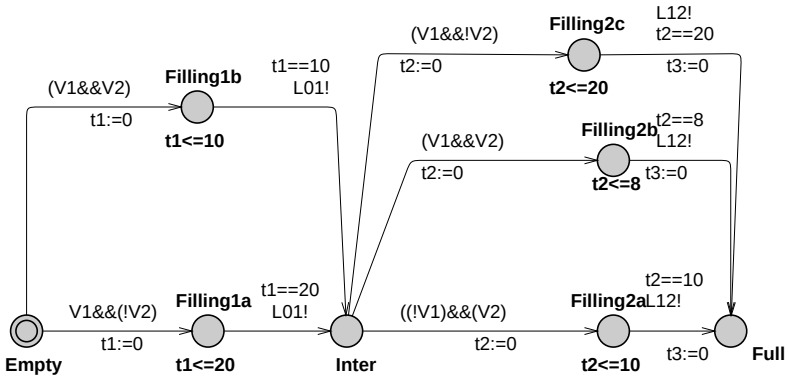


FIGURE 4.5.2. Modeling of component: Tank 1

automaton reaches the state **Inter** in time 20_{tu}. Other transitions between the states **Empty**, **Inter**, **Full** and **Overflow** work analogously. These transitions produce an event corresponding to the sensor reading $\mathbf{L01}, \mathbf{L10}, \mathbf{L12}, \mathbf{L21}$.

This model of plant ($\mathbf{T_1}$) describes all possible evolutions, faultless and also faulty. For the diagnosis purposes, it is needed that the plant model covers any possible behaviour and not such that is desired, the faultless cases. Our model of the $\mathbf{T_1}$ component covers dynamic description based on valves states.

4.5.1.3. *Controller.* See the part of controller model at Figure 4.5.3. This model corresponds to the control sequence in our illustrative example. The $\mathbf{V_1}$ is opened when the level $\mathbf{L_1}$ is reached, $\mathbf{V_1}$ is closed and $\mathbf{V_2}$ is opened to reach the level $\mathbf{L_2}$.

4.5.2. Alarm treatment diagnosis in model checker. In this example, we consider additional alarm signal **Overflow** of Tank 1. Signal is activated with additional sensor fixed on the top of the tank for the indication of an overflow state.

Thus, we use dynamic global model $\mathcal{G}$ described in the previous example. We add an overflow signal **Overflow** to model the overflow situation, see **Overflow** state at Figure 4.5.4.(a). The overflow can be physically caused by the following faults: f_2 ($\mathbf{V_1}$ stuck open), f_4 ($\mathbf{V_2}$ stuck open), f_3 ($\mathbf{L_1}$ faulty), f_5 ($\mathbf{L_2}$ faulty). See Table 1 for faults summary; $\Sigma_f = f_2, f_3, f_4, f_5$.

Let suppose that the signal **Overflow** is received in time τ. According to the algorithm described, we formulate the property for each fault and check:

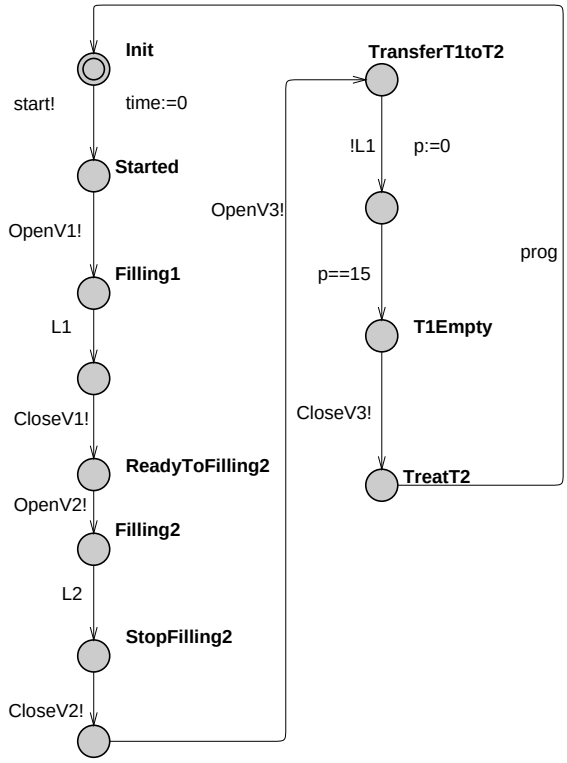


FIGURE 4.5.3. Controller modeling

E$\Diamond\mathcal{G}$.F and time=AlarmTime

Thus, by applying the function $TRUE$ of all possible paths which lead to **alarmOverFlow**; we find the following results:

- If time $\tau = 28_{tu}$ then $TRUE(\Sigma_f, alarmOverFlow) = f_4$
- If time $\tau = 40_{tu}$ then $TRUE(\Sigma_f, alarmOverFlow) = f_2$
- If time $\tau = 30_{tu}$ then $TRUE(\Sigma_f, alarmOverFlow) = f_3, f_5$

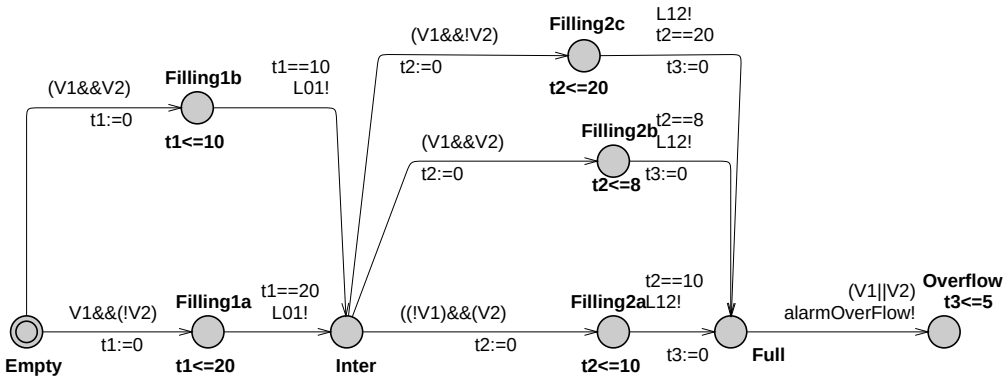
One can see that time helps us to isolate the fault. In the case of common results for time $\tau = 30_{tu}$, we get to know that the projection differs from the event $\mathbf{L_2}$, which can be checked for its occurrence (by reaching some state). Presence or absence of event $\mathbf{L_2}$ in ρ isolates the respective fault f_3 or f_5. Finally, we show $\mathcal{D}$-Diagnoser on Figure 4.5.4.(b), see mentioned trajectories in this model.

4.5.3. Diagnoser verification.

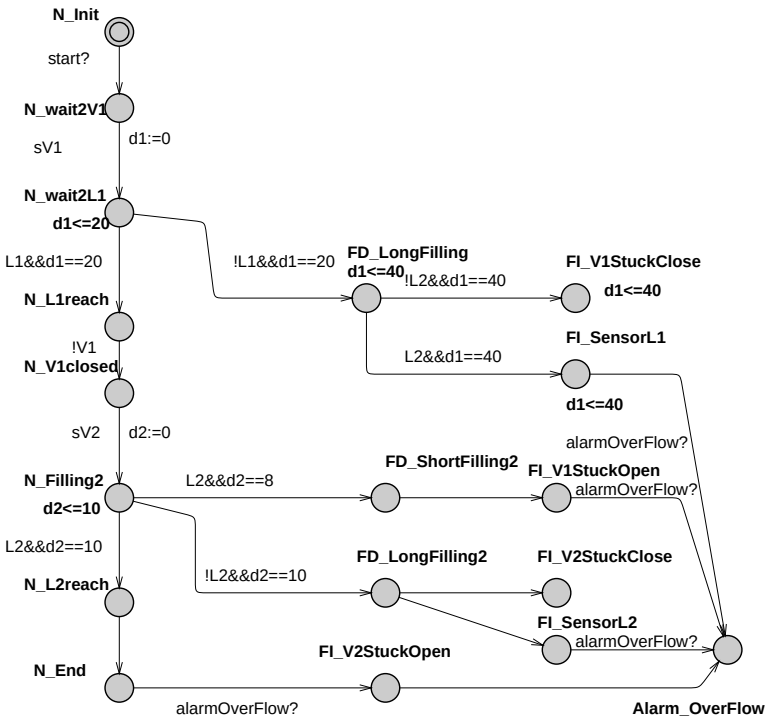
4.5.3.1. *Verification of correctness.* We examine the diagnoser shown in the Figure 4.5.4. The property of our interest is formalized as follows:

id	description
F2	$\mathbf{V_1}$ stuck open
F4	$\mathbf{V_2}$ stuck open
F3	$\mathbf{L_1}$ sensor fault
F5	$\mathbf{L_2}$ sensor fault

TABLE 1. Possible faults for the overflow failure



(a) Plant model



(b) Diagnoser

FIGURE 4.5.4. Alarm treatment - diagnoser

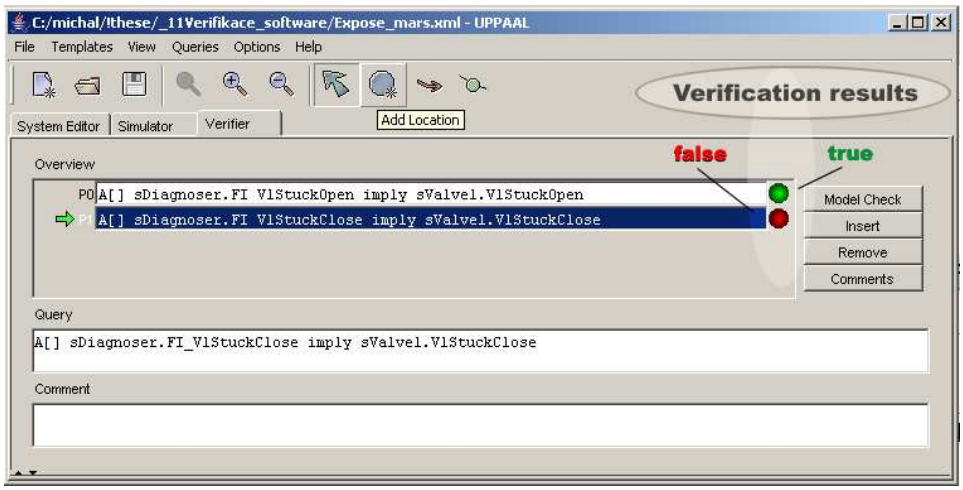


FIGURE 4.5.5. Correctness verification in UPPAAL

$A\Box\mathcal{D}$.V1StuckClose imply $\mathcal{G}$.V1StuckClose. Roughly speaking, it has following meaning: Is every diagnoser result of fault isolation V_1 *stuck close* caused by this fault? The model-checker explored the system model and by diagnostic path tool, we have discovered that there exists also possibility that this fault is announced in some other cases. The executed automaton run found the multi-fault trajectory, where both considered sensors were faulty (sensor being stuck in close position). This multi-fault automaton run in diagnoser is accepted by the diagnoser as a fault V_1 *being stuck close*. With this knowledge, the system could be redesigned to take considered multi-faults into account.

4.5.3.2. *Verification of time-diagnosability.* The fault f_1 ($\mathbf{V_1}$ *stuck close*) will be treated to show a time diagnosability check.

$\mathcal{D}$-Diagnoser which will be verified is shown in Figure 4.5.6. We will verify the property that the fault f_1 is 20-detectable and 40-isolable, *i.e.* 20-40-diagnosable. The automaton run for f_1 is $\rho_1 = senV1 \rightarrow !L1(20) \rightarrow !L2(40)$. The event $!L1$ denotes checking the value of the variable L_1 with the negative answer ($L_1 = 0$ in time $t = 20_{tu}$). The concurrent automaton run for the faulty sensor L_2 is $\rho_2 = senV1 \rightarrow !L1(20) \rightarrow L2(40)$. The projection for fault detection is the same for both runs: $P(\rho, L2) = senV1, !L1$

As it was mentioned, time-diagnosability for f_1 is expected: 20-40-diagnosable. We define the following formulas to check time diagnosability of f_1:

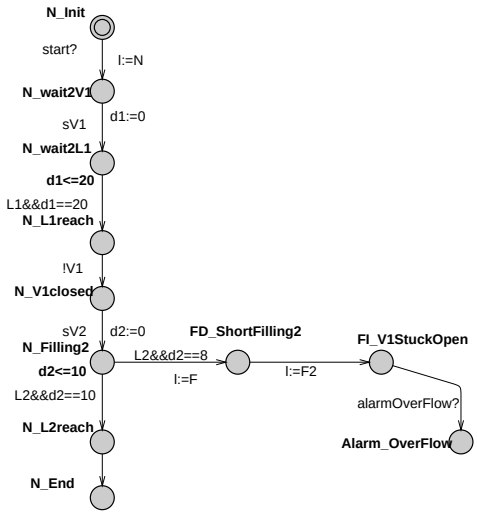


FIGURE 4.5.6. $\mathcal{D}$-diagnoser: 20-40-Diagnosability

(1) *Determine occurrence of fault*: It is possible that f_1 appears after started the program at $time = 0$?

$\mathrm{E}\Diamond\mathcal{D}$.FI_F1 and $\mathcal{D}$.F1 and time==0

(2) *Verify the time of detection* δ: It is possible that f_1 is detected at $time = 20$?

(a) $\mathrm{E}\Diamond\mathcal{D}$.FI_F1 and $\mathcal{G}$.F1 and time<20

(b) $\mathrm{E}\Diamond\mathcal{D}$.FI_F1 and $\mathcal{G}$.F1 and time==20

(3) *Verify the time of isolation* θ: It is possible that f_1 is isolated at $time = 40$?

(a) $\mathrm{E}\Diamond\mathcal{D}$.FI_F1 and $\mathcal{G}$.F1 and time<40

(b) $\mathrm{E}\Diamond\mathcal{D}$.FI_F1 and $\mathcal{G}$.F1 and time==40

One can see the verification results under UPPAAL at Figure 4.5.7. The results confirm our expectation that fault f_1 is 20-40-diagnosable.

4.6. Chapter conclusion

The advantage of model checker comprises three points:

(1) *Component based* modeling creates a dynamic global model $\mathcal{G}$, which is useful for fault diagnosis analysis.

(2) *Alarm treatment* case was treated by model-checking.

(3) *Verification* of $\mathcal{D}$-Diagnoser can confirm *correctness* of the diagnoser construction.

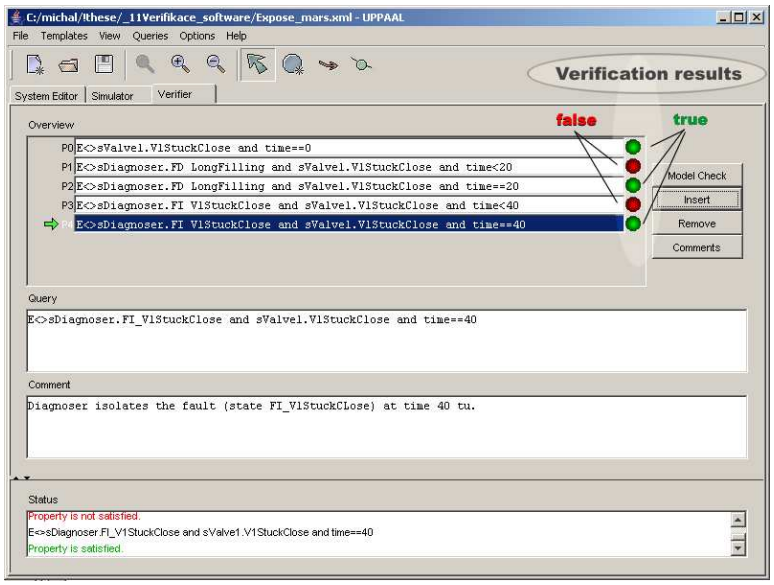


FIGURE 4.5.7. Time diagnosability verification in UPPAAL

Component approach was proposed and shown on examples. Components approach has the advantage in the systematic manner of complex system modeling. The composition (parallel product) of timed automata is called dynamic global model $\mathcal{G}$. This composition creates language of all possible timed traces of the system (infinite).

For alarm treatment diagnosis, the backward time analysis was proposed as an algorithm for the alarm treatment case, *i.e.* an alarm state was detected and a coherent trace was searched. The model checker searched coherent trace firstly by time, secondly by occurrence of event which differentiates the traces.

The diagnoser built has been examined for its correctness. We have found out that the time language $L(\mathcal{D})$ contains faulty traces modeled by $L(\mathcal{G})$. The correctness was checked by model-checking tool, where on-line synthesis of timed automaton components $\mathcal{G}_i$ is possible. The correctness checks the property of implication (reachability check). Briefly, it means that the diagnoser $\mathcal{D}$ announces the fault only if the fault has been already present.

CHAPTER 5

Diagnoser for quantised systems

In this chapter, we extend our focus to continuous variable system, *i.e.* systems with continuous-variable signals. Quantised systems are continuous variable systems viewed as a timed discrete event systems. The dynamic behaviour is imprinted in timed automata. For each fault, it will be obtained one automaton run. The diagnosis algorithm searches the coherent automaton run and fault is detected and isolated by automaton run which suits the observation. The composition of obtained automata has no meaning, as it was the case of parallel composition used in component based modeling. Above this chapter, we have considered a sequential control (sequence, SFC, Grafcet). For quantised system, we do not consider a component based modeling. We consider classical feedback controller (*e.g.*PID) which is difficult to model by TA. Hence, model of quantised systems are related to trajectories describing its dynamics (times of changing discretizated zones). Our focus stays out of hybrid automaton. Hybrid automaton is a state machine augmented with differential equations. It is a standard model for describing a hybrid system. The hybrid automaton models offer an effective way how to imprint the knowledge in the system. We would like to avoid to use the differential equations which could be difficult to implement in PLC. Instead of them we use temporal parameter, which could be seen as a result of differential equation. On the one hand with temporal parameter we must guard if the conditions are still valid for respective temporal parameter. On the other hand, one of the major advantages of quantised system diagnosis by TDES model is that it does not require detailed in-depth modeling of the system to diagnose and hence is ideally suited for the diagnosis of large complex system like manufacturing processes, power plants, etc. Another application areas include automated manufacturing systems like automobile manufacturing where systematic diagnostic procedures are necessary to check equipment integrity before they leave the production line.

L'extension de la démarche aux systèmes à variable continu est traitée dans ce chapitre. Les systèmes appelés "Quantised" sont des systèmes à variables continus discrétisés exploités comme des systèmes discrets temporisés. Le comportement dynamique est représenté par des automates temporisés. Chaque faute, est représenté par une trajectoire dans l'automate. L'algorithme de diagnostic recherche parmi les différents chemins possibles celui qui est cohérent par rapport aux critères temporels. Ainsi la faute est détectée puis isolée grâce à la trajectoire cohérente de l'automate associée à l'observation. Pour les systèmes continue, l'approche basée sur le modèle de composant n'est pas utilisée. Par conséquent, pour ce type de systèmes le modèle est lié à la trajectoire décrivant sa dynamique temporelle (les changements d'état sont liés à la discrétisation). Un des avantages principaux dans le diagnostic de système par TDES est que le modèle n'exige pas une modélisation détaillée pour la construction du diagnoseur. Il est par conséquent bien approprié au diagnostic de systèmes complexes comme les processus de fabrication, les centrales, etc. Notre démarche peut être aussi appliquées à d'autres domaines comme pour les équipements électroniques embarqués où il est nécessaire de développer des procédures de diagnostic systématiques qui permettent de contrôler l'intégrité de matériel avant la mise en service.

5.1. Abstraction of TDES from Continuous model

This section explains how timed discrete-event models can be obtained from a given dynamic systems. It concerns technological systems that have continuous-variable signals. For these systems, discrete events results from an abstraction step, where the continuous movement of the system is replaced by discrete input, state or output jumps. This way of obtaining discrete-event models from continuous-variable systems has been described, using strict formulations, in [**KLSS02, Lun98, SS04**] for a quantised system. The figure 1.3.1 shows a quantised system of which states and outputs are only qualitatively measurable.

Now, let us see the formal definition of continuous system, for which the quantification is considered.

5.1.1. Continuous system [Phi01]. A *continuous system* is a system with the following properties:

(1) time set $T \in \mathbb{R}$,
(2) state space $X \subseteq \mathbb{R}^n$, which is open and connected,
(3) set of inputs and outputs $U \in \mathbb{R}^m$,
(4) the transition map that is induced by a differential equation of the form $[\dot{x}(t) = f(x(t), u(t))]$, $x(t_0) = x_0$ where f is a continuous function in its arguments $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^m$

So, the term "continuous system" refers to the continuity of the state and input space of the system and is not related to the time set T (and thus should not be confused with continuous-time as opposed to discrete-time systems.)

5.1.2. Quantification. The discrete-event behaviour of quantised system is, in general, non-deterministic with respect to the event order and temporal distances of events. Hence, it is reasonable to use a non-deterministic timed automata that allow to cope with the uncertain temporal event distance or non-deterministic recent sequence.

The state transition relation of a timed automata can be directly obtained from a continuous system description. Abstraction algorithms are described in [**SFL03**] or, differently, in [**HGS04**].

The continuous variable system is represented by the state space model $\dot{(x)} = f(x(t), u(t), f), \qquad x(0) = x_0$, where the behaviour of the state vector $x \in (R_n)$ depends on the input vector $u \in (R_m)$ and the fault $f \in \Sigma_f$. Quantisers map the state space into a finite set $Q = 0, 1, 2, \ldots, N$ of qualitative values, *i.e.* partition of R_m into a finite number of disjoint sets $Q_x(i)$ each of which denotes the set of states $x \in R_n$ with the same qualitative value to which they belong. The mapping invoked by the quantisers is represented by $[x(t)] = i \Leftrightarrow x(t) \in Q_x(i)$.

If the state trajectory $x(t)$ crosses the hypersurface, an event is generated. Figure 5.1.1 shows the event e_{ji} with the corresponding hypersurface between partitions.

5.2. Heat Exchanger Application

The section deals with fault diagnosis problem based on the timed automata modelling. We propose that continuous system can be viewed as a time discrete-event system. Knowledge about faultless behaviour can be imprinted by timed automata. This modelling tool has advantage in easy

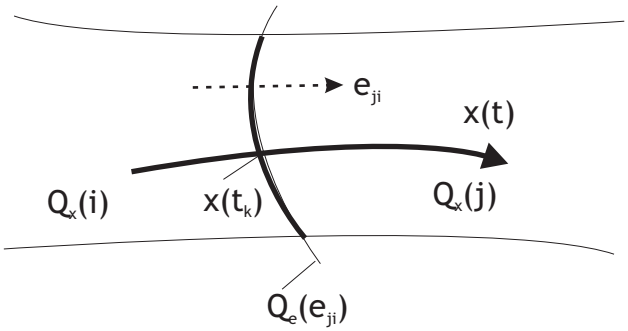


FIGURE 5.1.1. Generation of an event from quantised state space.

implementation in the PLC (by SFC language). The fault detection algorithm guards the zone of the tolerance. The fault identification algorithm is started when the fault detection announce the fault (when the zone of tolerance is reached). In the heat exchanger context, this algorithm checks all automata with all their temperatures and evaluates which dynamics of the real system is coherent with the automaton.

The theory of quantised system was examined on the application of Heat exchanger. Heat exchanger represents the class of application where the system conditions of use are similar and it is needed to store the knowledge of system timing and sequencing.

One of the most important problems for control systems of heat exchangers is the problem of diagnostics and fault identification. The heat exchanger systems include many technological devices that may fault or degrade for several reasons, as fouling of the exchanger, blocking of the valves or just a stuck pump. To reduce the down time, it is important to be able to detect the possible faults and to isolate the problem. Our aim is a sophisticated approach of fault detection and identification to build a system that is more robust. Detection and identification of failures is a critical and complex task.

5.2.1. TDES of a continuous system. Our aim is to obtain a discrete-event model which represents behaviour of continuous dynamic systems. The idea of fault diagnosis of quantised systems is shown at Figure 5.2.1, where the variables of heat exchanger are described.

5.2.2. About HE. The heat exchanger model realised at the laboratory (see Figure 5.2.2) is utilizing ordinary building automation components to build an authentic technological model of a central heating.

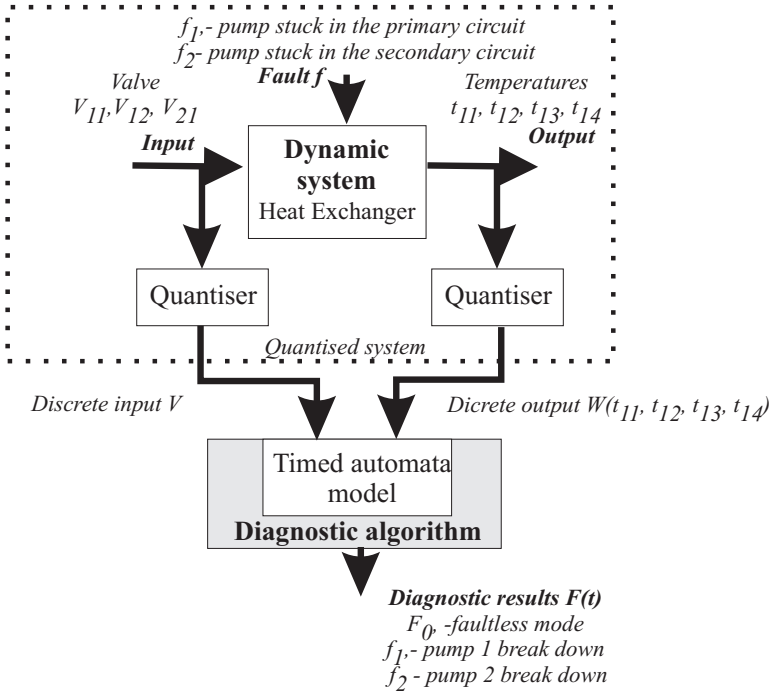
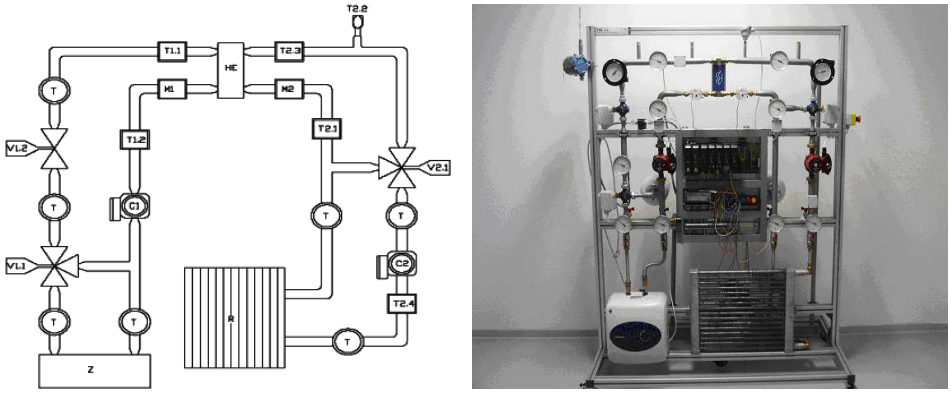


FIGURE 5.2.1. Heat exchanger as quantised system

The model consists of a heat source a 10 liter tank water heater, a plate heat exchanger and an air-cooler. The primary circuit the heater and the primary side of the exchanger consists of a circulation pump, emergency valve and a three-way control valve. The temperature of the water entering the heat exchanger is controlled by the three-way valve that is mixing the hot water from to heater with the water cooled in the exchanger. The secondary circuit the secondary side of the heat exchanger and the cooler consists of circulation pump, three-way control valve and a fan attached to the cooler.

The temperature can be measured at six different locations using external thermometers attached to the pipes. The flows through primary and secondary side of the heat exchanger are measured by two flow meters with pulse outputs. The pressure can be measured at twelve different locations and can be measured as absolute or differential pressure. The available actuating outputs are on-off signals for the pumps and the fan and three analogue values controlling the positions of the servo valves.

5.2.2.1. *Modelling.* The plate heat exchanger itself is modelled in Matlab/Simulink to provide easier and faster way for testing the effect of different faults. The model is based on first principle modelling and takes



(a) Schematic (b) Photo of heat exchanger model

FIGURE 5.2.2. Heat exchanger model

Parameter	Value
Plate thermal conductivity	$16,3\mathrm{Wm}^{-1}\ \mathrm{K}^{-1}/23°C$
Plate thickness	$0,35\mathrm{mm}$
heat transfer area	$0,2\mathrm{m}^2$
Number of plates	14
Effective plates	12
Number of passes	1
Relative fluid direction	Counter-current
Length x width x height	$33 \times 78 \times 208\mathrm{mm}$
Liquid volume	$0,1\mathrm{dm}^3$

TABLE 1. Heat exchanger parameters

into account heat transfer from one media (hot water) into another (cold water). The properties of the modelled plate heat exchanger (Alfa-Laval CB14-HVC) used in the laboratory model are summarized in the Table 1.

The modeling is based on the equality between the heat disposed by the hot fluid and the heat accumulated in the wall and gained by the secondary fluid. The model of the heat transfer through the wall is a basic task and is not important for the HE model definition. But it should be noted that the accumulation of heat has significant effects on the model dynamics. For the modeling, the exchanger is divided into segments along the flow path. In each block, the fluid is treated as being static, the flow happens only between the blocks. The model is implemented in the Matlab/Simulink, where the equations for each segment are implemented as one block and the heat exchanger is then combination of several of these blocks. The

block architecture makes it possible to construct parallel flow or counter flow models only by changing the interconnection. Details on the modeling can be found in [**LCH05**].

5.2.3. Diagnoser. The construction of timed automata is processed by the determination task of the dynamic systems. The quantification does not only detect events but also considers the time at which each event occurs. This knowledge of events and time is imprinted into the timed automata. The determination was described in details in the following papers *e.g.* [**SFL03, HGS04**].

5.2.3.1. *Fault detection algorithm.* Fault detection algorithms are based on the TA model. The fault changes the event sequence or the temporal distances between events [**HLS04**]. The on-line algorithm works roughly as follows: In each step we observe the time and incoming events. If the time pass over the specific time interval defined for faultless mode, fault is announce by an alarm. If an incoherent event incomes, the alarm could be also activated.

The task of fault detection and identification is difficult task in the setting of the temporal parameters that are realised in the timed automata. It must be considered a tuning of the sensitivity and reaction time of algorithm. If the sensitivity is tuned very fine, an alarm could be activated in the faultless case.

5.2.3.2. *Fault isolation.* In the case of fault identification, we implement the knowledge of considered faults. The model must contain the qualitative description of faulty evolution. This qualitative model can be obtained mathematically or by simulation as well. The fault is modelled as an unobservable fault. The faulty evolution is derived from the faultless trajectory.

5.2.3.3. *Diagnosis for system under control.* Parameters to observe are heat exchanger temperatures, see Figure 5.2.1. Considered fault will propagate in all temperatures, both faults are distinguishable (diagnosable) by different projection (trajectory). Our task is to detect the trajectory violation and identify the corresponding behaviour with the faulty automaton run. The temperatures are quantised. For the faultless mode, the region of tolerance is defined. This region was identified by simulation where all working conditions were simulated. If the temperature increases or decreases

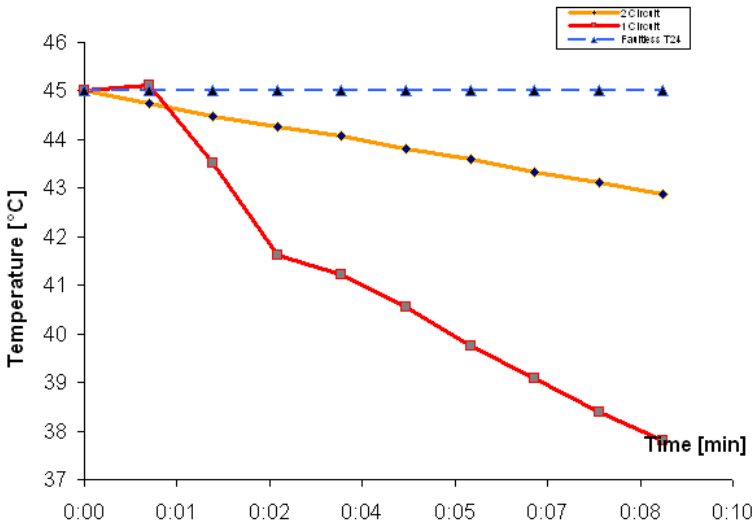
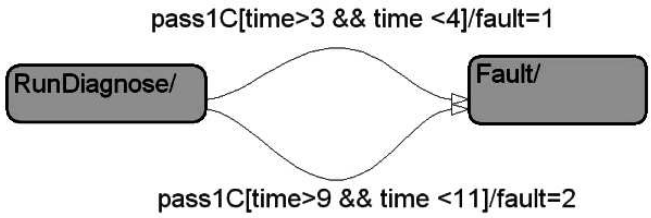


FIGURE 5.2.3. Evolution of parameter T_{24} for fault f_1 and f_2

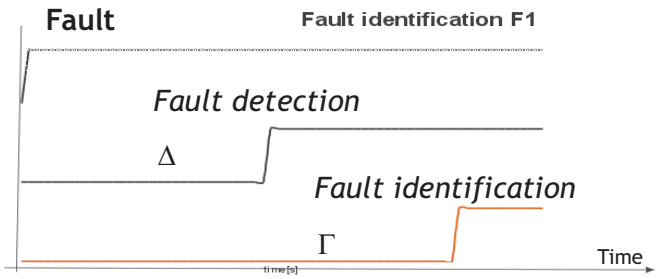
beyond the respective margin, the process of fault identification is started. Temperatures are observed to diagnose the scope and tendency of the faulty evolution, see the simulated evolution on Figure 5.2.3. One can see that this trajectory evolution is distinguishable. This observer is implemented as a TA.

From the Figure 5.2.3, one can see that the stuck of the pump in primary circuit causes the rapid cooling of water in second circuit (compare with stuck of the pump in second circuit). This knowledge can be implemented as follows: in the defined interval, we examine the temperature. Timed automata implementation is shown in Figure 5.2.4.(a). The event `pass1C` is produced when the temperature decreases 1°C below the tolerance region. This event identifies the fault f_1 when this event appears in time interval [30,40]s. The fault identification checks consistency of all automata (all temperatures evolution), like the one described for T_{24}. On Figure 5.2.4.(b), time delay to detect and identify the fault (equivalent with δ and θ diagnosability) is shown.

5.2.4. Implementation. We have obtained the TA model by the trajectory determination. This timed model can be implemented by different PLC languages. According to simplicity of implementation of TA, sequential function chart (SFC) is proposed. There, we can combine the sequence programming with guarding the time intervals.



(a) TA for fault identification



(b) Time dependency: Fault Occurence, Detection, Isolation

FIGURE 5.2.4. Fault Isolation in heat exchanger

The fault detection and identification is implemented in Delta V programming tool. Thanks to the advantages of sequential function chart (SFC) programming, the timed automata could be implemented.

The final implementation consists of the following SF charts with functions:

- guard the start up system
- watch dog: if the zone of tolerance was overtaken, the diagnostic module is started
- diagnostic module to fault identification
- fault decision according to the diagnostic results (voting system)

5.3. Chapter conclusion

In the context of continuous-variable system, we have mentioned an existing direction of TA usage. This quantised system approach was demonstrated on Heat exchanger application. The results show that diagnosis without in-depth system knowledge can be used, but with some limitations: the state space of continuous system is infinite and by discretization we make the state space finite. As it is known, the trajectories of continuous system depend on the system dynamics and on the controller. For some

restrictive conditions, the continuous system behaves properly according to the discrete model (timed automata). This restriction can formulated according to functioning mode. When the restriction to the system makes a discrete model of reasonable size, diagnostics by a TDES model (timed automata) is suitable to apply.

CHAPTER 6

Conclusion

This thesis dealt with the fault diagnostics of timed system. Our aim was the extension of classical diagnoser approach to taking time into account [**Kno05a**]. Generally, time extension causes the precision of diagnostics result because the important aspect (time) is taken into account. We have shown in chapter 2 that time consideration permit us to obtain diagnostic results for system, for which no diagnosis result (un-diagnosability) is obtained for "no-timed" context. On the other hand, not every system is diagnosable. If the observed events have the same projection into model, the fault is un-diagnosable in timed as in non-timed framework. In this thesis, we have calculated two parameters to verify if the system instrumentation permit us detect and isolate the fault. We have defined *timed systems* as system which behaviour depends on its timing; because this definition is too wide, we have considered two classes: time discrete-event systems (TDES) and quantised system.

Main focus of the thesis is given to timed system which fall in the class of TDES. In this system we have considered a sequential control which can be described by timed automata. In this context, we have treated the problem of diagnoser design and consecutively its verification. Timed systems which represent continuous-variable system have been treated partially in the Chapter 5. This timed systems are called quantised system. Quantised systems are related to classical controller known (*e.g.* PD,PID etc.) from control theory, TDES systems are related to sequential controller (language SFC, GRAFCET).

For timed systems was searched the diagnoser. This diagnoser building has been based on determination of faulty behaviour to obtain a TA model containing faultless and faulty traces. This diagnoser in the form of one TA we called $\mathcal{D}$-Diagnoser. Algorithm of Diagnoser building described how to obtain diagnoser in three parts: 1.) determination of faultless trajectories 2.) fault detection transitions and 3.) fault isolation transition[**KSA05,**

Kno05b, KSAA05]. The advantage of $\mathcal{D}$-Diagnosis design is in timed automaton model which can be translated to a language of industrial automata. It makes it attractive for simple industrial usage.

Thus, we have initiated the fault diagnosis by model-checker. We have shown a modeling of system based on components. The composition of all components together with time makes a huge state space, therefore synthesising is calculated in special software, called model-checker. First aim was formal verification of diagnoser by model-checking. This comparison can be done for any diagnoser which is possible to model by TA. Described principle can by applied for *e.g.* verification of existing diagnosers, different diagnoser design and re-design. We have used diagnoser as one of the component and we have checked reachability of diagnoser states and model states. Practically, diagnoser verification help us checking the diagnoser and also its implementation. Another advantage is exploration and verification of multi-faults trajectories. It is important to know that diagnoser signals false error for respective multi-fault occurrence. With this knowledge the diagnoser can be re-built or system re-designed (*e.g.* add sensor) [**KSAZ06**].

Thus, we have also illustrated that model-checker can be useful tool for fault diagnosis. We have chosen off-line diagnosis, described as alarm treatment. When the alarm is announced, the diagnosis process looks into the cause. In this context, we have presented backward time analysis for this purposes. Main disadvantage of this method stays in model-checker implementation for system in industrial size, where a huge number of models makes a verification time consuming process. On the other hand, the effort in this domain is still big and the usage for industrial size systems can be question of the time. One of the promising result is related to model-checker realisation *e.g.* UPPAAL TRON can be run as routine on server which verify the model on-the-fly (on-line) [**sUT, LMN04a**]. One of the perspective, we consider in this logic of diagnosis: The observation is on-the-fly verified against model to decide which fault has been occurred in the system.

Finally, we have given an effort to real application on quantised systems: Heat Exchanger. [**KJH+06**] There, we have proven that diagnosis mechanism can be build but with certain limitation. To obtain suitable model of TA, we must chose a compromise between precision and size of the discretized model. For some restrictive conditions, the continuous system

behaves properly according to the discrete model (timed automata); *e.g.* in heat exchanger application we have supposed the control on one output value (temperature in the secondary circuit is fix). Algorithm worked properly in this operation mode, but if the control is change the real dynamics could not be coherent with the model.

Maintenance and supervision are the two logical extensions of this work. For the maintenance, it consists to exploit the diagnosis analysis results to implement the predictive maintenance. Indeed, as the implementation of a predictive maintenance requires data on the degradation state of the equipment. The diagnosis step will provide then information on the state of the process which constitutes thus the whole of the data input of the maintenance.

This development can also be an input for the implementation of supervision systems for industrial installation. The supervision of an industrial process consists in managing the data flow coming from the parts operative and control part. The diagnosis would be an effective interface as well to direct the maintenance actions as to supervise the whole installation.

Ce travail de thèse a permis de proposer une nouvelle forme de modélisation et d'analyse des systèmes à évènements discrets pour le diagnostic. Le modèle proposé permet de prendre en compte la dynamique du système grâce l'utilisation des automates temporisés et à l'exploitation du temps. Deux formes de modélisation sont proposées :

- *Une modélisation globale qui prend en compte l'ensemble des états du système*
- *Une modélisation par composant qui prend en compte le fonctionnement en présence de fautes de chaque élément.*

L'approche de diagnostic par analyse arrière est proposée. Une faute est détecter et isolée grâce à l'exploitation due modèle globale et la date d'occurrence de l'alarme. L'algorithme permettant la construction du diagnoseur est défini. Il est basé sur l'identification des trajectoires du modèle représentant le fonctionnement normal enrichi par l'ensemble des chemins menant aux états de fautes. La validation du modèle est une essentielle dans ce travail. Elle permet de vérifier l'atteignabilité des états de fautes dans le modèle. Le principe du "model-checking" est utilisé par comparaison du modèle $\mathcal{G}$ et du $\mathcal{D}$-diagnoseur.

L'approche développée peut être étendue à d'autres types de systèmes comme les systèmes hybrides et ouvre d'autres perspectives. Nous pouvons citer :

- *l'optimisation des temps de détection et d'isolation*
- *l'application aux systèmes continus*
- *validation sur un exemple industriel*
- *diagnostic en ligne basé sur le model checking*
- *exploitation des résultats de diagnostic pour la maintenance.*

Bibliography

[AD90] R. Alur and D. Dill, *Automata for modeling real-time systems*, 17th Int.Coll. Automata, Languages, and Programming (ICALP'90), Warwick University, England, vol. 443 of Lectures Note in Computer Science, Springer, july 1990, pp. 322–335.

[AD94] R. Alur and D.L. Dill, *A theory of timed automata*, Theoretical Computer Science (TCS) **126** (1994), no. 2, 183–235.

[BGBDS04] M-M. Ben Gaid, B. Bérard, and O. De Smet, *Modélisation et vérification d'un évaporateur en* UPPAAL, Actes du 6ème Atelier Approches Formelles dans l'Aide au Développement de Logiciels (Besançon, France), Juin 2004.

[BKLS03] M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki, *Diagnosis and fault-tolerant control*, Springer Verlag, 2003.

[BLPM00] P. Baroni, G. Lamperti, P.Pogliano, and M.Zanella, *Diagnosis of the class of distributed discrete-event systems*, IEEE Transitions on Systems, Man, and Cybernetics **30** (2000), no. 6, 731–752.

[BY04] Johan Bengtsson and Wang Yi, *Timed automata: Semantics, algorithms and tools*, In Lecture Notes on Concurrency and Petri Nets. W. Reisig and G. Rozenberg (eds.), LNCS 3098, Springer-Verlag, 2004.

[CGLP03] M.O. Cordier, A. Grastien, Ch. Largouët, and Y. Pencolé, *Efficient trajectories computing exploiting inversibility properties.*, Proceedings of the Fourteenth International Workshop on Principles of Diagnosis, 2003, pp. 93–98.

[CMS03] V. Cocquempot, T. El Mezyany, and M. Staroswiecki, *Switching time estimation and fault detection for hybrid systems using structured parity residuals*, IFAC Conference Safeprocess 2003 (Washington, USA), December 2003, pp. 2045–2055.

[CSM04] V. Cocquempot, M. Staroswiecki, and T. El Mezyany, *Surveillance des systèmes hybrides non linéaires à l'aide de relations de redondance analytiques*, CIFA 2004 Conference Internationale Francophone d'Automatique (Douz, Tunisie), December 2004.

[FLMM00] A. Fanni, M. Lera, E. Marongiu, and A. Montisci, *Neural network diagnosis for visual inspection in printed circuit boards*, In Proc. DX00 11th Int. Workshop on Principles of Diagnosis (Via Lattea, Italy), 2000.

[Fra87] P.M. Frank, *Fault diagnosis in dynamic systems via state estimation – a survey*, System Fault Diagnostics, Reliability and Related Knowledge-Based Approaches. (D.Reidel Publ.Co), vol. 1, 1987, pp. 35–98.

[Fre71] D. Freedman, *Markov chains.*, Holden-Day Series in Probability and Statistics. (Holden-Day. San Francisco), 1971.

[HC94] L. Holloway and S. Chand, *Time templates for discrete event fault monitoring in manufacturing systems*, 1994.

[HGS04] A. Hélias, F. Guerrinand, and J.-P. Steyer, *Abstraction of continuous system trajectories into timed automata*, Workshop on Discrete Event Systems WODES'04 (Reims, France), IFAC, 2004, pp. 319–324.

[HLS04] I. Hristov, J. Lunze, and P. Supavatanakul, *A time discrete-event approach to diagnosis of the damadics actuator benchmark*, 5th DAMADICS Workshop, April 2004.

[HNJY94] T.A. Henziger, X. Nicollin, J.Sifakis, and S. Yovine, *Symbolic model checking for real-time systems*, Journal of Information and Computation **111(2)** (1994), 193–224.

[HU79] J.E. Hopcroft and J.D. Ullman, *Introduction to automata theory*, Languages and Computation. (Addison-Wesley Series in Computer Science.Addison-Wesley.), 1979.

[IEC00] IEC-61508, *Functional safety of electrical/electronic/programmable electronic safety - related systems, part 1 to part 6*, 2000.

[IEC05] IEC-61508-0, *Functional safety and iec 61508*, Working draft of IEC TR 61508-0 (Geneva, Switzerland), 2005.

[Ise84] R. Isermann, *Process fault detection based on modeling and estimation methods – a survey*, Automatica **20** (1984), no. 4, 387–404.

[KJH+06] M. Knotek, P. Jedlicka, O. Hyncica, Z. Simeu-Abazi, and F. Zezulka, *Fault diagnosis applied on the het exchanger*, In Proceedings of IFAC Workshop: Programmable Devices and Embedded Systems PdeS'06 (Brno, Czech Republic), 2006.

[KLSS02] V. Krebs, J. Lunze, G. Schullerus, and P. Supavatanakul, *Relations of timed event graphs and timed automata in fault diagnosis*, Research report Bericht Nr. 2002-21, October 2002.

[Kno05a] M. Knotek, *Bibliography survey on a diagnostic of discrete-event systems*, Tech. Report NI05-020, Laboratoire d'Automatique de Grenoble, France, 2005.

[Kno05b] Michal Knotek, *Fault diagnosis based on backward time analysis*, Student EEICT 2005 (Brno, Czech Republic), 2005.

[KS98] Stefan Kowalewski and Olaf Stursberg, *The batch evaporator: A benchmark example for safety analysis of processing systems under logic control*, Proceedings 4th Workshop on Discrete Event Systems (WODES), IEE, London, 1998, pp. 302–307.

[KSA05] M. Knotek and Z. Simeu-Abazi, *Fault location by time analysis*, In Proc. IESM'05 (Marrakech, Marocco), may 2005.

[KSAA05] M. Knotek, Z. Simeu-Abazi, and H. Alla, *Diagnosis based on backward time analysis*, Actes du Congrès: Qualité et Sûreté de Functionnement (Bordeaux, France), 2005, pp. 255–261.

[KSAZ06] M. Knotek, Z. Simeu-Abazi, and F. Zezulka, *Fault diagnosis based on timed automata: Diagnoser verification*, In Proc. CESA'06 (China), oct 2006.

[LCH05] F. Zezulka L. Cerný and O. Hynčica, *The diagnostics of heat exchanger*, In Proceedings of 6th International Carpathian Control Conference. (ISBN 963-661-645-0, ed.), Miskolc, Hungary, ICCC 2005., 2005.

[LMN04a] K. Larsen, M. Mikucionis, and B. Nielsen, *Online testing of real-time systems using uppaal: Status and future work*, Dagstuhl Seminar. Proceedings volume 04371. Perspectives of Model-Based Testing (Schloss Dagstuhl, Wadern, Germany), 2004.

[LMN04b] Kim G. Larsen, M. Mikucionis, and B. Nielsen, *Online testing of real-time systems using uppaal*, Formal Approaches to Testing of Software (Linz, Australiea), 2004.

[LN03] J. Lunze and J. Neidig, *A new approach to distributed diagnosis using stochastic automata networks*, Research report 2003.17, October 2003.

[LS01] J. Lunze and J. Schröder, *Bridge benchmark problem for diagnosis of discrete-event systems*, Forschungsbericht, Arbeitsbereich Regelungstechnik, TU Hamburg-Harburg, 2001.

[LSS00] J. Lunze, F. Schiller, and J. Schröder, *Diagnosis of transient faults in quantised systems*, Proceedings SAFEPROCESS (Budapest, Hungary), IFAC, 2000, pp. 1174–1179.

[LSS01] J. Lunze, J. Schröder, and P. Supavatanakul, *Diagnosis of discrete event systems: the method and an example*, Proceedings of the Workshop on Principles of Diagnosis, DX'01 (Via Lattea, Italy), 2001, pp. 111–118.

[Luk99] Ben Lukoschus, *An abstract model of VHS case study 1 (experimental batch plant)*, Tech. Report (number to be assigned), Christian-Albrechts-Universität zu Kiel, 1999.

[Lun98] J. Lunze, *Qualitative modelling of dynamical systems: motivation, methods, and prospective applications*, Mathematics and Computers in Simulation **46** (1998), 465–483.

[MR00] D. Maquin and J. Ragot, *Diagnostic des systèmes linéares*, Hermes Science Publications, 2000.

[MS03] M. Mikucionis and E. Sasnauskaite, *On-the-fly testing using uppaal*, Master thesis, 2003.

[Ols93] G.J. Olsder, *On structural properties of min-max systems.*, Report 93–95 Bericht Nr. 2002-21, Delft University of Technology., 1993.

[PD00] A. Panati and D. Theseider Dupr(e), *Causal simulation and diagnosis of dynamic systems*, In Proc. DX00 11th Int. Workshop on Principles of Diagnosis (Via Lattea, Italy), 2000.

[PH00] D.N. Pandalai and L.E. Holloway, *Discrete-event models of quantised systems for diagnosis*, IEEE Transitions on Automatic Control **45** (2000), no. 5, 868–882.

[Phi01] P.P.H.H. Philips, *Modelling, control and fault detection of discretely-observed systems.*, Ph.D. thesis, Technische Universiteit Eindhoven, 2001, 2001, ISBN 90-386-1729-1.

[RC02] L. Rozé and M.O. Cordier, *Diagnosing discrete-event systems: Extending the "diagnoser approach" to deal with telecommunication networks*,

Discrete Event Dynamic Systems: Theory and Applications (2002), no. 12, 43–81.

[Rei85] W. Reisig, *Petri nets: An introduction.*, of Monographs in Theoretical Computer Science, vol. 4, Springer Verlag, New York, USA, 1985.

[RW87] P. Ramadge and W. Wonham, *Supervisory control of a class of discrete-event processes*, SIAM J.Control Optim. **25** (January 1987), no. 1.

[Sav01] Alexandru Tiberiu Sava, *Sur la synthése de la commande des systémes á événements dicrtets temporisés*, Ph.D. thesis, Laboratoire d'Automatique de Grenoble, november 2001.

[SFL03] P. Supavatanakul, C. Falkenberg, and J. Lunze, *Identification of timed discrete-event models for diagnosis*, International Workshop on Principles of Diagnosis (DX'03), June 2003.

[SPVJ02] G. Schullerus, P.Supavatanakul, V.Krebs, and J.Lunze, *Efficient hierarchial diagnosis for timed dicrete-event systems*, Forschungsbericht Bericht Nr. 2002.4, Ruhr-Universitat Bochum, June 2002.

[SS04] P. Supavatanakul and G. Schullerus, *A hierarchical heterogeneous approach to diagnosis of discrete-event systems*, Research report 2004.05, 2004.

[SSL+95] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis., *Diagnosability of discrete event systems.*, IEEE Transactions on Automatic Control **40** (1995), no. 9, 1555–1575.

[SSL+96] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, *Failure diagnosis using discrete event models.*, IEEE Transactions on Control Systems Technology **4** (1996), no. 2, 105–124.

[sUT] Uppaal TRON, *http://www.cs.aau.dk/ marius/tron*.

[Tha00] Dang Thi Xuan Thao, *Vérification et synthèse des systèmes hybrides*, Ph.D. thesis, Institut National Polytechnique de Grenoble, France, 2000.

[Tri98] S. Tripakis, *L'analyse formelle de systèmes temporisés en pratique.*, Ph.D. thesis, Université Joseph Fourier, Grenoble, France, 1998.

[Tri02] Stavros Tripakis, *Fault diagnosis for timed automata*, In Proc. 7th Int. Symp. Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT 02) (Springer, ed.), vol. 2469 of LNCS, 2002, pp. 205–224.

[Uppaal] Uppaal, *www.uppaal.com*.

[Yov93] S. Yovine, *Méthodes et outil pour la vérification symbolique des systèmes temporisés.*, Ph.D. thesis, VERIMAG – Institut National Polytechnique de Grenoble, 1993.

[Yov97] Sergio Yovine, *Kronos: a verification tool for real-time systems.*, Journal on Software Tool for Technology Transfer **1** (1997).

[Zad99] S.H. Zad, *Fault diagnosis on dicrete-event and hybrid systems*, Ph.D. thesis, University Of Toronto, 1999.

[ZKW99] S.H. Zad, R.H. Kwong, and W.M. Wonham, *Fault diagnosis in finite-state automata and timed discrete-event systems.*, in Topics in Control and its Applications: A Tribute to Edward J. Davison, D.E. Miller and L. Qiu, Springer Verlag, 1999, pp. 81–105.

Index

Curriculum Vitae

Personal

Name:	Ing. Michal Knotek
Birth date:	2.7.1979
Nationality:	Czech
E-mail:	michal.knotek@gmail.com

Education

2003–2006	*Doctor of automation and production* PhD. specialization: Fault diagnosis in automation. Joint supervision between Laboratoire d'Automatique Grenoble France and Brno University of Technology Czech Republic
1998–2003	*Diplom of engineer – Automation, measurement and cybernetic.* Brno University of Technology, CZ; Results: excellent
1993–1998	High school Videnska, Brno; oriented on computer science

Professional experiences

- System engineer at Honeywell, Brno Czech Republic: Fault isolation design for Boeing 787 : since 2006
- Ph.D study: Research and development on fault diagnosis, teaching: 2003–2006.
- Automation with PLC: Certificate Siemens A/D Germany: 2005.
- Diploma thesis at UJF Grenoble (Robust control on Simatic S7-300): 2003.
- Development, design of database information system. Artisys s.r.o, Brno: 1998–2002

Languages

English: fluently; French: fluently; German: passive understanding; Russian: basic knowledge; Czech: maternal;

Introduction to Uppaal

UPPAAL [UPPAAL] is a toolbox for symbolic simulation and automatic verification (via automatic model checking) of real time systems modeled as networks of extended timed automata. The tool provides reachability analysis for automatic verification of properties for real-time systems. It contains a number of additional features including graphical interfaces for editing and simulating system models. The UPPAAL model checker tool consists of the programs: graphical user interface (GUI) and the model checker engine (server). UPPAAL GUI provides cross-platform user interaction build on Java. UPPAAL server provides an efficient computation of a system symbolic state after given transition and verification result for a given property. The UPPAAL GUI (re)starts the server program whenever user chooses to update simulator and/or verifier with a new system model. The GUI and the server communicate through TCP/IP socket connection established automatically after the server start. The UPPAAL GUI has three parts [**MS03**]:

The *System Editor* allows the user to describe and edit the timed-automata system. The system timed automata consists of global declarations, a timed-automaton templates, process assignment and system definition sections.

The *Simulator* allows the user to virtually interact with the system described. The simulator shows the system state by displaying the states of compound automata and the values of variables. The simulator allows the user to choose enabled transitions manually or randomly. It also has a feature of displaying the history of events in sequence chart.

The *Verifier* accepts the user formulated properties to be verified on a particular timed- automata model, and displays the result of verification: true or false depending on whether the property was satisfied or not, and an event trace example if the property proof requires one.