Modeling Patterns for Performance Analysis of Production and Safety Systems in Process Industry

Huixing Meng

NNT : 2017SACLX074

THESE DE DOCTORAT
DE
L'UNIVERSITE PARIS-SACLAY
PREPAREE A
L'ÉCOLE POLYTECHNIQUE

ECOLE DOCTORALE N° 573
Interfaces: approches interdisciplinaires, fondements, applications et innovation

Spécialité de doctorat: Informatique

Par

M. Huixing MENG

Modélisation des patterns d'analyse des performances des
systèmes de production et de sûreté de fonctionnement dans l'industrie des procédés

Modeling Patterns for Performance Analysis of
Production and Safety Systems in Process Industry

Thèse présentée et soutenue à l'Ecole Polytechnique, le 17 novembre 2017

Composition du Jury:

M. Dominique BARTH	Université de Versailles Saint-Quentin-en-Yvelines	Président/ Examinateur
M. Jean-Yves CHOLEY	Supméca - Institut supérieur de mécanique de Paris	Rapporteur
Mme. Mitra FOULADIRAD	Université de Technologie de Troyes	Rapporteur
Mme. Leïla KLOUL	Université de Versailles Saint-Quentin-en-Yvelines	CoDirecteur de thèse
M. Antoine RAUZY	Norwegian University of Science and Technology	Directeur de thèse

Long Abstract in French

Titre: Modélisation des patterns d'analyse des performances des systèmes de production et de sûreté de fonctionnement dans l'industrie des procédés

Mots clés: Pattern, Performance, Fiabilité, Disponibilité, AltaRica

Résumé long: Les systèmes de production et de sûreté de fonctionnement sont d'une importance majeure dans l'industrie des procédés. Leurs performances impactent directement les intérêts de l'industrie. En effet, ces systèmes ont des comportements similaires. Ces comportements peuvent être conceptualisés dans des modèles via des patterns de modélisation. La réutilisation de ces patterns permet de rendre le processus de modélisation à la fois simplifiée et plus efficace.

Dans ce projet de thèse, les patterns de modélisation sont étudiés selon plusieurs axes, à savoir la forme de capitaliser les patterns (catalogue), l'approche pour développer ces patterns (méthodologie) et l'aspect expérimental pour tester ces modèles.

Nous proposons une approche basée sur des modèles d'analyse de la performance des systèmes de production et de sûreté de fonctionnement dans l'industrie des procédés. Nous nous somme particulièrement concentrés sur les systèmes pétroliers et gaziers. Les patterns de modélisation sont capables de stabiliser l'expérience à partir du processus de modélisation (expérience). Trois principaux travaux sont réalisés dans cette thèse:

- Nous avons étudié les patterns de modélisation suivant plusieurs aspects; Nous les avons classés en fonction de leurs finalités, ce qui reflète la fonction d'un pattern spécifique de modélisation. Les schémas de modélisation sont classés en comportementales, propagation de flux, et des patterns de composition. Chaque type de pattern de modélisation est illustrée à l'aide des Guarded Transition Systems (GTS). Une méthodologie pour développer des patterns de modélisation est proposée.

- Nous avons établi un catalogue de patterns de modélisation. En examinant de nombreux systèmes de production et de sûreté de fonctionnement, nous avons proposé 24 patterns de modélisation comprenant 8 patterns comportemantals, 13 patterns de propagation de flux et 3 patterns de composition. Chaque pattern est représenté avec un ensemble d'éléments structurés (p. ex., de structures et de modèle exemple). Les structures de patterns comportementals et ceux de composition sont illustrés avec des représentations graphiques de GTS. Cependant, Les structures de flux de propagation sont représentés avec leurs graphes de dépendance. Des modèles exemples sont donnés avec le programme AltaRica 3.0 où les intéractions entre les patterns de modélisation sont aussi rapportées. Nous avons également discuté de la méthodologie pour réutiliser ces patterns de modélisation.
- Nous avons mené des études expérimentales sur un ensemble des systèmes de production et de sûreté de fonctionnement qui ont été extraits de la littérature. Les systèmes de production sont censes couvrir l'ensemble des difficultés de modélisation pour l' analyse de la pérformance de la production. Les systèmes de sûreté de fonctionnement sont composés d'un groupe de systèmes dans ISO/TR 12489, qui inclut la majorité des problèmes de fiabilité. Pour chaque système, nous avons présentés sa description, ses propre patterns de modélisation et les résultats expérimentaux correspondants. Les résultats expérimentaux obtenus en déployant les patterns de modélisation convergent vers les résultats rapportés dans la littérature. Il est démontré que les patterns de modélisation suggérés proposés sont applicables aux systèmes de production et de sûreté de fonctionnement des modèles dans l'industrie des procédés.

Acknowledgment

First of all, I would like to express my sincere thanks to my advisers, Professors Antoine Rauzy and Leïla Kloul, for offering me this precious opportunity to conduct the research presented in this thesis. Without their patient guidance and insightful suggestions during the past three years, this dissertation would not have been possible.

I would also like to thank Professors Dominique Barth, Jean-Yves Choley, and Mitra Fouladirad, who kindly accepted to be the jury members of my thesis. They took enormous effort to carefully examine or review my thesis.

I would also like to thank my colleagues in the AltaRica team for their help. Besides my supervisors, their names are Michel Batteux, Jean-Marc Roussel, Tatiana Prosvirnova, Benjamin Aupetit, Anthony Legendre, Melissa Issad, Loïc Peletan, Benoît Lebeaupin, Walid Bennaceur, Abraham Cherfi, and Pierre-Antoine Brameret.

I would also like to thank Cyrille Folleau (SATODEV), Stéphane Collas and Jean-Pierre Signoret (TOTAL) for their invaluable advice.

I would also like to thank my colleagues and friends in the building Turing, whose names are Mireille Regnier, Evelyne Rayssac, Vanessa Molina Magana, Ruqi Huang, Afaf Saaidi, Guangshuo Chen, William George, Fei Song, Dorian Nogneng, Emmanuel Haucourt, and Juraj Muchalix.

I would also like to thank the laboratory DAVID (formerly PRiSM) at UVSQ. I worked there about once a week in the past three years.

I would also like to thank Alexandra Belus, Emmanuel Fullenwarth, and Audrey Lemaréchal (Graduate School) and Nathalie Legeay (Administration of international scientists) at École Polytechnique for their invaluable help.

I would also like to thank the financial support from the China Scholarship Council.

I would also like to thank all my teachers and friends in my 24-year student life.

Last but not least, I owe special thanks to my family, especially to my parents and my wonderful beloved Jinduo, for their consistent encouragement and support in the past (and following) years.

Huixing

Paris, 16/11/2017

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The target of this thesis is to develop a pattern-based approach for performance analysis of production and safety systems in process industry. We focus particularly on oil and gas systems. Indeed, these systems share common behaviors. Modeling patterns allow to capitalize the experience from modeling systems. By reusing modeling patterns, a modeling mission can be simplified when evaluating the performance of these systems.

Topics of this thesis are illustrated by the Venn diagram in Figure 1.1: production and safety systems, performance analysis, as well as modeling languages and methodologies. The focus locates at the intersection of these three sets.

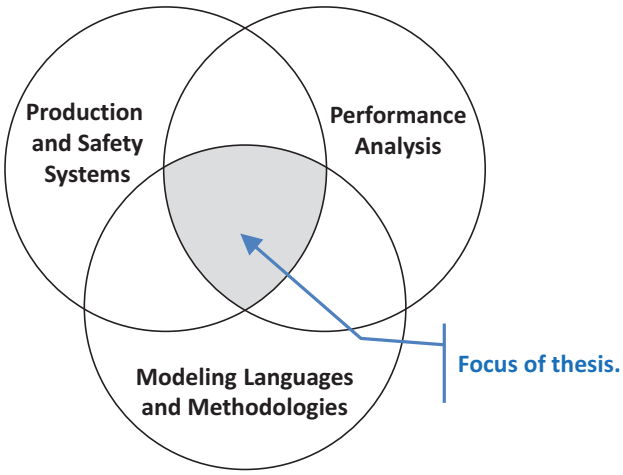


Figure 1.1: Venn diagram of thesis topics.

In the following, we introduce the context, the problem formulation, and the objective of the thesis. The context lies in the domain of performance analysis of production and safety systems. We try to address the problem of simplifying a modeling mission by reusing modeling knowledge. Accordingly, to reduce the modeling burden, we aim at proposing a pattern-based approach for performance analysis.

1.1 Production and Safety Systems

Production and safety systems confront risk of production losses and safety incidents. Both are related as shown in Figure 1.2. Production losses belong to high-frequency and low-consequence issues. On the contrary, safety incidents are considered as low-frequency and high-consequence problems.

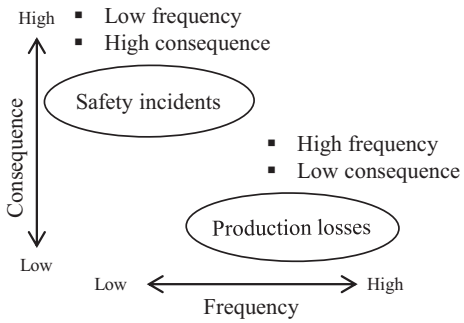


Figure 1.2: Production losses and safety incidents.

Sound performance of production and safety systems is crucial for industrial plants. For example, the oil industry involves large capital investment costs and operational expenditures. The profitability of this industry heavily relies on the RAM (Reliability, Availability and Maintainability) of systems [6]. In particular, offshore oil and gas activities are characterized by an incident-prone environment, which includes harsh natural phenomena (e.g. the wind, wave, current, hydrate, high pressure, and high/low temperature) and complex working processes (e.g. complicated systems and procedures). Therefore, offshore activities are accompanied with high risk of production losses and safety incidents.

1.2 Performance Analysis

Designers and operators can benefit from the performance analysis of a system. The performance of production systems is important for *explicit* benefits of industrial plants. Cumulative production loss can bring negative impact on the financial performance [112], while performance of safety systems is significant for *implicit* benefits of industrial plants. Safety systems aim to protect equipments under control from hazardous events. They are regarded as crucial safety barriers to prevent, control, or mitigate undesired events or accidents [115]. Various industrial sectors have paid close attention to safety systems.

We need to select criteria (metrics) to evaluate the system performance. To some extent, performance is treated as the ability to perform. Therefore, we mainly focus on the *availability* throughout the thesis. With regard to production systems, the metric is production availability, which is the ratio of the actual production of a system to its planned production. Regarding safety systems, the metric is mainly the average unavailability of the system, which is also called "average Probability of Failure on Demand" (PFDavg) in functional safety terminology.

A wide variety of works related to performance analysis have been carried out. However, few studies have considered this issue from the viewpoint of patterns. Details are provided in Sections 2.2 and 2.3.

1.3 Problem Formulation

To analyze performance of production and safety systems, we first need to model them. However, it may be tedious and error-prone to model complex systems. Thus modeling tasks are required to be simplified. Problems considered are formulated as follows:

1. *How to capitalize modeling knowledge*? Knowledge is the awareness gained by experience of a fact or situation. Likewise, modeling knowledge captures modeling experience and shared behaviors during modeling process. It plays a crucial role for effectively handling and reusing models. Without capitalization of modeling knowledge, the modeling process of complex systems remains tedious because it requires to create a model from scratch.
2. *How to implement modeling knowledge*? To implement is to put a decision/plan into effect. After capitalizing modeling knowledge, we have to implement it concretely with a specific modeling language. A modeling language needs to be selected for stabilizing modeling knowledge. Review of classical and model-based formalisms is required before choosing a modeling language.
3. *How to reuse modeling knowledge*? Reuse of modeling knowledge is a common practice in performance analysis of systems. After capitalizing and implementing modeling knowledge, we need to explicitly figure out a way to reuse it. Reusing capitalized modeling knowledge allows us to model new systems in a more efficient way.

1.4 Modeling Patterns

Modeling complex systems is a complex process in itself. Without a solid methodology, it cannot be made efficient. In particular, two rules need to be obeyed: First, models should be designed at the right level of abstraction. On the one hand, the model has to be detailed enough to encompass relevant information. On the other hand, an overly detailed model may lead to intractable calculations of performance indicators. Second, modeling experience is expected to be capitalized. Otherwise, modeling activity is unlikely to be profitable. There should be as much *reuse* as possible.

The whole point of a pattern is to reuse, rather than to reinvent [34]. A pattern describes a problem occurring frequently and depicts the core of the solution for this problem [2], thus we could use this solution constantly. Patterns were first formally proposed in civil engineering. They have been adopted in software engineering afterwards. We could depict the pattern problems, condition constraints, code templates, final results, and trade-offs when employing patterns. Further details of modeling patterns studies can be found in Section 4.1.

The requirement of reusing modeling experience can be met by developing *modeling patterns*. The advantage of this approach is twofold. First, it enables beginners to avoid “blank page” syndrome. With patterns, analysts have models at hand that are both a source of inspiration and an example of the right level of abstraction. Second, it makes it possible to capitalize modeling experience, by considering components in libraries as prototypical examples that must be tailored for a particular problem and be enriched with new experiences.

1.5 Objectives of the Thesis

This thesis aims at developing a modeling pattern-based approach for performance analysis of production and safety systems, especially in process industry. This methodology relies on:

1. *Capitalization of modeling knowledge as a catalog.* The pattern can be utilized for reusing capitalized knowledge. Inspired by design patterns, we present modeling patterns as a catalog. We describe and store modeling knowledge with structured items. Thus frequently occurring behaviors and their solutions can be recorded.
2. *Implementation of modeling patterns as a library with a formal language.* AltaRica language is devoted to performance and safety analysis. It is introduced in IEC 61508 [50] as a technique for calculating probabilities of hardware failures in safety systems. It is also mentioned in ISO/TR 12489 [60]. AltaRica has become a defacto European industrial standard for model-based safety assessment [15]. Thus the AltaRica modeling language is selected to implement modeling patterns in this thesis.
3. *Reuse of modeling patterns with experimental studies.* To demonstrate capabilities of modeling patterns, we reuse implemented modeling patterns to model and analyze a set of production and safety systems. Obtained experimental results are compared with those from the literature.

1.6 Structure of the Thesis

The rest of the thesis is organized as follows. Performance analysis of production and safety systems, as well as model-based safety assessment are discussed in Chapter 2. Chapter 3 recalls the AltaRica modeling language and its mathematical background (i.e. guarded transition systems). Chapter 4 introduces modeling patterns in the realm of guarded transition systems. The methodology of how to develop modeling patterns is proposed as well. Chapter 5 shows the catalog of proposed modeling patterns. Chapter 6 demonstrates capabilities of modeling patterns with experimental studies. Chapter 7 concludes the thesis with summary and recommendations.

Chapter 2

Performance Analysis of Production and Safety Systems

This chapter introduces performance analysis of production and safety systems. Firstly, related definitions for performance analysis are discussed. Secondly, production-performance analysis are reviewed. Subsequently, reliability analysis of safety systems are examined. Related work about model-based safety assessment is presented. Finally, modeling languages that can be used for performance analysis are introduced.

2.1 Glossary for Performance Analysis

☞ **Reliability**: The ability of an item to perform a required function under given conditions for a given time interval [44].

The reliability $R(t)$ of an item is:

$$R(t) = Pr(T > t), \forall\, t > 0 \tag{2.1}$$

where T is the time to failure.

☞ **Availability**: The ability of an item to be in a state to perform a required function under given conditions at a given instant of time, or in average over a given time interval [58].

The availability $A(t)$ and reliability $R(t)$ of a nonrepairable component are *identical*.

☞ **Average availability**: Average value of the availability over a given interval $[t_1, t_2]$ [60].

The average availability $\bar{A}(t_1, t_2)$ of an item is defined as follows:

$$\bar{A}(t_1, t_2) = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} A(\tau) \mathrm{d}\tau \tag{2.2}$$

☞ **Unavailability**: Probability for an item *not* to be in a state to perform as required at a given instant [60].

Unavailability $U(t) = 1 - A(t)$. It is called Probability of Failure on Demand (PFD) in the functional safety standard terminology.

☞ **Average unavailability**: Average value of the unavailability over a given interval $[t_1, t_2]$ [60].
The average unavailability $\bar{U}(t_1, t_2)$ of an item is defined as follows:

$$\bar{U}(t_1, t_2) = 1 - \bar{A}(t_1, t_2) = 1 - \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} A(\tau) \mathrm{d}\tau \tag{2.3}$$

The average unavailability is labeled as the average Probability of Failure on Demand (PFDavg) in the functional safety terminology.

☞ **Production availability**: The ratio of production to planned production, or any other reference level, over a specified period of time [58].

The output of a production system varies a lot, and the availability is not enough to measure the performance of the production system [105]. The production availability, production assurance, or production regularity have the similar meaning about how a system is capable of meeting demands for deliveries or performance [7]. The production availability P_A is defined as follows:

$$P_A = \frac{V_P}{V_R} \tag{2.4}$$

where V_P is the produced volume and V_R is a reference production volume.

☞ **Failure frequency**: Conditional probability per time unit that the item fails between t and $t + dt$, provided that it was working at time 0 [60].

☞ **Average failure frequency**: Average value of the time-dependent failure frequency over a given time interval $[t_1, t_2]$ [60]. Formally, it is defined as follows:

$$\bar{w}(t_1, t_2) = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} w(t) \mathrm{d}t \tag{2.5}$$

where $w(t)$ is the failure frequency at time t. Average failure frequency is termed as Probability of Failure per Hour (PFH) in functional safety standards.

2.2 Production-performance Analysis

Long outages are particularly important for process plants [83]. Indeed, production availability is a feasible measurement for evaluating changing production yields. It is defined as the ratio of the actual production to the planned production (field capacity), over a specified period of time [4, 91, 58]. Production availability can combine both RAM (Reliability, Availability and Maintainability) indicators and production expectations. Stakeholders are prone to attach great importance to production availability at ordinary times, which has a strong economic effect [11].

The production target of the production system is often presented in form of production availability [113]. When assessing production availability, we need to consider dynamic behaviors, operation procedures and complex maintenance policies. Since a production system holds several production levels, Flow Diagrams (FD) are used instead of RBD [113].

ISO 20815 [58] and NORSOK Z-016 [91] standards provide a general framework to perform production availability studies. Two categories of methods have been used so far: *analytical* methods and *simulation* methods (see [125, 8] for a review). Analytical methods use a set of predefined formula based on a theoretical model for system availability, which include reliability block diagrams, fault tree analysis, Markov modeling, Petri nets, or combinations of them [42, 68, 36]. These methods are interesting but limited in terms of size and complexity of systems. Simulation based studies are performed using discrete events Monte-Carlo simulation, where events such as failures, repairs, and reconfigurations are associated with stochastic delays [16, 12, 129, 130, 127]. Simulation methods are flexible and can provide highly accurate predictions of system performance. However, it is usually time consuming to use simulation methods.

There are three main ways to develop tools for production availability studies on industrial systems [112]: fault injection in existing design packages, specific Monte Carlo software packages, and generic models. Most production availability analysis performed in industry are conducted via Monte Carlo simulation software. Generic models are developed by the reliability engineering community, which include the Markovian approach, Generalized Stochastic Petri nets (GSPN), and the AltaRica language. Among various attempts to develop methods and tools for production availability analysis, Petri nets and the AltaRica language are among the solid solutions [112].

Some scientific and industrial projects have been carried out on production-performance analysis. A multi-state, multi-output offshore installation is proposed as test case within the European Union sponsored Thematic Network SAFERELNET [111, 129]. FAMUS (Flow Assurance by Management of Uncertainties and Simulation) is a Joint Industry Project run by DNV and IFP and sponsored by Total, Statoil, ENI, GDF, and Petrobras. Its goal is to couple RAM modeling with Flow Assurance issues in order to provide realistic decision support [25, 30]. The production system is modeled by hybrid stochastic Petri nets, which can model both discrete and continuous aspects of the system [25].

Several commercial tools are available to conduct production-performance analysis. MAROS (Maintainability, Availability, Reliability, Operability Simulation) program is developed for RAM analysis for upstream oil and gas industry [26]. The production availability in MAROS program is termed as production efficiency. TARO^{TM} program is developed for RAM analysis of downstream oil and gas industry, especially for refining and petrochemical plants [27]. An Event-Driven algorithm is used to create lifecycle scenarios of the system under investigation accounting for its reliability, maintainability and operating policies [27]. GRIF (GRaphical Interface for reliability Forecasting) is a system analysis software platform for determining essential indicators of dependability [122]. It includes the BStok (Stochastic block diagrams or RBD driven Petri Nets) and Petro (Multi-flow Stochastic Block diagram) modules that can be used to assess production availability of a production system. MAROS and GRIF/BStok are compared by researchers from DNVGL [20]. Both tools can generate similar results of production performance and meet industrial needs.

Several studies are conducted on production-performance analysis of offshore systems. Pro-

duction availability of an offshore plant is assessed using Monte Carlo simulation [129]. This offshore installation is part of SAFERELNET project. SPN are also applied to model the production availability of the same system [17]. FPSO (Floating Production Storage and Offloading) systems are employed for processing and provisionally storing the crude oil from offshore platforms or subsea wells. SPN are selected for modeling production availability of a FSPO system [85]. SPN are also used for the production availability analysis of offshore facilities with GRIF [126]. Their work mainly focuses on the subsea part of the FPSO. A recent study investigates the production availability of subsea separators [118].

Modularized models keep a better control in modeling development [112]. Flow diagram-driven Petri nets are proposed to deal with production availability issues for production systems [114].

In this thesis, we propose modeling patterns for production-performance analysis implemented using the AltaRica language [86]. We apply proposed modeling patterns on several production systems, including the SAFERELNET test case.

2.3 Reliability Analysis of Safety Systems

Safety Instrumented Systems (SIS) are crucial safety barriers for preventing hazardous accidents in industrial systems. These systems are composed of sensors (e.g. pressure sensors), logic solvers (e.g. programmable logic controllers), and final elements (e.g. isolation valves). Logic solvers translate signals transmitted from sensors into decisions made on final elements. SIS have received huge attention from various industrial sectors. Associated standards are proposed in specific industries, such as the process industry [56], the nuclear power industry [52], the machinery industry [46, 61], the automotive industry [59], as well as the railway industry [45, 54, 49]. The main standard is IEC 61508 [50]. Sound performance of SIS is crucial for protected systems (i.e. Equipment Under Control: EUC).

Reliability issues of SIS have been studied extensively (see e.g. [60, 104, 105, 29]). Many aspects related to SIS have been investigated, including proof tests (see e.g. [73, 21, 79]), K-out-of-N voting structures (see e.g. [121, 92, 62]), common cause failures (see e.g. [81, 66, 40]), spurious failures (see e.g. [82, 64]), human and organizational factors (see e.g. [109, 100]), uncertainty (see e.g. [57, 65]), and optimization issues (see e.g. [120, 80]). In particular, Markov Chain models are applied to study proof tests considering demand rate and imperfect behaviors in [73]. In [21], FT and PN are used to investigate proof tests. Another way to study SIS is to use (simplified) equations/formulae. In [79], PN and approximation formulae are employed to analyze the safety performance of insert testing (proof test after the repair of a dangerous detected failure by the same maintenance team). In [121], MooN structures (i.e. k-out-of-n: G system with added voters) are analyzed using FT and formulae. In [92], the authors generalize an analytical equations for analyzing any KooN structures. In [62], the authors give a generalized PFD formula for KooN systems used in IEC 61508. In [81], a Common Cause Failure (CCF) defense approach is presented, which comprises checklists and analytical tools, for SIS in oil and gas industry. SIS normally operate in low demand mode, which means that regular testing and inspection are required to reveal SIS failures. In [66], average PFD formulae for KooN systems are proposed. They take into account both CCF and non-periodic partial testing. In [40], generic estimates of beta-factors for CCF are discussed. In [82], generic formulae are established for

spurious trips, and in [64], the authors further develop available analytical formulae for spurious trip rate. In [109], a methodology is proposed to evaluate human and organizational factors in operational phase of SIS. In [100], a framework is given to manage factors influencing beta-factor (for modeling CCF) of SIS in operational phase. Uncertainty of the PFD estimate is classified as completeness uncertainty, model uncertainty, and parameter uncertainty [65]. In order to avoid evaluation uncertainties, influencing factors (e.g. design, environment, and use) in reliability are discussed [18, 19]. A methodology for failure rate evaluation of SIS considering influencing factors is proposed. In [57], an approach combining Monte Carlo and fuzzy set is put forward to handing uncertainties in SIS. In [120], the researchers show multi-objective optimization of proof testing policies using genetic algorithms. In [80], the authors gave a model to optimize operation and testing of SIS, applying modeling by fault trees together with optimization by genetic algorithm.

Few works related to patterns of SIS have been carried out. Related works can be found in [60, 114], where the Reliability Block Diagram driven Petri nets are proposed for reliability analysis of SIS. The readability of PN is improved by means of RBD. FT patterns are proposed to model safety mechanisms of automotive electric and electronic functions [23]. FT patterns include classic second order Safety Mechanisms (SM2) representation, maintenance, periodic tests, and the scenario without SM2. The proposed FT pattern models are tested using XFTA[1], a Fault Tree calculation engine.

SIS have common behaviors such as periodic test policies to discover dangerous undetected failures. In a recent work, we propose a pattern-based methodology for reliability assessment of SIS [87]. Based on a series of SIS provided in ISO/TR 12489, a set of modeling patterns is put forward.

2.4 Model-based Safety Assessment

The classical safety assessment techniques suffer from several intrinsic and incidental limitations. One of main limitations is that FT and ET, the two formalisms that are mainly used to design models, stand at a relative low level. Not only their expressive power is limited, but models are distant from systems under study. As a consequence, models are hard to design and even harder to share amongst stakeholders and to maintain throughout the life-cycle of systems. Hence the interest for higher level modeling formalisms increases steadily.

Major advantages of Model-Based Systems Engineering (MBSE) include enhanced communications between stakeholders and team members. MBSE also allows shared understanding of the domain, improved knowledge capture, design precision and integrity without disconnections among data representations, better information traceability, enhanced reuse of artifacts, and reduced development risk [101]. The model-based approach for safety analysis is gradually winning the trust of safety engineers but is still a wide domain of research [76]. Model-Based Safety Assessment (MBSA) is a reliability engineering branch of MBSE. MBSA techniques have been developed in recent years to address challenges in analyzing and verifying complex safety–critical systems. MBSA focuses on developing effective and robust safety assessment techniques through the automation of the safety analysis process [110].

[1] see e.g. http://www.altarica-association.org

Among the MBSA techniques, the AltaRica language is introduced in IEC 61508 [50] as a technique for calculating probabilities of hardware failures in SIS. It is also mentioned in ISO/TR 12489 as a formal language to model the functioning and dysfunctioning of industrial systems [60]. AltaRica has become a defacto European industrial standard for MBSA [15]. It is currently employed as an internal representation language by several safety analysis workshops: Cecilia OCAS (Dassault Aviation), Simfia (EADS Apsys), Safety Designer (Dassault Systemes), and AltaRica Studio (LaBRI) [76].

The first version of AltaRica has been designed at the end of 1990s [95, 3]. Since then, a significant amount of scientific research has been done and a solid industrial experience has been acquired, including the certification of aircrafts. Commercially distributed environments make it possible to create, to edit, to assess and to simulate models graphically. AltaRica Data-Flow, the 2.0 version of AltaRica, has already been used for evaluating the production availability of an oil production system [11]. Mathematical foundations of AltaRica Data-Flow [11] and AltaRica 3.0 [99] are mode automata and guarded transition systems, respectively. Guarded transition systems make the new version of the language possible to handle systems with instant loops and to define acausal components [108].

Several assessment tools are available for analyzing AltaRica 3.0 models. These tools include Markov chain generator, fault tree compiler, stepwise simulator, and stochastic simulator. The last one is currently the most powerful one, especially when other tools cannot work [76]. Indeed, stochastic simulation is an important tool for safety and reliability analysis of the systems, which could generate reasonable results for safety and reliability indicators [10, 128].

2.5 Modeling Languages

Modeling languages are indispensable for performance analysis. These languages can be classified into two categories: classical and model-based approaches, as shown in Figure 2.1. Classical approaches are those traditionally leveraged for reliability assessment. They are further classified into Boolean and state/transition formalisms.

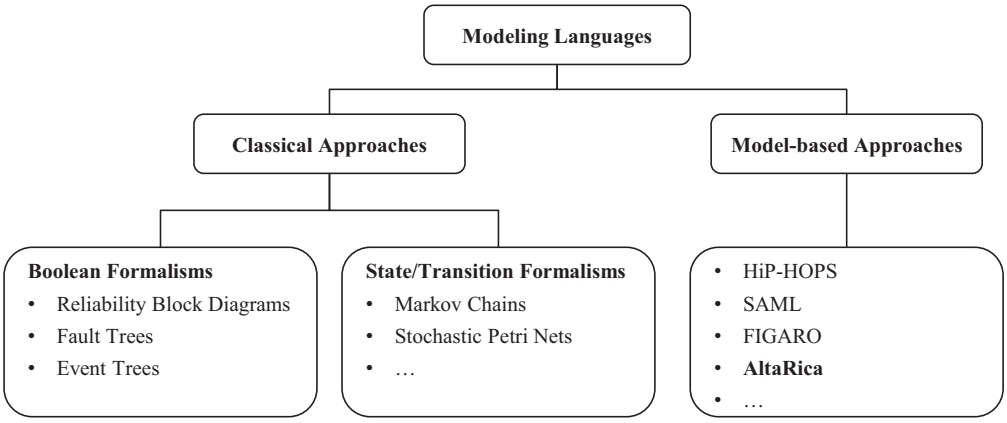


Figure 2.1: Categories of modeling languages.

Boolean formalisms are commonly used in safety and reliability studies of industrial systems. Boolean formalisms can describe *static* (logical) links between elementary failures and

system failure. Reliability block diagrams, fault trees, and event trees belong to Boolean formalisms.

State/transition formalisms can describe how a system behaves (jumps between states) according to arising events (e.g. failures and repairs). Markov Chains (MC) and Stochastic Petri Nets (SPN) are example of such formalisms.

Classical approaches are well established and are used extensively for reliability assessment. Nevertheless, models designed with these formalisms are far from the functional architecture of the system. As a consequence, models are hard to design and to maintain throughout the lifecycle of systems. A small change in specifications may require a complete revisit of safety models, which is both resource consuming and error prone [76]. Therefore model-based approaches are proposed to track this issue. Model-based approaches describe system with high modeling formalisms. Many approaches have been developed, such as Hip-HOPS, SAML, FIGARO, and AltaRica.

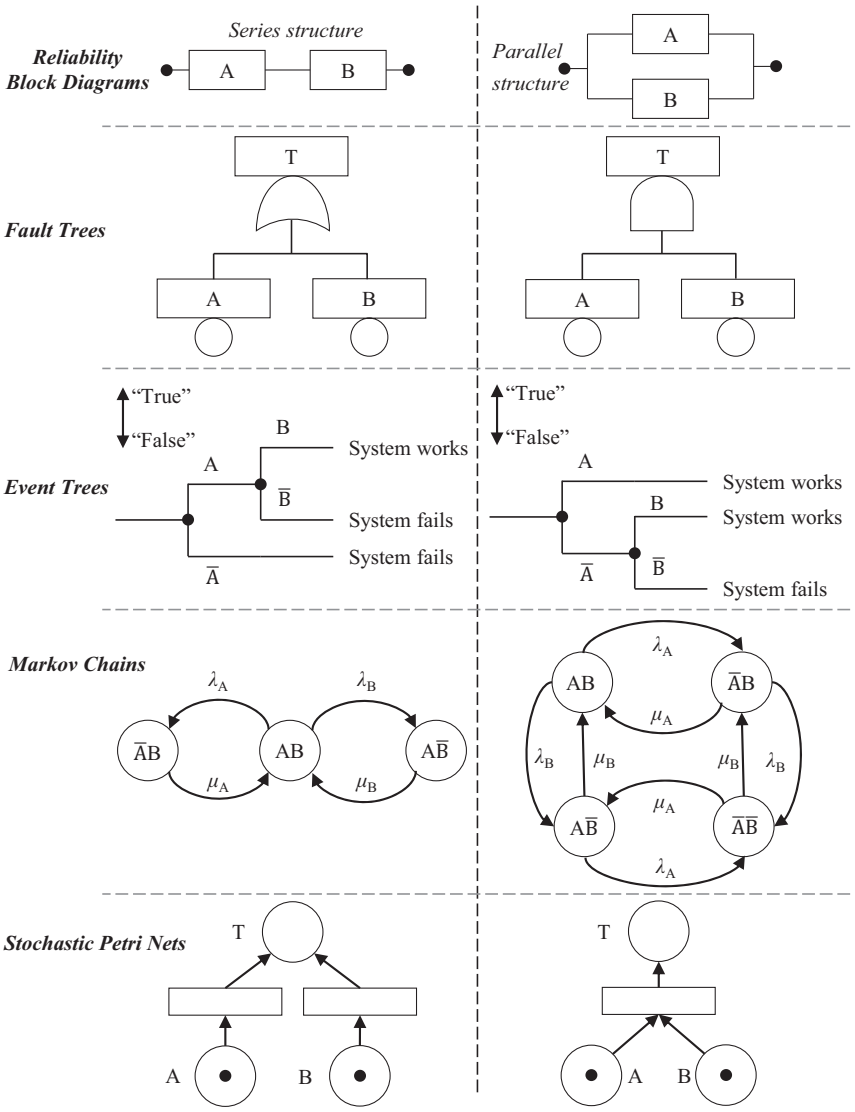


Figure 2.2: Demonstrations of classical approaches.

2.5.1 Classical Approaches

We take simple series and parallel structures to illustrate classical approaches, as shown in Figure 2.2. Classical approaches are graphical and event-based. They can explicitly describe systems of interest. However, they have their own drawbacks. It is not easy to construct complex systems using classical approaches. A Boolean formalism provides *static* combinations of blocks/events. This formalism is not appropriate to model dynamic systems with complex maintenance policies and dynamic behaviors.

In general, state/transition formalisms are more powerful than Boolean ones. They can describe dependencies and dynamic behaviors. However, state/transition formalisms become hard to control when the system is complex. MC provide many metrics to evaluate the system of interest. However, MC are particularly appropriate to model small systems with complex/dynamic behaviors. This is because the number of states increases *exponentially* with the increase of the number of components. SPN are capable of modeling systems with complex behaviors. The size of an SPN model keeps *linear* with the number of components. But SPN become hard to control when modeling large systems. In the following, several classical approaches are briefly presented.

Boolean Formalisms

In a Reliability Block Diagram (RBD), the components of a system are represented using *blocks*. The RBD shows the logical connections between these components. It is thus a success-oriented network for illustrating the system function. An international standard, IEC 61078 [55], is available to describe procedure of using RBD.

A Fault Tree (FT) is a top-down (deductive) logic diagram that displays interrelationships between a potential critical event in a system and causes of this event. A fault tree provides valuable information about possible combinations of basic events that can result in TOP event (i.e. a potential critical event). An international standard, IEC 61025 [47], is issued to depict procedure of applying FT. Details of FT can also be found in [124, 116]. A fault tree can always be translated into a reliability block diagram, and vice versa. In a word, FT and RBD are equivalent to some extent, up to minor modifications.

An Event Tree (ET) pictures the evolution of hazardous events. It is inductive and follows a forward logic. The resulting diagram displays possible accident scenarios, that is, event sequences that may follow a specified hazardous event. Responses of the system/plant to the hazardous event are illustrated in event tree. The international standard IEC 62502 [51] illustrates the procedure of using ET. The combination of ETA (Event Tree Analysis) and FTA (Fault Tree Analysis) is sometimes referred to as Cause-Consequence Analysis (CCA).

State/Transition Formalisms

A Markov chain is an analytical method that is based on a Markov process with discrete states and continuous time [103]. An international standard, IEC 61165 [48], is published to elaborate the procedure of applying this formalism. Markov chains are suitable to analyze small but complex systems. They provide a range of performance measures for the system.

A Petri Net (PN) is a graphical and mathematical tool for modeling and analysis of discrete event systems [103]. It is said to be a stochastic Petri net when random delays are associated

with places or transitions. An international standard, IEC 62551 [53], provides guidance for application of Petri nets in dependability field. The modeling formalism PN is powerful among classical approaches. Petri nets can replace FT, ET, and MC and add several new features to analysis [103]. For example, PN can be tested with stepwise simulation/execution (animators).

2.5.2 Model-based Approaches

```
domain ComponentState {WORKING, FAILED}
class RepairableComponent // Repairable component
    ComponentState varState (init = WORKING);
    Boolean varInFlow, varOutFlow (reset = false);
    parameter Real Lambda = 0.001; // Failure rate
    parameter Real Mu     = 0.1;   // Repair rate
    event failure (delay = exponential (Lambda));
    event repair  (delay = exponential (Mu));
    transition
        failure: varState == WORKING -> varState := FAILED;
        repair:  varState == FAILED  -> varState := WORKING;
    assertion
        varOutFlow := if varState == WORKING then varInFlow else false;
end
// ---------------------------------------------------------------------------
class SeriesStructure
    RepairableComponent A, B; // instantiate RepairableComponent
    parameter Boolean inFlow = true; // system input is true
    Boolean outFlow (reset = false);
    observer Boolean systemState = outFlow; // indicate system state
    assertion // different with parallel structure
        A.varInFlow := inFlow;
        B.varInFlow := A.varOutFlow;
        outFlow     := B.varOutFlow;
end
// ---------------------------------------------------------------------------
class ParallelStructure
    RepairableComponent A, B;
    parameter Boolean inFlow = true;
    Boolean outFlow (reset = false);
    observer Boolean systemState = outFlow;
    assertion //different with series structure
        A.varInFlow := inFlow;
        B.varInFlow := inFlow;
        outFlow     := A.varOutFlow or B.varOutFlow;
end
```

Figure 2.3: Demonstration of the AltaRica 3.0 language.

Model-based approaches try to address difficulties encountered by classical ones. These approaches are attracting increasing attention. In this realm, systems are usually modeled with a high-level formalism. They aim at the automation of safety and reliability studies. In this way, it becomes easier to design, share, and maintain system models. Each approach has its own advantages and drawbacks. For instance, it is hard for model-based approaches to capture

intrinsically continuous phenomena. It is also difficult for them to deal with processes which are dynamically created and destroyed in a mission. In the following, several model-based approaches are briefly recalled.

Hip-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies) is an automated FT, FMEA (Failure Mode and Effects Analysis) and optimisation tool. It is developed by the Dependable Systems Research Group at University of Hull in the United Kingdom. There are three main phases in HiP-HOPS: model annotation, fault tree synthesis, as well as FT and FMEA analysis phase [93, 94, 110].

SAML (Safety Analysis and Modeling Language) was designed as a tool independent formal system specification and modeling language developed in Germany [37, 76]. A SAML model is expressed in terms of finite stochastic state automata. Besides technical systems with deterministic behavior, SAML may also denote failure models with stochastic behaviors and system environment. Due to the combination of stochastic and non-deterministic specification, the semantics of a SAML model is defined as Markov decision process.

FIGARO is a modeling language for reliability assessment. It is developed by EDF R&D (Research and Development Division, Electricity of France) [14, 13]. FIGARO is applied to describe knowledge bases of KB3. KB3 workbench allows automatic reliability analysis on the basis of system layout, system mission, as well as a generic Knowledge Base (KB).

AltaRica is a language for event driven modeling of complex systems [95, 3, 11]. It is especially suitable for performance and safety analysis. The current version of the language is AltaRica 3.0 [99, 98]. Models in AltaRica 3.0 are mathematically described by guarded transition systems [108]. Several assessment tools are available, such as stepwise simulators, compilers to fault trees, generators of critical sequences, compilers to Markov chains, and stochastic simulators [99, 98]. The corresponding AltaRica 3.0 model of series and parallel structures in Figure 2.2 is illustrated in Figure 2.3. Components in the structure is regarded as repairable components. A class named RepairableComponent is predefined. It is instantiated by two components afterwards. Differences between series and parallel structures are in their assertions.

2.6 Summary

In this chapter, performance analysis of production and safety systems are introduced. The glossary and modeling languages for performance analysis are reviewed. Topics of production-performance analysis, reliability analysis of safety systems, as well as model-based safety assessment are discussed. Since we use the AltaRica 3.0 language throughout this thesis, the following chapter is dedicated to introduce it.

Chapter 3

AltaRica Modeling Language

In this chapter, we present the AltaRica 3.0 modeling language and its mathematical background, that is, the Guarded Transition Systems (GTS). They have been used to implement proposed modeling patterns. Further details of GTS and AltaRica 3.0 can be found in [108, 98].

We illustrate GTS and AltaRica 3.0 with a simplified example, as it is depicted in Figure 3.1. Components A, B1, and B2 make up a series-parallel structure (main structure), and they are repairable. Component C acts as a cold standby component. The failure of the main structure activates component C.

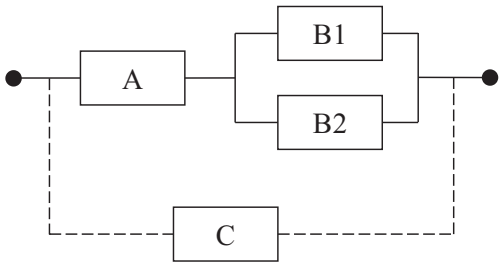


Figure 3.1: Block diagram of a simplified system.

Functional breakdown of this system is shown in Figure 3.2. Each leaf stands for a basic component. Each intermediate node represents a subsystem.

3.1 Formal Definition of Guarded Transition Systems

A GTS is a quintuple $\langle V, E, T, A, \iota \rangle$, where V is a finite set of variables, E is a finite set of events, T is a finite set of transitions, A is an assertion, and ι is the initial assignment of variables.

(i) V is the disjoint union of the set S of state variables and the set F of flow variables: $V = S \uplus F$. Each variable $v \in V$ takes its value from a domain denoted by $domain(v)$. Variables can be Boolean, Integers, Floating point numbers, members of finite sets of symbolic constants or anything convenient for the modeling purpose.

A variable assignment is a function from V to $\prod_{v \in V} domain(v)$. A variable update is a function from $\prod_{v \in V} domain(v)$ into itself. It is a function that transforms a variable assignment into another one.

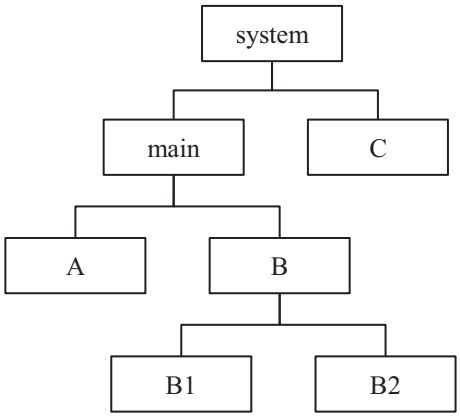


Figure 3.2: Functional breakdown of the simplified system.

(ii) Each event $e \in E$ is associated with:

– A monotonically increasing and invertible function $delay_e$ from $[0,1]$ into $\Re^+$, the set of positive real numbers.

– A weight (a real number) $weight_e$ (by default, $weight_e = 1.0$).

(iii) Each transition $t \in T$ is a triple $\langle e, g, a \rangle$, denoted by $g \xrightarrow{e} a$, where e is an event in E, g is a Boolean condition (guard) over the variables in V and a is an instruction over the variables of V, that is a variable update. a is called the action of the transition.

(iv) The assertion A is an instruction over the variables of V.

Let σ be a variable assignment and $t : g \xrightarrow{e} a$ be a transition which is potentially fireable in σ, such that $\sigma(g) = true$. Firing t updates σ into the assignment $\rho = A(a(\sigma))$, which means applying on σ the update of a first, then the update A (the global assertion).

We say that a variable $v \in V$ is impacted by the update of σ into ρ if $\rho(v) \neq \sigma(v)$. By extension, we say that the transition $g' \xrightarrow{e'} a'$ is affected by this variable update if at least one of the variables occurring in g' is impacted by the update.

Let $t : g \xrightarrow{e} a$ be a transition in T. By extension, we define $weight_t$ as $weight_e$. If the two transitions can be fired at the same time, then the weight is used to choose randomly among them.

3.2 States and Transitions

The AltaRica 3.0 modeling language is an event-driven language. State variables are updated with transitions.

Transitions in GTS are categorized into immediate and timed transitions, as shown in Figure 3.3. Immediate transitions are fired with zero delays. Timed transitions are fireable after non-zero delays. An immediate transition has a positive priority, while a timed transition has a priority equals to zero.

Immediate transitions can be classified into plain immediate transitions and conditional (on-demand) immediate transitions. Plain immediate transitions obey the *diamond property*. According to this property, if we have two fireable immediate transitions t_a and t_b, t_b remains

fireable after the firing of t_a and vice versa. For conditional immediate transitions such as $turnOn$ and $failureOnDemand$ for a cold standby component, a weight is assigned to each transition.

Timed transitions can be classified into Dirac and regular timed transitions. Dirac transitions are fired after deterministic (> 0) delays. Delays associated with regular timed transitions obey certain probability distributions, such as exponential or Weibull ones.

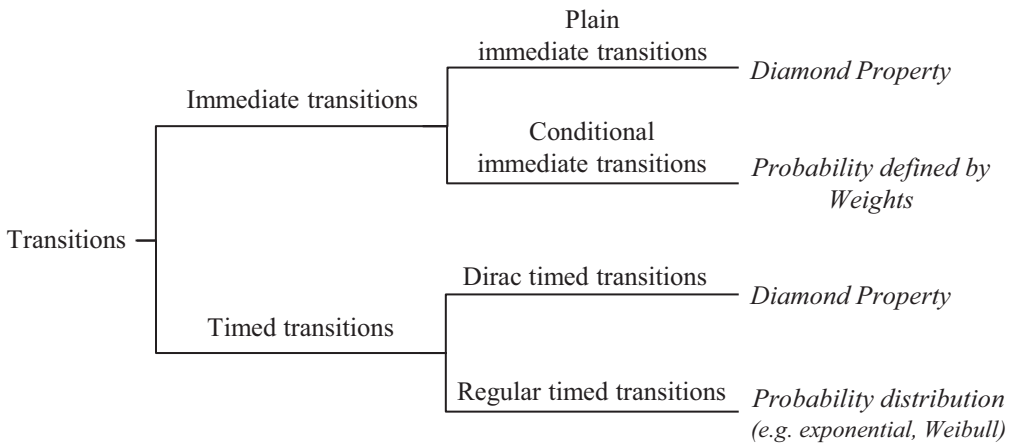


Figure 3.3: Transitions in guarded transition systems.

Note that several conditional immediate transitions can be fired simultaneously.

Let Γ be a run. There may be several transitions $t_1,\ldots,t_k$ actually fireable in Γ. During a stochastic simulation, we have to choose which of the t_i's to fire. This choice is done via weights. Namely, the probability $p(t_i)$ to draw the transition t_i is defined as follows:

$$p(t_i) \stackrel{def}{=} \frac{weight_{t_i}}{\sum_{j=1}^{k} weight_{t_j}}$$

Consider the cold standby component C in Figure 3.5 as an example, $failureOnDemand$ and $turnOn$ are conditional immediate transitions. Both are firable when the component state is `STANDBY`. They are associated with respective weights γ and 1-γ. This probability is defined via attribute `expectation` in AltaRica model.

The system in Figure 3.1 is composed of two types of components: repairable component and cold standby component. The GTS representing a repairable component is shown in Figure 3.4. It is composed of following elements:

- A state variable $state$, which takes its value from the enumeration {WORKING, FAILED}. The initial value of $state$ is WORKING.
- Two Boolean flow variables: $inFlow$ and $outFlow$.
- Two events: $failure$ and $repair$. Both are stochastic events with exponentially distributed delays.
- Two transitions:
  - failure: $state$ = WORKING $\longrightarrow$ $state$:= FAILED

  - repair: $state$ = FAILED $\longrightarrow$ $state$:= WORKING
- An assertion made of a unique assignment: outFlow := if $state$ == WORKING then inFlow else false

```
domain RepairableComponentState {WORKING, FAILED}
class RepairableComponent
    RepairableComponentState state (init = WORKING);
    Boolean inFlow, outFlow (reset = false);
    parameter Real Lambda = 1.0e-5;
    parameter Real Mu = 1.0e-1;
    event failure (delay = exponential(Lambda));
    event repair  (delay = exponential(Mu));
    transition
        failure: state == WORKING -> state := FAILED;
        repair:  state == FAILED  -> state := WORKING;
    assertion
        outFlow := if state == WORKING then inFlow else false;
end
```

Figure 3.4: GTS representing a repairable component.

The GTS representing a cold standby component is given in Figure 3.5. It is composed of following elements:

- A state variable $state$, which takes its value from the enumeration {STANDBY, WORKING, FAILED}. The initial value of $state$ is STANDBY.
- Three Boolean flow variables: $demand$, $inFlow$, and $outFlow$. $demand$ determines when to turn on or turn off the component.
- Five events: $turnOn$, $failureOnDemand$, $turnOff$, $failure$, and $repair$. $turnOn$, $failureOnDemand$, and $turnOff$ are immediate transitions (i.e. delay = 0), while $failure$ and $repair$ are stochastic transitions with exponentially distributed delays. $turnOn$ and $failureOnDemand$ are associated with weights $1-\gamma$ and γ, respectively.
- Five transitions:
  - turnOn: $state$ = STANDBY and demand $\longrightarrow$ $state$:= WORKING
  - failureOnDemand: $state$ = STANDBY and demand $\longrightarrow$ $state$:= FAILED
  - turnOff: $state$ = WORKING and not demand $\longrightarrow$ $state$:= STANDBY
  - failure: $state$ = WORKING $\longrightarrow$ $state$:= FAILED
  - repair: $state$ = FAILED $\longrightarrow$ $state$:= STANDBY
- An assertion made of a unique assignment: outFlow := if $state$ == WORKING then inFlow else false

Relationships between components can be described through flow propagations and synchronization mechanisms. In $GTS = \langle S \uplus F, E, T, A, \iota \rangle$, flow propagations refer to connections of flows via assertions (A). Synchronization mechanisms refer to simultaneously occurrence of several events (E). We describe flow propagations and synchronization mechanisms one by one in the following two sections .

```
domain ColdStandbyComponentState {WORKING, FAILED, STANDBY}
class ColdStandbyComponent
    ComponentState state (init = STANDBY);
    parameter Real Lambda = 1.0e-5;
    parameter Real Mu = 1.0e-1;
    parameter Real Gamma = 1.0e-2;
    Boolean demand (reset = false);
    Boolean inFlow, outFlow (reset = false);
    event turnOn              (delay = 0, expectation = 1-Gamma);
    event failureOnDemand (delay = 0, expectation = Gamma);
    event turnOff             (delay = 0);
    event failure (delay = exponential(Lambda));
    event repair  (delay = exponential(Mu));
    transition
        turnOn:          state == STANDBY and demand      -> state := WORKING;
        failureOnDemand: state == STANDBY and demand      -> state := FAILED;
        turnOff:         state == WORKING and not demand -> state := STANDBY;
        failure:         state == WORKING                 -> state := FAILED;
        repair:          state == FAILED                  -> state := STANDBY;
    assertion
        outFlow := if state == WORKING then inFlow else false;
end
```

Figure 3.5: GTS representing a cold standby component.

3.3 Flow Propagation

The flow propagation is indispensable for a modeling formalism to depict flows circulating among components and subsystems. In GTS, these flows are represented via flow variables in F and assertions in A. Flow variables are recalculated through assertions after firing transitions (i.e. state variables in S are updated accordingly). Assertions in A are capable of constructing complex hierarchical structures by connecting flow variables of separate components. Therefore an assertion in A is appropriate to express remote interaction between components.

Two categories of assertions can be found in systems, that is, *looped assertions* and *data-flow (non-looped) assertions*. In GTS, a fixpoint mechanism is applied to deal with looped models. Flow variables in these models depend instantaneously on each other.

Let v, u, and w be three variables from V. Let I be an instruction built over variables from V. If v depends on w (i.e. $v \rightarrow w$) and u depends on v (i.e. $u \rightarrow v$), we say that u depends on w (i.e. $u \rightarrow w$). The dependency relationship between variables can be depicted in a *dependency graph*. A dependency graph is an oriented graph $G_D = (V_D, E_D)$, where V_D is a set of vertices (variables in GTS), and E_D is a set of dependency edges. If there are looped structures in a system, then the corresponding dependency graph of this system contains cycles. Note that if variables are state variables, the corresponding vertices have no out-going edges.

3.3.1 Looped Assertions

Here we propose a gas system to illustrate looped assertions, as shown in Figure 3.6. In this systems, components D, E, and F are repairable components. The turbo-compressor D is fueled by the gas (`D.fuelGas`) from downstream. Note that `D.fuelGas` is treated as a condition, not as an inflow because `D.fuelGas` is consumed to support running of D. Components E and F are dehydration units, which are employed to dehydrate the gas. The inflow of the system (`inFlow`) is a constant. The `outFlow` of the system equals to `F.outFlow` − `D.fuelGas`.

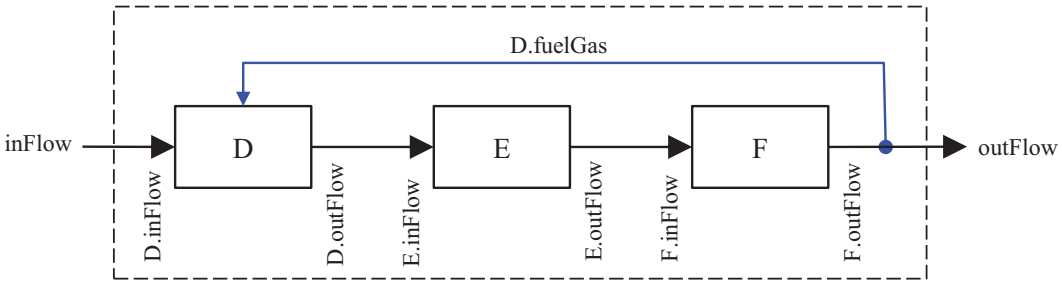


Figure 3.6: A gas system.

Dependency graph of looped instructions is a *Directed Cyclic Graph* (DCG). Figure 3.7 shows the dependency graph of the assertions of the gas system in Figure 3.6. There is one loop inside, that is, "`D.outFlow` → `D.fuelGas` → `F.outFlow` → `F.inFlow` → `E.outFlow` → `E.inFlow` → `D.outFlow`". System state (`systemState`) is determined by states of components *D*, *E*, and *F*. `outFlow` of each component depends on its `inFlow`, `state`, and `systemState`. At the initial stage, if the components are working (`state` = `WORKING`), `systemState` = `TRUE`. Otherwise, `systemState` = `FALSE`. To increase readability of the dependency graph, we give background frameworks for different components.

3.3.2 Data-flow Assertions

Unlike looped assertions, non-looped assertions are called *data-flow assertions* as no variable depends on itself in an instruction. Dependency graph of data-flow instructions is a *Directed Acyclic Graph* (DAG). There is no circle in such a dependency graph. The dependency graph of assertions of the system in Figure 3.1 is shown in Figure 3.8. Note that the demand of *C* (`C.demand`) depends on the status of *A* and *B*.

3.4 Synchronization Mechanisms

The synchronization of events can also connect components in a system. For example, behaviors of Common Cause Failures (CCF) and shared repair crews can be captured via synchronization mechanisms. In the realm of GTS = $\langle V, E, T, A, \iota \rangle$, synchronization can be represented by a transition:

$$e: \ \{!a_1 \& \cdots \& !a_m\} \ \& \ \{?b_1 \& \cdots \& ?b_n\} \ \& \ \{(L_1 \rightarrow R_1) \& \cdots \& (L_r \rightarrow R_r)\}, m \geq 0, n \geq 0, r \geq 0$$

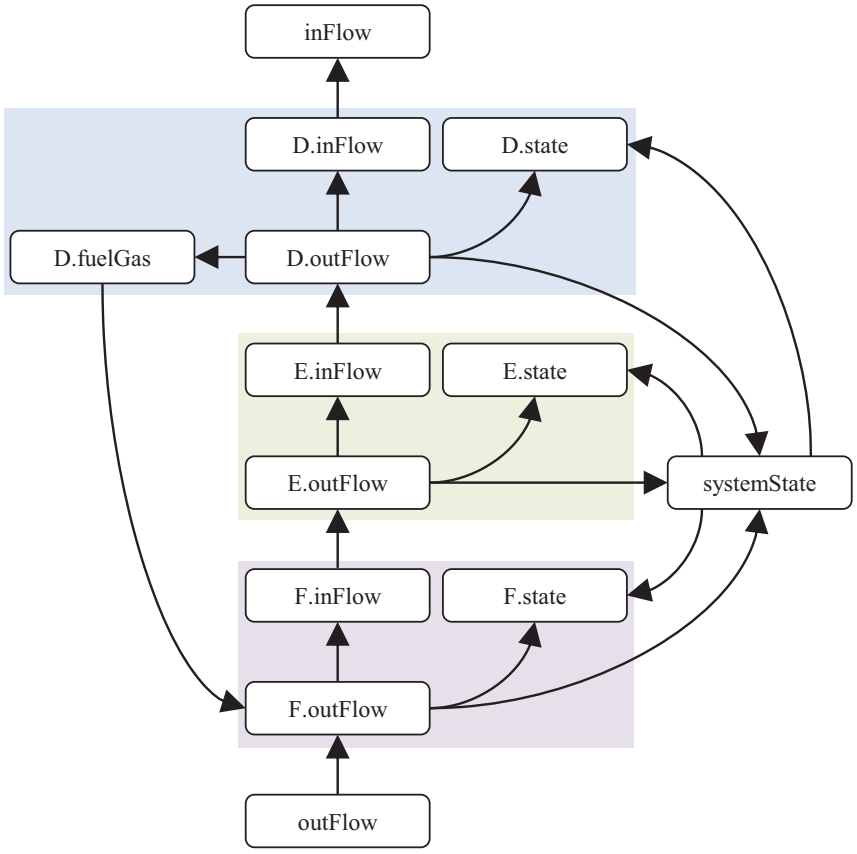


Figure 3.7: Dependency graph of assertions of the gas system in Figure 3.6.

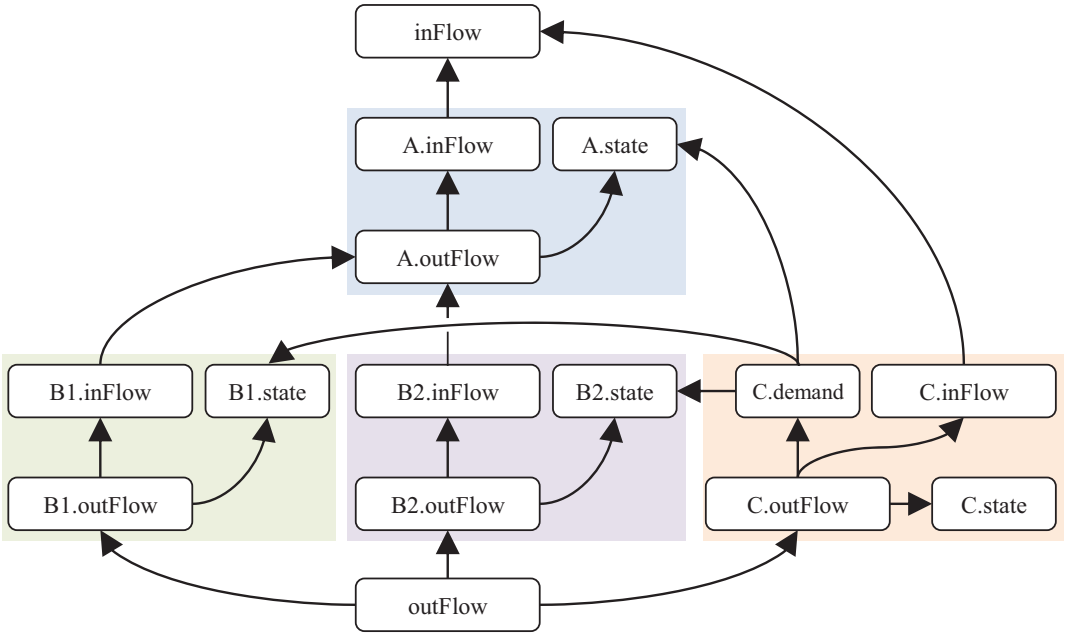


Figure 3.8: Dependency graph of assertions of the simplified system in Figure 3.1.

where e, a_i $(i = 0, \cdots, m)$, b_j $(j = 0, \cdots, n)$ are events from E. Operators $''!''$ and $''?''$ indicate that the associate events are mandatory or optional, respectively. L_k and R_k $(k = 0, \cdots, r)$ are guards and actions, respectively. $L_k \rightarrow R_k$ are unnamed transitions.

We assume that there is a CCF event in the subsystem composed of components B1 and B2 in Figure 3.1. That is, a failure in B1 or B2 can lead to failures of both B1 and B2. CCF event can reduce the benefit of this parallel redundancy.

```
block B
    …
    event CCFB (delay = 0); // Common cause failure
    …
    transition
        CCFB: ?B1.failure & ?B2.failure;
        …
end
```

Figure 3.9: GTS representing a common cause failure.

In AltaRica, we add a new event $CCFB$ and relevant transition to represent this CCF behavior between B1 and B2, as shown in Figure 3.9. Operator $''\&''$ synchronizes B1.failure and B2.failure. In transition $CCFB$, B1.failure and B2.failure are prefixed by $''?''$, which means that both events are optional. To trigger $CCFB$, at least one of the synchronized transitions (B1.failure or B2.failure) need to be fireable. All possible synchronized transitions are fired simultaneously once the synchronizing transition is fired. $CCFB$ is equivalent to the following flattened transition:

```
CCFB: B1.state == WORKING or B2.state == WORKING ->
    {if B1.state == WORKING then B1.state := FAILED;
     if B2.state == WORKING then B2.state := FAILED;}
```

3.5 Prototypes and Classes

Prototypes and classes are used in the AltaRica 3.0 language. A prototype is represented by a structural block. In a block, a component can occur only once. Classes are always employed in blocks. The entire system is usually described with a block.

The structural block modeling the system in Figure 3.1 is shown in Figure 3.10, in accordance with the functional breakdown of this system described in Figure 3.2. This partial AltaRica model represents the hierarchical structure of the system. In this model, each block appears only once.

In the system of Figure 3.1, components in main structure have shared behaviors. In other words, they can fail and be repaired afterwards. To model these similar components efficiently, we prefer to define a generic $class$ rather than duplicate them repeatedly. With a $class$, the component behaviors (i.e. state/flow variables, events, transitions, assertions, and initial assignments) are encapsulated. A $class$ is defined once and instantiated arbitrary times whenever

```
block System
    block main // Main structure of system (A, B1, and B2)
        block A … end
        block B
            block B1 … end
            block B2 … end
        end
    end
    block C … end // Cold standby
end
```

Figure 3.10: Prototype (Block) usage in the simplified system.

needed in the modeling process. Classes of repairable and standby components can be found in Figure 3.4 and Figure 3.5, respectively. Their corresponding graphical representations are shown in Figure 3.11 and Figure 3.12.

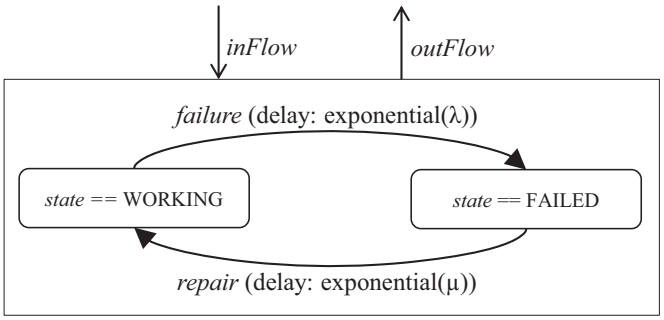


Figure 3.11: Graphical representation of the guarded transition system in Figure 3.4.

Components in this simplified system are modeled via instantiating classes. Subsystems main and B are modeled using blocks. Class usage in the system is shown in Figure 3.13.

Based on proposed classes for repairable components and cold standby components, we can model the system. Three repairable components and one cold standby component work jointly to represent the whole system. Basic components are assembled to model the system, as shown in Figure 3.14. The *demand* of component C is true when the main series-parallel structure fails. This hierarchical model can be flattened into one GTS by the AltaRica complier.

3.6 Semantics

The syntax of the AltaRica 3.0 language is illustrated in a Backus–Naur form (BNF) [9]. The semantics of the AltaRica 3.0 language is a Kripke structure (reachability graph). The structure is composed of *nodes* defined by variable assignments and *edges* defined by transitions (labeled by events). If the delays associated with events are exponentially distributed, the reachability graph can be regarded as a Continuous Time Markov Chain (CTMC).

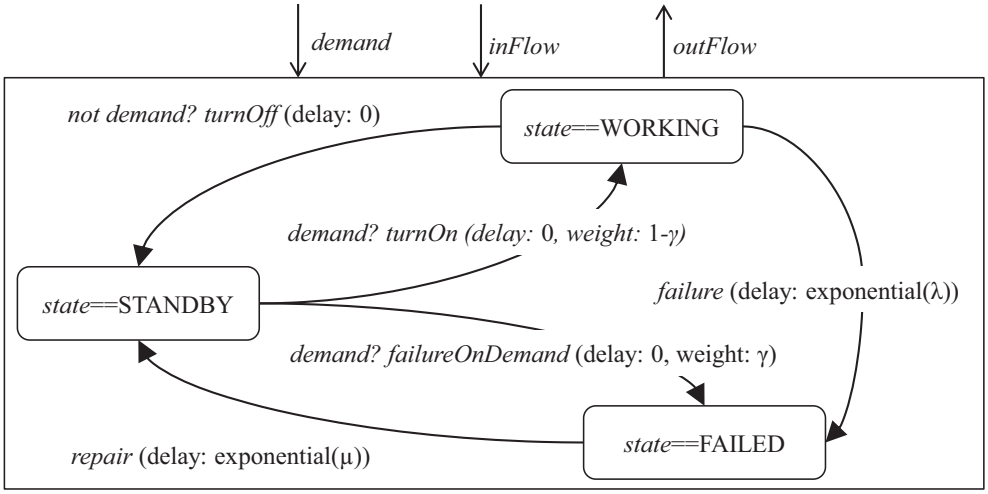


Figure 3.12: Graphical representation of the guarded transition system in Figure 3.5.

```
block System
    block main // Main structure of system (A, B1, and B2)
        RepairableComponent A (Lambda = 3.0e-6, Mu = 2.0e-3);
        block B
            RepairableComponent B1 (Lambda = 2.0e-5, Mu = 1.0e-3);
            RepairableComponent B2 (Lambda = 2.0e-5, Mu = 1.0e-3);
        end
    end
    ColdStandbyComponent C (Lambda = 1.0e-6, Mu = 1.0e-3, Gamma = 0.01);
end
```

Figure 3.13: Class usage in the simplified system.

In particular, semantics of a GTS is defined as the set of its possible runs. A schedule is a function from T to $\Re^+ \cup \{+\infty\}$. It is a function which associates a (possibly infinite) date with each transition. A run of a GTS is a finite sequence of triples (d_0,σ_0,δ_0), $(d_1,\sigma_1,\delta_1),\ldots,(d_n,\sigma_n,\delta_n)$ where the d_i's are dates, the σ_i's are variable assignments and the δ_i's are schedules.

The set of runs of the GTS is defined inductively, starting from date 0, as the smallest set such that:

Initialization: The sequence made of only one triple $(0,\iota,\delta)$ is a run, if for each transition $t : g \xrightarrow{e} a$ in T the following conditions hold:

(i) $\delta(t) < +\infty$ if and only if t is potentially fireable in ι.
(ii) If $\delta(t) < +\infty$, there exists $z \in [0,1]$ such that $\delta(t) = delay_e(z)$.

Run extension: if $\Gamma = (d_0,\sigma_0,\delta_0),\ldots,(d_n,\sigma_n,\delta_n)$ is a run $(n \geq 0)$, then $\Gamma' = \Gamma,(d_{n+1},\sigma_{n+1},\delta_{n+1})$, if there is a transition $t : g \xrightarrow{e} a$ scheduled at date $\delta_n(d) < +\infty$ such that:

(iii) t is potentially fireable in σ_n;

```
block System
    block main // Main structure of system (A, B1, B2)
        RepairableComponent A (Lambda = 3.0e-6, Mu = 2.0e-3);
        Boolean outFlow (reset = false); // Outflow of main structure
        block B
            RepairableComponent B1 (Lambda = 2.0e-5, Mu = 1.0e-3);
            RepairableComponent B2 (Lambda = 2.0e-5, Mu = 1.0e-3);
            Boolean outFlow (reset = false); // Outflow of parallel structure
            assertion
                B1.inFlow  := A.outFlow;
                B2.inFlow  := A.outFlow;;
                outFlow := B1.outFlow or B2.outFlow;
        end
        assertion
            outFlow := B.outFlow;
    end
    parameter Boolean inFlow = true;
    Boolean outFlow (reset = false); // System outflow
    ColdStandbyComponent C (Lambda = 1.0e-6, Mu = 1.0e-3, Gamma = 1.0e-2);
    assertion
        A.inFlow := inFlow;
        C.inFlow := inFlow;
        C.demand := A.state != WORKING or (B1.state != WORKING and B2.state != WORKING);
        outFlow := C.outFlow or main.outFlow;
end
```

Figure 3.14: AltaRica model for the simplified system in Figure 3.1.

(iv) There is no transition t' such that $\delta_n(t') < d$, or $\delta_n(t') = d$, and $priority_{t'} < priority_t$;

We say that such a transition t is actually fireable in the run Γ. σ_{n+1} and δ_{n+1} are related to σ_n and δ_n as follows.

(v) $\sigma_{n+1} = A(a(\sigma_n))$, i.e. σ_{n+1} is obtained from σ_n by firing transition t.

(vi) δ_{n+1} is obtained from δ_n by reexamining t and all transitions impacted by the update from σ_n to σ_{n+1}, that is, for all such transitions $t' : g' \xrightarrow{e'} a'$:

- $\delta_{n+1}(t') < +\infty$ if and only if t' is potentially fireable in σ_{n+1}.
- If $\delta_{n+1}(t') < +\infty$, then there exists $z \in [0,1]$ such that $\delta_{n+1}(t') = d + delay_{e'}(z)$.

Take the main structure of the system in Figure 3.1 as an example, its reachability graph is shown in Figure 3.15. The states of the main structure are expressed as nodes (rounded rectangles) in the reachability graph. The node with bold edges is the initial state of the main structure. The state of main structure is indicated as a Boolean variable `main.outFlow`. The other states of the graph states can be reached via enabled transitions. There are cycles in this reachability

graph. Since all components are repairable, from each node there are transitions leading to the initial node.

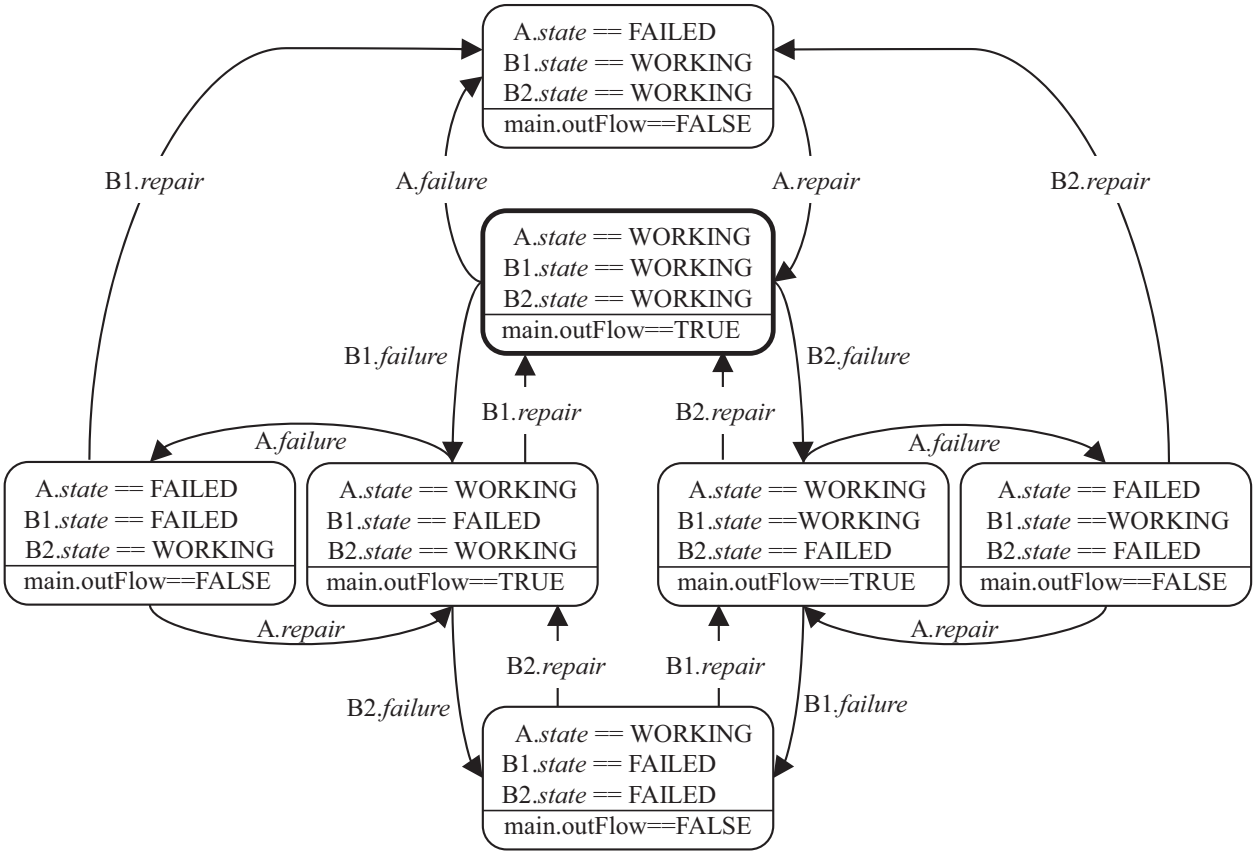


Figure 3.15: Reachability graph of the main structure in Figure 3.1.

3.7 Comparison with other State/transition Modeling Languages

There are differences and similarities between GTS and other state/transition modeling languages such as Markov chains (here we mainly consider CTMC) and stochastic Petri nets.

3.7.1 Differences

Systems are composed of components and subsystems according to the *hierarchical composition* scheme. With MC and SPN, it is difficult to model a system in a hierarchical way. But the integration with other methods may acquire such capability, for example, the reliability block diagrams driven Petri nets (see e.g. [114]). GTS are capable of modeling a system with hierarchies of reusable components.

A modeling formalism is expected to describe *remote interactions* between components. MC and SPN are difficult to model remote interactions in a system. With regard to GTS, it provides two ways to describe remote interactions. On the one hand, flow propagations are modeled

through assertions. One the other hand, synchronization of events can be used to represent interactions between components.

State space is a set of reachable states. The way of representing a state space can be either explicit or implicit. The state space of Markov Chains (MC) is given *explicitly*. However, the state spaces of Stochastic Petri Nets (SPN) and GTS are represented *implicitly*. In order to avoid the state explosion problem, the state space is preferred to be given in an implicit way together with approximations.

3.7.2 Similarities

In reliability studies, it is crucial to be able to describe various events such as failures, repairs, and tests. Therefore it is necessary for a formalism to model a system using *states* and *events*. MC, SPN, and GTS can model a system with states and transitions. MC can deal with exponentially distributed events. GTS and SPN allow arbitrary probability distributions for time delays.

Efficient assessment tools are available for MC, SPN, and GTS. For example, GRIF, a system analysis software platform for determining essential indicators of dependability, can be used to analyze MC and SPN. Fault tree compiler, Markov chain generator, stepwise simulator, stochastic simulator, as well as generator of critical sequence of events are available for assessing GTS.

3.8 Summary

In this chapter, we recall basics of the Guarded Transition Systems (GTS) and AltaRica 3.0 language. They are capable of describing common cause failures, shared resources, acausal components, as well as looped systems. The following chapter is dedicated to introduce modeling patterns.

Chapter 4

Modeling Patterns

Production and safety systems have similarities and differences. On the one hand, both systems can be repaired, series-connected, parallel-connected, and k-out-of-n composed. On the other hand, production systems are usually connected with multi-state components. Safety systems are commonly composed of binary components.

Capitalized modeling knowledge can make modeling activity profitable. The more reuse of such knowledge, the more benefit can be attained. A pattern describes a problem occurring frequently and depicts the core of the solution for this problem [2]. Thus we could use this solution constantly. Moreover, modeling patterns can make modeling process a modular approach.

In the following, we discuss the notion of modeling patterns, then we categorize them according to their purpose. Finally, a methodology to develop modeling patterns is proposed.

4.1 Notion of Modeling Patterns

In some modeling languages, for instance Modelica [33], modeling experience is capitalized by *designing libraries of reusable components*. The experience with AltaRica shows that when designing models, same modeling patterns occur systematically. For instance, modeling two components in cold redundancy involves basically the same AltaRica mechanisms, regardless of these components are pumps, valves, or repair crews. *Designing libraries of modeling patterns* is applying model engineering principles and techniques that have been proved to be very efficient in software engineering [35, 119].

The pattern can be utilized for reusing capitalized knowledge, which was initially proposed in civil engineering [2]. The concept was adopted in software engineering subsequently as *design patterns* [35]. These patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context [35]. A design pattern promotes design reuse, conforms to a literary style, and defines a vocabulary for discussing design [34]. Some researchers tried to provide a general framework of reusing patterns. Pattern based system engineering was proposed [24], whose procedure includes pattern definitions and system development with patterns [39]. The basic idea of pattern-related studies is that the design should be specific to the present problem but also general enough to solve future problems and to meet requirements [35].

Reuse of components and subsystems is a usual practice in modeling safety-critical sys-

tems. To reuse system behaviors, we need to standardize the representation of reusable components and figure out the way they exchange information [67]. A library of reusable argumentation patterns is put forward to capture known solution algorithms and architectural measures/constraints in [70]. This library focuses on safety mechanisms in automotive domain.

In RAMS (Reliability, Availability, Maintainability, and Safety) community, patterns have been discussed in [96]. Patterns involved in accident analysis are discussed in traffic domain [38] and industrial plants [123], albeit these studies mainly employ statistical methods to discover patterns of accident causes. Dependability pattern is the description of a particular recurring dependability problem that arises in specific contexts and presents a well-proven generic scheme for its solution [39]. Resilience design patterns are raised to meet demand of extreme-scale high-performance computing systems [43]. In order to conduct safety analysis efficiently and avoid redesign, the researchers proposed a framework termed SafeSysE which merges safety assessment and systems engineering [88]. FMEA and FTA are automatically generated. Block design patterns are proposed to automatically generate fault trees. Each pattern leads to a sub-fault tree.

An advantage of high level modeling languages (like AltaRica) is to reuse models of components or even subsystems. There are two ways for attaining such an objective [98]: *reuse of components* (objective-oriented), and *reuse of modeling patterns* (prototype-oriented). The reuse of components comes directly from programming languages (like C++ [117]) or modeling languages (like Matlab/Simulink [84] and Modelica). The reuse of modeling patterns starts from an existing code and adapts it to specific requirements [98].

From modeling experience of several aircraft systems using the AltaRica Data-Flow language, Safety Architecture Patterns (SAP) are proposed to simplify modeling missions [69]. SAP are component assemblies used to ensure the safety of architectures [69]. The application of SAP can be found in the avionics domain [69, 89]. Unlike their work, first, we use the AltaRica 3.0 language rather than AltaRica Data-Flow language. Mathematical backgrounds of the AltaRica Data-Flow language and the AltaRica 3.0 language are mode automata [106] and Guarded Transition Systems (GTS) [108], respectively. GTS extends mode automata with the capabilities of modeling instant loops and acausal components (i.e. inflows and outflows are decided at run time). Second, we propose patterns for modeling production and safety systems mainly in process industry. Their work primarily locates in aviation industry. Third, they mainly proposed a collection of redundancy based architecture patterns, while we describe behavioral, flow propagation, and composition behaviors of production and safety systems with modeling patterns. A set of SAP is also listed in [97], where they focus on patterns of the redundancy and software faults.

It also deserves to learn from Case-based reasoning (CBR). CBR is the process of solving new problems based on the solutions to similar past problems [1]. It is carried out in four steps: retrieve, reuse, revise, and retain.

4.2 Categories of Modeling Patterns

Modeling Patterns (MP) are a general means of modeling frequently occurring functional and physical behaviors. They can be classified according to their purpose, which reflects what a modeling pattern works for. Modeling patterns can have either a behavioral, a flow propaga-

tion, or a composition purpose. Behavioral Patterns (BP) describe basic behaviors of components. For instance, the repairable behavior is regarded as a basic character in production and safety systems. Flow Propagation Patterns (FPP) depict flow propagations inside and between components. Composition Patterns (CP) represent cooperations in a system, such as the cooperation between main and standby units.

Figure 4.1 shows categories of modeling patterns. FPP can be further classified as intra-component FPP and inter-component FPP. Specific modeling patterns are discussed in detail in Chapter 5. In the following subsections, we discuss categories of modeling patterns in the framework of GTS.

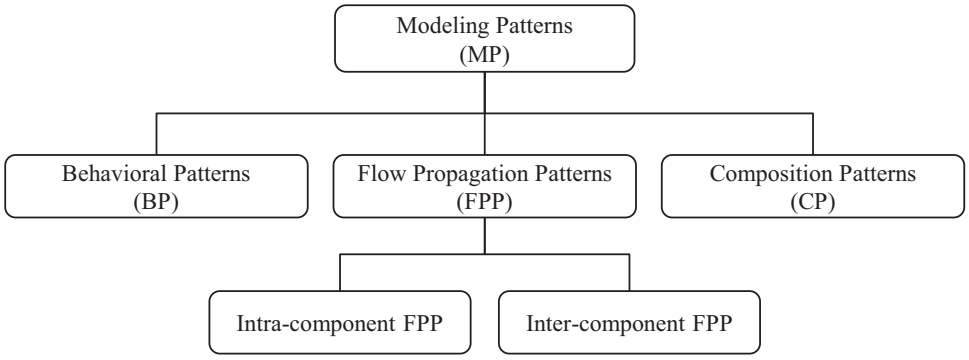


Figure 4.1: Categories of modeling patterns.

4.2.1 Behavioral Patterns

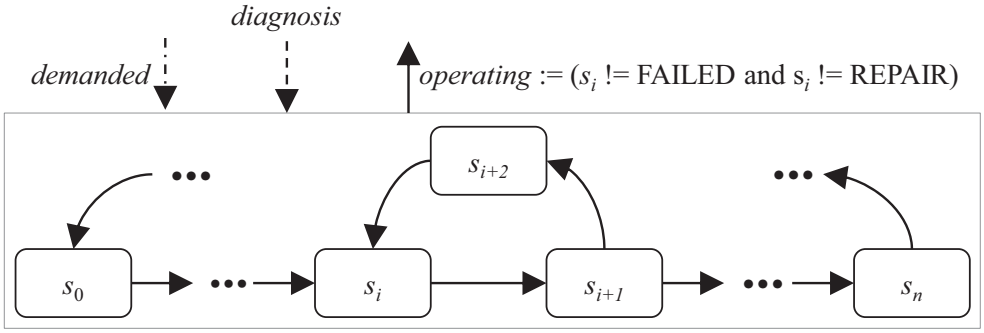


Figure 4.2: Guarded transition system for behavioral patterns.

Behavioral patterns capture the phenomena of internal state transitions at component level. Guarded transition system for a generic BP is shown in Figure 4.2. A BP is typically comprised of the following elements:

- State variables: s_i indicates the internal state of the component. Several applicable state variables identify component phases (e.g. preventive maintenance and periodic test).
- Flow variables: Boolean $demanded, diagnosis$, and $operating$. $demanded$ indicates the demand of activation of a component. $diagnosis$ indicates the diagnostic test used

to discover Dangerous Detected (DD) failures of a component. *operating* indicates if the component works or not.

- Events: changes between states with immediate or stochastic delays.
- Assertions: since the left member of an assertion is a flow variable, there are assignments for *demanded*, *diagnosis*, and *operating*.
- Input variables: *demanded* and *diagnosis*.
- Output variables: *operating*.

4.2.2 Flow Propagation Patterns

Flow propagation patterns capture flow-circulating behaviors inside (intra-) or between (inter-) components. A FPP is typically composed of the following elements:

- State variables: the current state is assigned by BP.
- Flow variables: Boolean *demanded*, *diagnosis*, and *operating*.
- Events: since the transitions are mainly between state variables, there are few events for FPP. Indeed, flow variables are assigned in assertions but not in transitions.
- Assertions: a set of assignments for flow variables.
- Input variables: *demanded* and *diagnosis*.
- Output variables: *operating*.

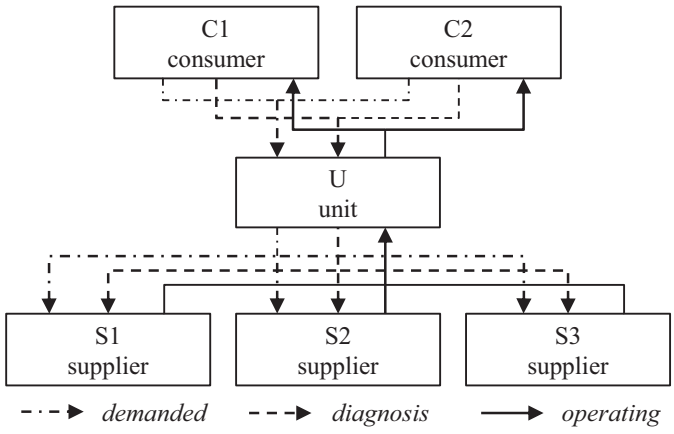


Figure 4.3: Flow propagations.

Flow propagations in a system (with three suppliers and two consumers) are shown in Figure 4.3. Unit U has three suppliers (upstream units S1, S2 and S3) and two consumers (downstream units C1 and C2). Note that the unit here can be a single component or a macro component composed of several components (elaborated in following section). *demanded* flow for each unit is composed of *demandedIn* and *demandedOut*. Analogously, *operatingIn* and *operatingOut* are associated with each block. This scheme can be extended to more suppliers

and consumers. Regarding unit U, we have typically:

$$
\begin{aligned}
U.demandedIn &:= C1.demandedOut \text{ or } C2.demandedOut \\
U.demandedOut &:= U.demandedIn \\
S1.demandedIn &:= U.demandedOut \\
S2.demandedIn &:= U.demandedOut \\
S3.demandedIn &:= U.demandedOut \\
U.operatingIn &:= S1.operating \text{ or } S2.operating \text{ or } S3.operating \\
U.operatingOut &:= U.operatingIn \text{ and } U.s != FAILED \text{ and } U.s != REPAIR \\
&\quad \text{and } U.diagnosis
\end{aligned}
$$

4.2.3 Composition Patterns

Composition patterns capture hierarchical composition phenomena between components. CP also share behaviors described in BP and FPP. A CP is typically made of the following elements:

- State variables: s_i indicates the internal state of a component. Several applicable state variables identify component phases (e.g. preventive maintenance and periodic test).
- Flow variables: Boolean $operating$, $demanded$ and $diagnosis$ when applicable.
- Events: changes between states with immediate or stochastic delays.
- Assertions: a set of assignments for flow variables.
- Input variables: $demanded$ and $diagnosis$.
- Output variables: $operating$.

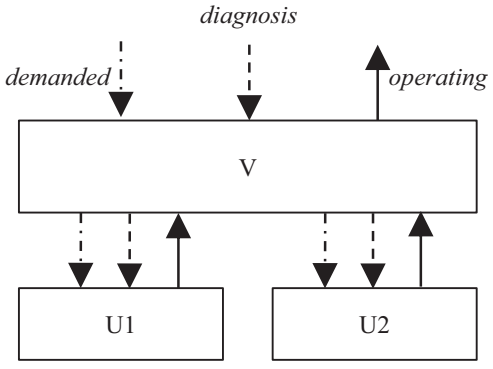


Figure 4.4: Hierarchical composition.

The hierarchical composition in a safety system is shown in Figure 4.4. A virtual macro unit (V) is created to gather two or more units (U1 and U2). The distribution of flows on children units is governed by composition type.

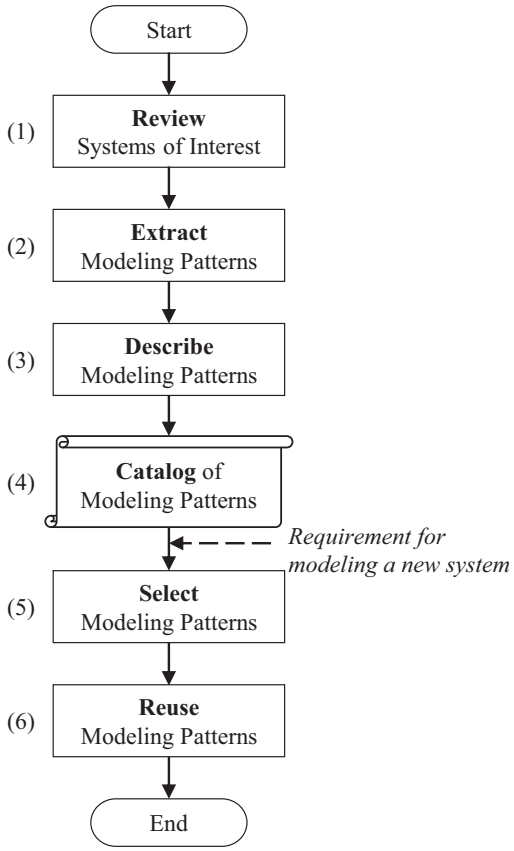


Figure 4.5: Methodology to develop modeling patterns.

4.3 Methodology to Develop Modeling Patterns

Based on the introduction of modeling patterns, we propose a methodology to develop modeling pattens. The methodology is shown in Figure 4.5, which is composed of six steps:

- Step (1): *Review* production and safety systems in the literature. A set of production and safety systems is reviewed, from the viewpoint of frequently occurring behaviors. That is, we focus on the way each component operates and the hierarchy structure of each system.
- Step (2): *Extract* modeling patterns from referred systems. Shared system behaviors are extracted and capitalized as modeling patterns. They are regarded general to cover most modeling behaviors of related systems. We record behavioral, flow propagation, and composition features of modeling patterns.
- Step (3): *Describe* modeling patterns with a finite set of structured items. To reuse modeling knowledge afterwards, we have to record each modeling pattern in a consistent format. For example, we need to take note of intent, motivation, and sample model for each modeling pattern. In a uniform way, we can make modeling patterns straightforward to be studied and reused. These items are inspired from design patterns [35] and introduced at the beginning of Chapter 5.
- Step (4): A *catalog* of modeling patterns can be proposed. Twenty-four (24) extracted modeling patterns are listed as a catalog. The catalog is further elaborated in Chapter 5.

Relationships between modeling pattens are discussed as well.

- Step (5): To model a new system, we *select* applicable modeling patterns from catalog for each component and subsystem. Modeling patterns may be integrated to model a complex unit. A methodology of modeling patterns reuse is discussed in Section 5.5.
- Step (6): Selected modeling patterns are *reused* to model targeted systems. Based on selected modeling patterns, we can model systems in a modular way. Experimental studies of reusing modeling patterns are illustrated thoroughly in Chapter 6. Simulation results obtained using modeling patterns approach are compared with those reported in the literature.

4.4 Summary

In this chapter, we discuss modeling patterns from several viewpoints. Initially, we discuss the notion of modeling patterns. According to their purpose, modeling patterns are classified as Behavioral Patterns (BP), Flow Propagation Patterns (FPP), and Composition Patterns (CP). A methodology of how to develop modeling patterns is proposed. The next chapter is devoted to exhibit the catalog of proposed modeling patterns.

Chapter 5

Catalog of Modeling Patterns

The objective of this chapter is to exhibit modeling patterns as catalog. Based on reviewing numerous production and safety systems from [4, 42, 5, 68, 74, 107, 111, 129, 41, 22, 36, 32, 114, 72, 6, 126, 102, 20, 85, 90, 29, 28, 60], twenty-four (24) modeling patterns are extracted, which include eight (8) Behavioral Patterns (BP), thirteen (13) Flow Propagation Patterns (FPP), and three (3) Composition Patterns (CP). A set of structured items is employed to display modeling pattens. Eventually, relationships between modeling patterns are discussed.

Each modeling pattern is presented with a set of structured items, which are adapted from [35]:

- Name: the name of a modeling pattern needs to be both generic and specific, that is, at a trade-off level of abstraction. The name acts as vocabulary for communicating in the domain-specific community.
- Intent: the function and the application scenario of a modeling pattern.
- Also Known As: the other usually-used name of the pattern. It is therefore optional for a specific pattern.
- Motivation: a scenario that elaborates model problem and how proposed pattern solves the problem.
- Structure: a graphical representation of a modeling pattern. Regarding BP and CP, we apply *graphical representations of Guarded Transition Systems* (GTS) as their structures. Concerning FPP, we use *dependency graphs* as their structures. Elements participating in the modeling pattern and their responsibilities are elaborated. The way that participants collaborate to take their responsibilities is outlined. The trade-offs and results by using modeling patterns are pointed out.
- Implementation: notes and hints that are required to be paid attention to when applying modeling patterns.
- Sample Model: the model fragments that show how to implement the pattern using a modeling formalism. We use *AltaRica 3.0 code* throughout the catalog.
- Known Uses: examples of the pattern found in realistic case studies.
- Related Patterns: the relationship and difference between the targeted pattern and relevant ones.

5.1 Behavioral Patterns

Behavioral patterns capture shared basic behaviors of components in production and safety systems. The structure of a BPP is given via its graphical representation of GTS.

5.1.1 PERFECT

Intent: Provides a general way to model components which cannot fail, that is, components which are perfectly reliable.
Motivation: Consider the perfect behavior where components can work in any scenario. Although perfect behavior may not be the real situation, some categories of components, such as oil wells may be assumed to be perfect in the literature.
Structure: The graphical representation of GTS for PERFECT is shown in Figure 5.1. There is only one state and no transition. The component state keeps invariant (i.e. WORKING).

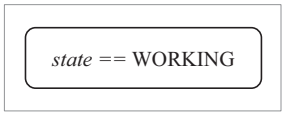


Figure 5.1: Graphical representation of GTS for PERFECT.

Implementation: The implementation of PERFECT pattern is straightforward.
Sample Model: The AltaRica 3.0 code of PERFECT is shown in Figure 5.2.

```
domain ComponentState {WORKING}
class PERFECT
    ComponentState varState (init = WORKING);
end
```

Figure 5.2: The AltaRica 3.0 code of PERFECT.

Known Uses: The well, separator, and pumps are assumed to be perfect in an offshore installation [129]. Perfect behaviors can also be found in the wellbore and surface manifold of a separation plant [90].

5.1.2 NonRepairable

Intent: Provides a generic way to model components that can fail but cannot be repaired.
Motivation: Many modern products can be treated as nonrepairable systems because of technological advances and their short-term technological obsolescence [31]. In production and safety systems, some components cannot be restored after their failures. For instance, uncovered DU (Dangerous Undetected) failures in safety systems cannot be detected and repaired.
Structure: The graphical representation of GTS for NonRepairable is shown in Figure 5.3. The component is initially in WORKING state. Once a failure occurs, the component state becomes FAILED.
Implementation: Consider following implementation issues when applying the pattern:

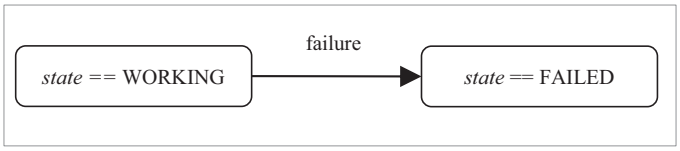


Figure 5.3: Graphical representation of GTS for NonRepairable.

- *Degradation may exist*: there may be intermediate states between WORKING and FAILED states.
- *Different lifetime distributions*: events (transitions) may folllow different lifetime distributions. Exponential distribution is one of the most widely used lifetime distributions.

Sample Model: The AltaRica 3.0 code of NonRepairable is shown in Figure 5.4.

```
domain ComponentState {WORKING, FAILED}
class Nonrepairable
    ComponentState varState (init = WORKING); //Initial state of component
    parameter Real Lambda = 1.0e-3; //Failure rate
    //Transition "failure" obeys exponential distribution with a rate "Lambda"
    event failure (delay = exponential(Lambda));
    transition
        //Failure transition is between WORKING and FAILED states
        failure: varState == WORKING -> varState := FAILED;
end
```

Figure 5.4: The AltaRica 3.0 code of NonRepairable.

Known Uses: In safety systems, DU failures are preventing activation on demand and can be revealed only by periodic tests. Part of DU failures cannot be covered by imperfect periodic tests that is the test coverage < 100%. Such DU failures can be modeled using this pattern.
Related Patterns: NonRepairable can be obtained by adding failure-related transitions and states to PERFECT pattern.

5.1.3 CorrectiveMaintenance

Intent: Provides a general way to model repair after the component's failure till its restoration.
Also Known As: Repair, Breakdown maintenance, Run-to-failure maintenance, Curative maintenance
Motivation: Corrective Maintenance (CM) is conducted after an item has failed. The target of CM is to bring component back to a functioning state. Many components are assumed to be repaired after their failures.
Structure: The graphical representation of GTS for CorrectiveMaintenance is shown in Figure 5.5. The component is initially working ($state$ == WORKING). Once a failure occurs, the component falls into FAILED state. When the corrective maintenance crew is available, the component state becomes UNDER_REPAIR. Finally, the component returns to its initial state once the repair operation is finished. In CorrectiveMaintenance pattern, if the state is WORKING, the component is up (available).
Implementation: Consider following issues when implementing this pattern:

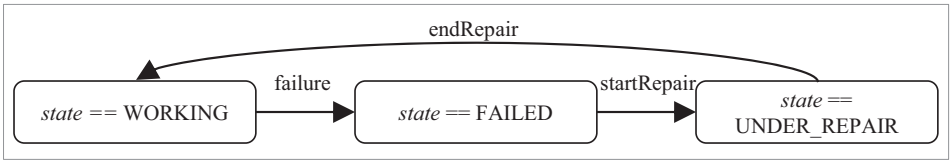


Figure 5.5: Graphical representation of GTS for CorrectiveMaintenance.

- *Repair crew*: with regard to a system, the number of repair crews is usually limited and predetermined. However, if the repair crew is assumed to be available at any time, the state UNDER_REPAIR can be omitted.
- *Different lifetime distributions*: involved events can follow different lifetime distributions. For instance, the failure transition may follow the exponential distribution. Nevertheless, the endRepair transition can follow a deterministic distribution.
- *Possible intermediate states*: there may be degradation states between WORKING and FAILED states.

Sample Model: The AltaRica 3.0 code of CorrectiveMaintenance is shown in Figure 5.6.

```
domain ComponentState {WORKING, FAILED, UNDER_REPAIR}
class CorrectiveMaintenance
    ComponentState varState (init = WORKING);
    parameter Real Lambda = 1.0e-3; //Failure rate
    parameter Real Mu     = 1.0e-1; //Repair rate
    event failure     (delay = exponential(Lambda));
    event startRepair (delay = 0); //No delay
    event endRepair   (delay = exponential(Mu));
    transition
        failure:     varState == WORKING      -> varState := FAILED;
        startRepair: varState == FAILED       -> varState := UNDER_REPAIR;
        endRepair:   varState == UNDER_REPAIR -> varState := WORKING;
end
```

Figure 5.6: The AltaRica 3.0 code of CorrectiveMaintenance.

Known Uses: CorrectiveMaintenance can be found in almost every system in process industry.
Related Patterns: CorrectiveMaintenance pattern can be obtained by assigning repair-related transitions and states to NonRepairable pattern.

5.1.4 PreventiveMaintenance

Intent: Provides a general way to model preventive maintenance policy which is carried out with predefined intervals and durations. The component is often assumed to be out of work under preventive maintenance.
Motivation: Preventive Maintenance (PM) is conducted according to a specified time schedule. PM seeks to reduce the failure probability of the component. It may involve inspection, adjustments, lubrication, parts replacement, calibration, and repair of items that are beginning to wear out. PM is generally performed on a regular basis, regardless of whether the functionality of performance is degraded or not [105].

Structure: The graphical representation of GTS for PreventiveMaintenance is shown in Figure 5.7. This can be seen as CorrectiveMaintenance pattern to which the preventive maintenance behavior is added. Besides transitions described in CorrectiveMaintenance pattern, if the preventive interval is reached and preventive maintenance crew is available, the component runs into preventive maintenance phase. If the component state is WORKING before changing its phase into preventive maintenance, the component keeps its state after transition endMaintenance. Otherwise, the component state becomes WORKING after endMaintenance.

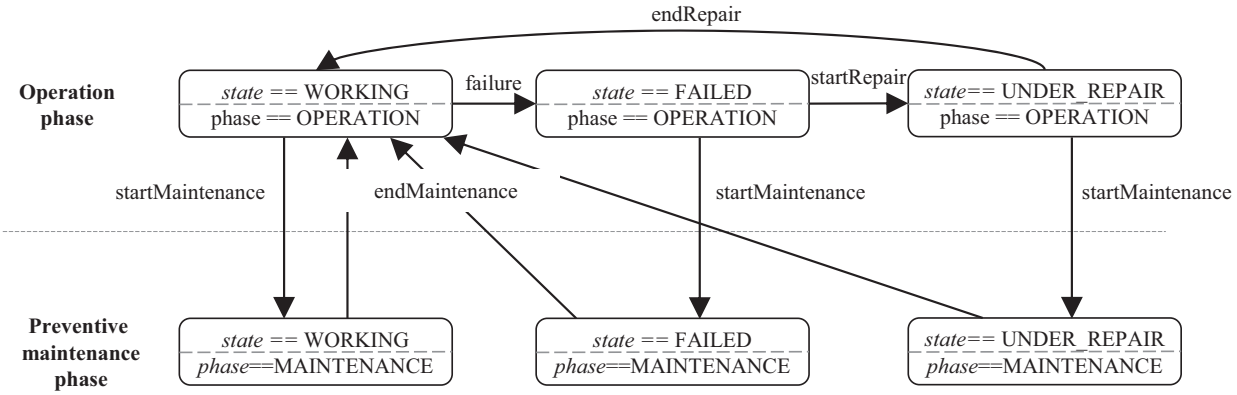


Figure 5.7: Graphical representation of GTS for PreventiveMaintenance.

Implementation: Be aware of that there may be more complex PM policy. For example, some components can work with three types of PM, such as every 3 months, 1 year, and 3 years. Normally, corresponding maintenance durations are different. We may use some state variables to define maintenance type and to count number of times of different maintenances.
Sample Model: The AltaRica 3.0 code of PreventiveMaintenance is shown in Figure 5.8.
Known Uses: PreventiveMaintenance can be used to model primary separators, heat exchangers, thermal chemical processor in a FPSO (Floating Production Storage and Offloading) system [85]. Components (two turbo-compressors and two turbo-generators) with three types of PM can be found in another offshore installation [129].
Related Patterns: PreventiveMaintenance can be obtained by adding PM-related states and transitions to CorrectiveMaintenance and DEGRADATION patterns.

5.1.5 DEGRADATION

Intent: Provides a generic way to model degraded behaviors between working and failed states. The component may degrade, in which situation the component becomes less available and more vulnerable.
Motivation: Traditional reliability models, which assume systems and their components can be either working perfectly or completely failed, are unable to characterize the multi-state nature of advanced engineering systems [63, 77]. Thus degradation behaviors are required to be studied. For example, working conditions in oil and gas systems are usually harsh. Exposed to such circumstances, components are prone to degrade.
Structure: The graphical representation of GTS for DEGRADATION is shown in Figure 5.9. Once there is a degradation, the component transfers to DEGRADED state. If a repair crew is avail-

```
domain ComponentState {WORKING, FAILED, UNDER_REPAIR} //States of component
domain Phase {OPERATION, MAINTENANCE} //Phases of component
class PreventiveMaintenance
    ComponentState varState (init = WORKING);
    Phase phase (init = OPERATION);
    parameter Real Lambda = 1.0e-3; //Failure rate
    parameter Real Mu     = 1.0e-1; //Repair rate
    parameter Real PreventiveInterval = 8760.0; //Maintenance interval
    parameter Real PreventiveDuration = 10.0;   //Maintenance duration
    event failure     (delay = exponential(Lambda));
    event startRepair (delay = 0);
    event endRepair   (delay = exponential(Mu));
    event startMaintenance (delay = PreventiveInterval);
    event endMaintenance   (delay = PreventiveDuration);
    transition
        failure:     varState == WORKING and phase == OPERATION ->
                    {varState := FAILED; phase := OPERATION;}
        startRepair: varState == FAILED and phase == OPERATION ->
                    {varState := UNDER_REPAIR; phase := OPERATION;}
        endRepair:   varState == UNDER_REPAIR and phase == OPERATION ->
                    {varState := WORKING; phase := OPERATION;}
        startMaintenance: phase == OPERATION -> phase := MAINTENANCE;
        endMaintenance:   phase == MAINTENANCE ->
                         {varState := WORKING; phase := OPERATION;}
end
```

Figure 5.8: The AltaRica 3.0 code of PreventiveMaintenance.

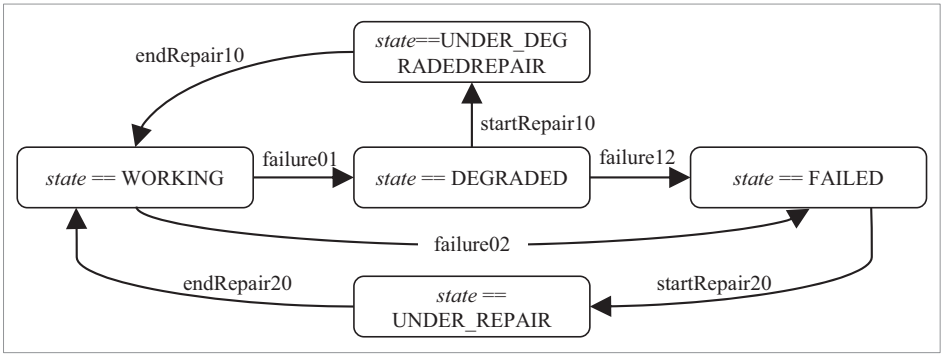


Figure 5.9: Graphical representation of GTS for DEGRADATION.

able, the component state turns into UNDER_DEGRADEDREPAIR. The degraded component can also fall into FAILED state. If the corrective repair crew is available, the component transfers from FAILED to UNDER_REPAIR. The component returns to WORKING if the repair is finished.
Implementation: Here are four implementation issues to be considered:

- *The DEGRADATION and UNDER_DEGRADEDREPAIR states may be assumed as up states*, that is, the component works in full capacity. But in some situations, the component is assumed to work with decreased or null capacity.
- More intermediate states can be applied in degradation process.
- A degraded component can also be nonrepairable, by deleting repair-related states and transitions.

- *Weibull like distribution may be used.* Here we use multi-state models to define finite degradation states. Some systems are required to apply other lifetime distributions, like Weibull distribution (with shape parameter > 1), to model degradation behaviors.

Sample Model: The AltaRica 3.0 code of DEGRADATION is shown in Figure 5.10.

```
domain ComponentState {WORKING, FAILED, UNDER_REPAIR, DEGRADED, UNDER_DEGRADEDREPAIR}
class DEGRADATION
    ComponentState varState (init = WORKING);
    parameter Real Lambda01 = 0.79e-3; //Failure rate from WORKING to DEGRADED
    parameter Real Lambda12 = 1.86e-3; //Failure rate from DEGRADED to FAILED
    parameter Real Lambda02 = 0.77e-3; //Failure rate from WORKING to FAILED
    parameter Real Mu10     = 3.20e-2; //Repair rate from DEGRADED to WORKING
    parameter Real Mu20     = 3.80e-2; //Repair rate from FAILED to WORKING
    event failure01 (delay = exponential(Lambda01));
    event failure12 (delay = exponential(Lambda12));
    event failure02 (delay = exponential(Lambda02));
    event startRepair10 (delay = 0);
    event startRepair20 (delay = 0);
    event endRepair10   (delay = exponential(Mu10));
    event endRepair20   (delay = exponential(Mu20));
    transition
        failure01:     varState == WORKING  -> varState := DEGRADED;
        startRepair10: varState == DEGRADED -> varState := UNDER_DEGRADEDREPAIR;
        endRepair10:   varState == UNDER_DEGRADEDREPAIR -> varState := WORKING;
        failure12:     varState == DEGRADED             -> varState := FAILED;
        failure02:     varState == WORKING              -> varState := FAILED;
        startRepair20: varState == FAILED               -> varState := UNDER_REPAIR;
        endRepair20:   varState == UNDER_REPAIR         -> varState := WORKING;
end
```

Figure 5.10: The AltaRica 3.0 code of DEGRADATION.

Known Uses: Since the severe working conditions (e.g. high temperature and high pressure) in oil industry, related equipment reveal degradation phenomena. Degradation behaviors can be found in an oil production system (treatment units) [107] and an offshore installation (two turbo-compressors and two turbo-generators) [129].
Related Patterns: DEGRADATION can be obtained by adding intermediate transitions and states to CorrectiveMaintenance or NonRepairable patterns.

5.1.6 PeriodicTest

Intent: Provides a general way to model periodic test which is used to reveal dangerous undetected failures.
Also Known As: Proof test, Proof testing, Periodic testing
Motivation: A periodic test is a planned operation performed at constant time intervals to detect hidden failures [60].
Structure: The graphical representation of GTS for PeriodicTest is shown in Figure 5.11. Periodic tests are conducted at predefined intervals and durations.

- *PeriodicTest*: indicates the state of the periodic test. The periodic test is conducted when the *PeriodicTest* is true.

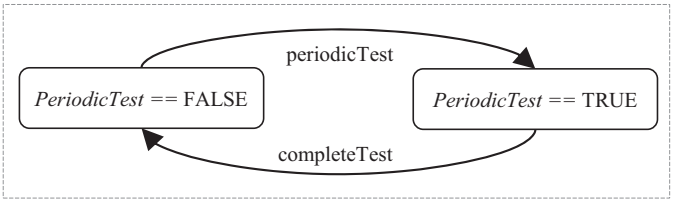


Figure 5.11: Graphical representation of GTS for PeriodicTest.

- Transitions: reveal transfers between the *PeriodicTest* states.

Implementation: Here are two implementation issues to be considered:

- It is usually used in safety systems.
- It is applied for revealing dangerous undetected failures.

Sample Model: The AltaRica 3.0 code of PeriodicTest is shown in Figure 5.12.

```
class PeriodicTest
    Boolean PeriodicTestDetectFailure ( init = false );//State of periodic test
    parameter Real PeriodicTestInterval = 8760.0;//Intervals between tests
    parameter Real PeriodicTestDuration = 10.0;  //Test duration
    event periodicTest ( delay = PeriodicTestInterval );
    event completeTest ( delay = PeriodicTestDuration );
    transition
        periodicTest: PeriodicTestDetectFailure == false
                  ->  PeriodicTestDetectFailure := true;
        completeTest: PeriodicTestDetectFailure == true
                  ->  PeriodicTestDetectFailure := false;
end
```

Figure 5.12: The AltaRica 3.0 code of PeriodicTest.

Known Uses: This pattern models the periodic test which can detect Dangerous Undetected (DU) failures. It can be used to model periodic tests in all safety systems in ISO/TR 12489 [60].
Related Patterns: PeriodicTest serves as basis of RevealUndetectedFailure and StaggeredPeriodicTest.

5.1.7 RevealUndetectedFailure

Intent: Provides a generic way to model the process to detect dangerous undetected failures in safety systems.
Motivation: To model how to reveal dangerous undetected failures.
Structure: The graphical representation of GTS for RevealUndetectedFailure is shown in Figure 5.13.
Implementation: Two implementation matters of this pattern deserve to be underlined:

- DU failures can only be discovered when *state*==DU and *phase*==TEST. That is, when *state*==DU, periodic test can be completed only after DU failures are revealed.
- After DU failures are detected, the component evolves as CorrectiveMaintenance pattern.

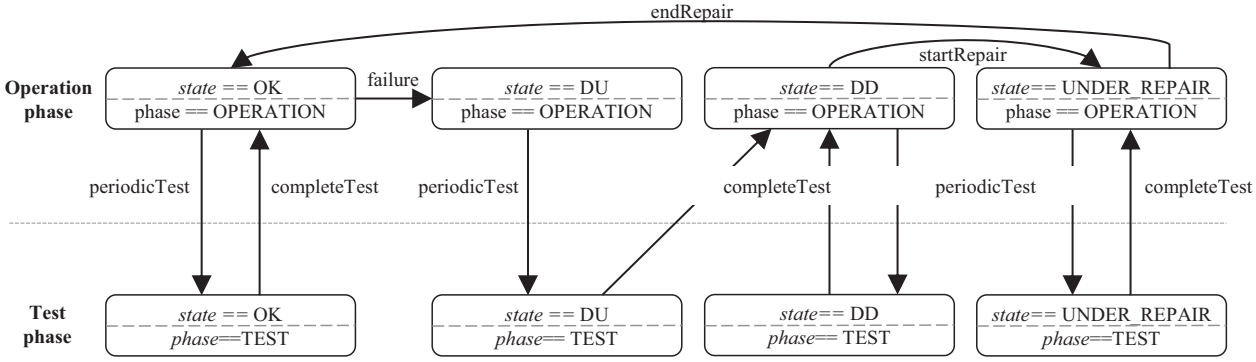


Figure 5.13: Graphical representation of GTS for RevealUndetectedFailure.

Sample Model: The AltaRica 3.0 code of RevealUndetectedFailure is shown in Figure 5.14.
Known Uses: This pattern can be employed to model behaviors of revealing DU failures in all safety systems in ISO/TR 12489 [60].
Related Patterns: RevealUndetectedFailure is proposed based on PeriodicTest and Corrective-Maintenance patterns.

5.1.8 StaggeredPeriodicTest

Intent: Provides a universal way to model the staggered periodic test, which can obtain higher availability than simultaneous tests.
Motivation: In simultaneous tests, redundant components are tested simultaneously. This strategy is not applied for safety systems that (is required to) remain in service permanently because the system is unavailable during test. When redundant components of a system are tested concurrently, unavailabilities of components *peak* at the same time. This has an adverse effect on system availability which can be mitigated by de-synchronizing tests [60]. A practical way to do that is staggering these tests. A *staggered testing* (of redundant items) is a test of several items with the same test interval but not at the same time. The unavailability peaks are also staggered and this improves average availability of the system [60, 78].

When several components are applied in redundant structures, tests may be staggered, in which situation the first test interval is different from others [50]. The sketch diagram of staggered periodic test is shown in Figure 5.15, where τ is the test interval. When several components are involved, the periodic test may be staggered, thus to increase the likelihood of detecting CCF (Common Cause Failures) and improve system availability. Compared with a reference periodic test, the duration of the first test interval in staggered periodic test is different (e.g. mτ, 0<m<1) from the duration of following test intervals (e.g. τ).
Structure: The graphical representation of GTS for StaggeredPeriodicTest is shown in Figure 5.16. Initially, the startStaggeredTest is triggered. Subsequently, the rest of the pattern architecture becomes similar to PeriodicTest pattern in Figure 5.11.

- *PeriodicTest*: indicates the state of periodic test.
- *startStaggeredTest*: indicates the start of staggered test. If the staggered test starts, the periodic test initiates concurrently. After the periodic test is conducted, startStaggeredTest cannot be enabled the second time.

```
domain ComponentState {OK, DU, DD, UNDER_REPAIR} //States of component
domain Phase {OPERATION, TEST} //Phases of component
class RevealUndetectedFailure
    ComponentState varState ( init = OK );
    Phase phase ( init = OPERATION );
    Boolean ComponentAvailable ( init = true ); //Component availability
    parameter Real LambdaDU = 1.0e-3; //Dangerous Undetected (DU) failure rate
    parameter Real Mu       = 1.0e-1; //Repair rate
    parameter Real PeriodicTestInterval = 8760.0;//Intervals between tests
    parameter Real PeriodicTestDuration = 10.0;  //Test duration
    event failure      ( delay = exponential ( LambdaDU ) );
    event startRepair  ( delay = 0 );
    event endRepair    ( delay = exponential ( Mu ) );
    event periodicTest ( delay = PeriodicTestInterval );
    event completeTest ( delay = PeriodicTestDuration );
    transition
        failure: varState == OK and phase == OPERATION and ComponentAvailable == true
             -> {varState := DU; ComponentAvailable := false;}
        startRepair: varState == DD and phase == OPERATION -> varState := UNDER_REPAIR;
        endRepair:   varState == UNDER_REPAIR and phase == OPERATION
                 -> {varState := OK; ComponentAvailable := true;}
        periodicTest: phase == OPERATION -> phase := TEST;
        completeTest: phase == TEST
                  -> {phase := OPERATION; if varState == DU then varState := DD;}
end
```

Figure 5.14: The AltaRica 3.0 code of RevealUndetectedFailure.

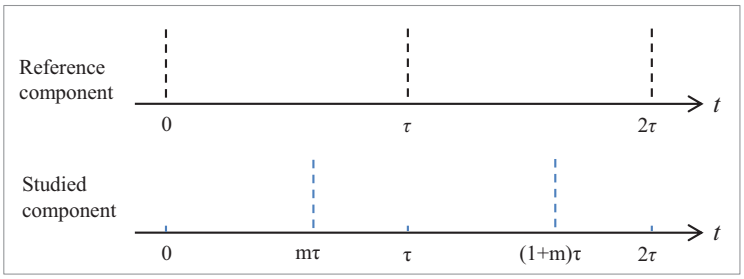


Figure 5.15: Sketch diagram of staggered periodic test.

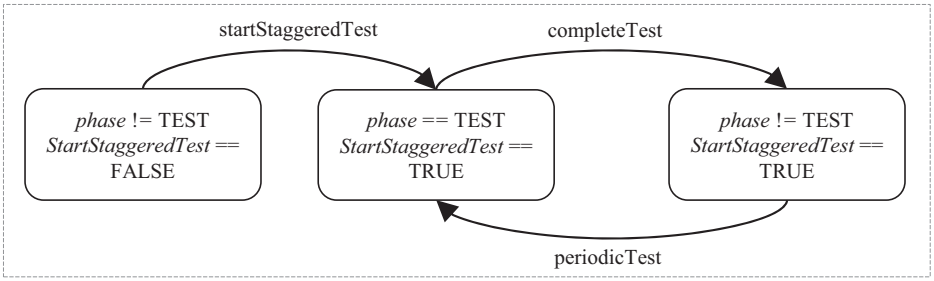


Figure 5.16: Graphical representation of GTS for StaggeredPeriodicTest.

Sample Model: The AltaRica 3.0 code of StaggeredPeriodicTest is shown in Figure 5.17.

```
domain ComponentState {OK, DU, DD, UNDER_REPAIR} //States of component
domain Phase {OPERATION, TEST} //Phases of component
class StaggeredPeriodicTest
    ComponentState varState ( init = OK );
    Phase phase ( init = OPERATION );
    Boolean ComponentAvailable ( init = true ); //Component availability
    Boolean StartStaggeredTest ( init = false );//Check if staggered test started
    parameter Real LambdaDU = 1.0e-3; //Dangerous Undetected (DU) failure rate
    parameter Real Mu       = 1.0e-1; //Repair rate
    parameter Real PeriodicTestInterval = 8760.0;//Intervals between tests
    parameter Real PeriodicTestDuration = 10.0;  //Test duration
    parameter Real StaggeredFactor = 0.5;//Determine time to start staggered test
    event startStagger ( delay = PeriodicTestInterval * StaggeredFactor );
    event failure      ( delay = exponential ( LambdaDU ) );
    event startRepair  ( delay = 0 );
    event endRepair    ( delay = exponential ( Mu ) );
    event periodicTest ( delay = PeriodicTestInterval );
    event completeTest ( delay = PeriodicTestDuration );
    transition
        failure: varState == OK and phase == OPERATION and ComponentAvailable == true
             -> {varState := DU; ComponentAvailable := false;}
        startRepair: varState == DD and phase == OPERATION -> varState := UNDER_REPAIR;
        endRepair:   varState == UNDER_REPAIR and phase == OPERATION
                 -> {varState := OK; ComponentAvailable := true;}
        startStagger: phase != TEST and StartStaggeredTest == false
                  -> {phase := TEST; StartStaggeredTest := true;}
        periodicTest: phase == OPERATION and StartStaggeredTest == true -> phase := TEST;
        completeTest: phase == TEST and StartStaggeredTest == true
                  -> {phase := OPERATION; if varState == DU then varState := DD;}
end
```

Figure 5.17: The AltaRica 3.0 code of StaggeredPeriodicTest.

Known Uses: This pattern can be employed to model staggered periodic test for the pressure sensors and isolation valves in a safety system (Typical Application 2-4 in ISO/TR 12489) [60].
Related Patterns: StaggeredPeriodicTest is based on PeriodicTest pattern.

5.2 Flow Propagation Patterns

Flow Propagation Patterns (FPP) depict flow propagation inside and between components. The structure of a FPP is illustrated with the dependency graph of assertions. An assertion is used to update flow variables on the basis of state variables.

5.2.1 SISO: Single-Input-Single-Output

Intent: Provides a general way to model physical structure with one inflow and one outflow.
Motivation: Consider a component with one inflow and one outflow. The flow chart of the component is shown in the following, where C is a targeted component, I and O are its inflow and outflow, respectively.
Structure: Dependency graph of the assertion of SISO is shown in Figure 5.18.

- varInFlow: defines the component inflow, which can be real or Boolean variable. It may be treated as a constant parameter.
- varState: declares the component state, which is assigned by using *behavioral patterns*.
- varOutFlow: monitors the component output which relies on its inflow and state. varOutFlow equals to varInFlow when the varState is up.

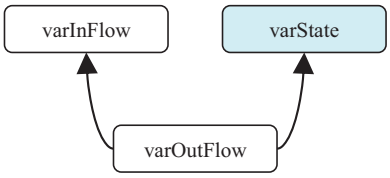


Figure 5.18: Dependency graph of the assertion of SISO.

Implementation: The implementation of SISO is straightforward.
Sample Model: The AltaRica 3.0 code of SISO is shown in Figure 5.19.

```
class SISO //Single-Input-Single-Output
    Component C;//The behavioral pattern of component is predefined
    Real varInFlow  (reset = 0.0);
    Real varOutFlow (reset = 0.0);
    assertion//Treat WORKING as the up state
        varOutFlow := if C.varState == WORKING then varInFlow else 0.0;
end
```

Figure 5.19: The AltaRica 3.0 code of SISO.

By considering capacity constraints of components, we propose to add flow variables *demands* for each component. *demands* are requirements from downstream components. SISO has Demand Input (DI), Demand Output (DO), Flow Input (FI), and Flow Output (FO). Among them, FI and FO are physical flows. The running scheme considering capacity limit in SISO is shown in Figure 5.20. Variables can be defined in SISO. The partial AltaRica 3.0 code is placed in assertion.

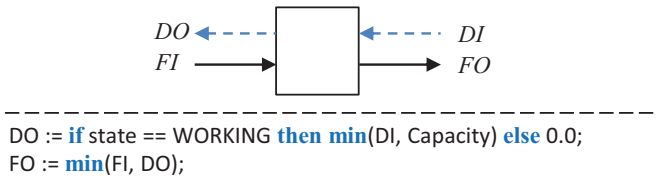


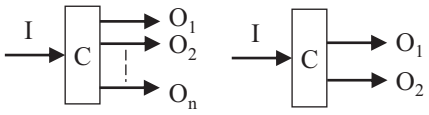
Figure 5.20: Running scheme considering capacity limit in SISO.

Known Uses: SISO can be frequently used in process industry. For example, this pattern can be employed to model the treatment units in an oil extraction installation [107] and the valves in a Christmas tree sitting above a basic wellhead [102].

Related Patterns: SISO is applied in other patterns, especially inter-component flow propagation patterns at subsystem level. SERIES and PARALLEL are separately the series and parallel connections of several SISO. SISO is a special case of MIMO.

5.2.2 SIMO: Single-Input-Multiple-Output

Intent: Provides a general way to model the physical structure that has one inflow and multiple outflows.
Also Known As: Divergence
Motivation: Consider a dehydration unit, which has one inflow and multiple outflows. We can judge the outputs by the component state and downstream demands. Demands are associated with outflows.



Structure: Dependency graph of the assertion of SIMO is shown in Figure 5.21. The varOutFlow1 depends on varInFlow, varState, as well as two demands. The way of modeling varOutFlow2 is similar to that of varOutFlow1.

- varInFlow: defines the component inflow.
- varState: declares the component state.
- demand1, demand2: are downstream demands corresponding to varOutFlow1 and varOutFlow2.
- varOutFlow1, varOutFlow2: observe outputs of the component, which rely on inflow, state, and demands. varOutFlow1 and varOutFlow2 are positive when demands are true and varState is up.

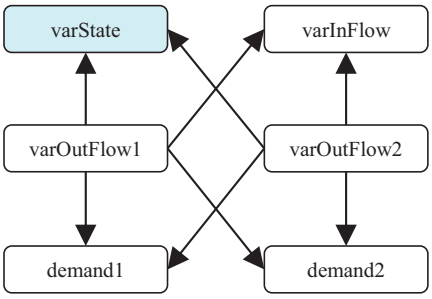
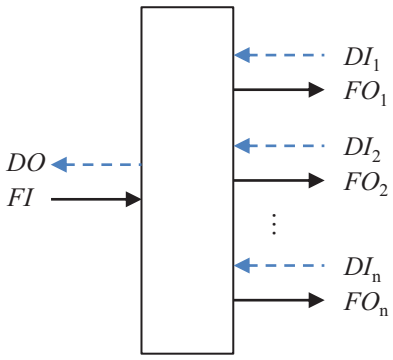


Figure 5.21: Dependency graph of the assertion of SIMO.

Implementation: This pattern needs to be adjusted according to the distribution policy. For example, the policy can be governed by preference (priority).
Sample Model: The AltaRica 3.0 code of SIMO is shown in Figure 5.22.

```
//Assume an equally split policy (i.e., 50%:50%), which needs to be predefined
//Assume no capacity limit
class SIMO //Single-Input-Multiple-Output
    Component C;//The behavioral pattern of component is predefined
    Real varInFlow, varOutFlow1, varOutFlow2 (reset = 0.0);
    Boolean demand1, demand2 (reset = false);
    assertion//Treat WORKING as the up state
        varOutFlow1 := if demand1 and demand2 and C.varState == WORKING then 0.5*varInFlow
                  else if demand1 and not demand2 and C.varState == WORKING then varInFlow
                  else 0;
        varOutFlow2 := if demand2 and demand1 and C.varState == WORKING then 0.5*varInFlow
                  else if demand2 and not demand1 and C.varState == WORKING then varInFlow
                  else 0;
end
```

Figure 5.22: The AltaRica 3.0 code of SIMO.




```
//We assume three downstream components, i.e. n=3
DO := if state == WORKING then min (DI1+DI2+DI3, Capacity) else 0.0;
FO1 := if DI1 > 0 then min(FI, DO) * DI1/(DI1 + DI2 + DI3) else 0.0;
FO2 := if DI2 > 0 then min(FI, DO) * DI2/(DI1 + DI2 + DI3) else 0.0;
FO3 := if DI3 > 0 then min(FI, DO) * DI3/(DI1 + DI2 + DI3) else 0.0;
```

Figure 5.23: Running scheme considering capacity limit in SIMO.

By considering capacity constraints, SIMO has demand inputs, demand output, flow input, and flow outputs. The running scheme considering capacity limit in SIMO is shown in Figure 5.23.

Known Uses: SIMO is commonly applied in oil and gas systems, such as a dehydration unit with one inflow from the gas compressor and several outflows [111, 129], and a flowline-riser loop in a subsea oil production system [126].

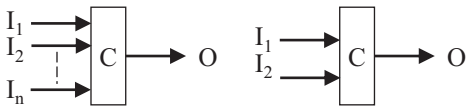
Related Patterns: SIMO is a special case of MIMO.

5.2.3 MISO: Multiple-Input-Single-Output

Intent: Provides a general way to model the physical structure that has multiple inflows and one outflow.

Also Known As: Convergence

Motivation: Consider a component with two inflows and one outflow. It is straightforward to obtain the outflow by the sum of inflows, as well as considering component state.



Structure: Dependency graph of the assertion of MISO is shown in Figure 5.24. The varOutFlow of MISO depends on its varState and varInFlows.

- varInFlow1, varInFlow2: stand for component inflows.
- varState: declares the component state.
- varOutFlow: observes the component output, which relies on inflows and component state.

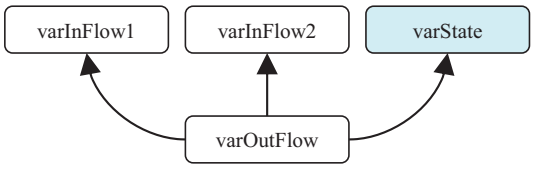


Figure 5.24: Dependency graph of the assertion of MISO.

Implementation: Here we assume that flows are mono-flux and no chemical reaction occurs.

Sample Model: The AltaRica 3.0 code of MISO is shown in Figure 5.25.

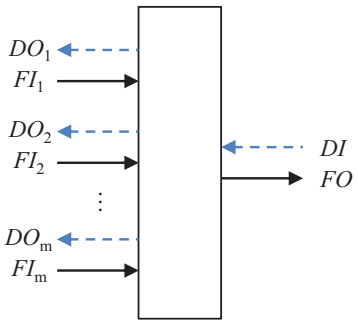
By considering capacity constraints, MISO has demand input, demand outputs, flow inputs, and flow output. The running scheme considering capacity limit in MISO is shown in Figure 5.26.

Known Uses: In an oil production system [107], MISO can be used to model a treatment unit with two inflows (from a well and another treatment unit) and one outflow. In a FPSO system [85], a sea water cooler can be modeled by MISO, which has two inflows (from heat exchangers) and one outflow (to a cargo tank).

Related Patterns: When inflows are mono-flux, MISO and MIMO can be modeled in a similar way.

```
class MISO //Multiple-Input-Single-Output
    Component C;//The behavioral pattern of component is predefined
    Real varInFlow1, varInFlow2 (reset= 0.0);
    Real varOutFlow (reset = 0.0);
    assertion//Treat WORKING as the up state
        varOutFlow := if C.varState == WORKING then (varInFlow1 + varInFlow2) else 0.0;
end
```

Figure 5.25: The AltaRica 3.0 code of MISO.

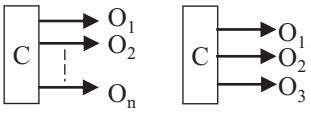



```
//We assume two upstream components, U1, U2, i.e. m=2
//We define new variable DO – Demand Output
//Capacity-based distribution
DO  := if state == WORKING then min(DI, Capacity) else 0.0;
DO1 := if U1.state == WORKING and U2.state == WORKING then DO*U1.Capacity/(U1.Capacity+U2.Capacity)
       else if U1.state == WORKING and U2.state != WORKING then DO
       else 0.0;
DO2 := if U1.state == WORKING and U2.state == WORKING then DO*U2.Capacity/(U1.Capacity+U2.Capacity)
       else if U2.state == WORKING and U1.state != WORKING then DO
       else 0.0;
FO := min(FI1, DO1) + min(FI2, DO2);
```

Figure 5.26: Running scheme considering capacity limit in MISO.

5.2.4 SOURCE

Intent: Provides a general way to model the physical structure with only outflows.
Motivation: Consider a structure with merely outflows, which can be regarded as the start of systems (e.g. oil wells, electricity generators).



Structure: Dependency graph of the assertion of SOURCE is shown in Figure 5.27.

- varState: indicates component status.
- varOutFlow1, varOutFlow2, varOutFlow3: define component outputs, which only depend on component state.

Implementation: Even though some components have “inflows”, they are still modeled using SOURCE pattern. For example, electricity generators have gas inflow and electricity outflow.

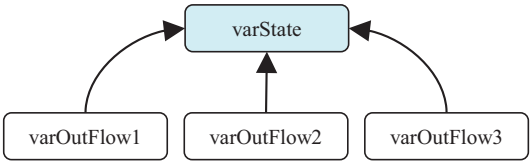


Figure 5.27: Dependency graph of the assertion of SOURCE.

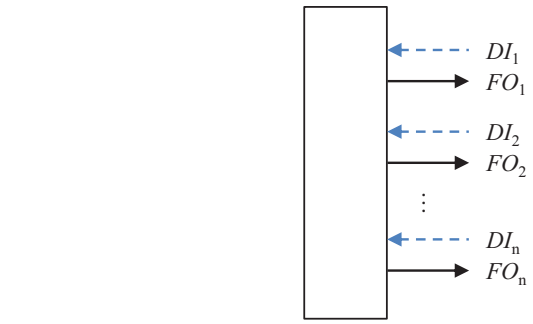
Because their inflow and outflow carry different media, we model them using SOURCE rather than SISO pattern.

Sample Model: The AltaRica 3.0 code of SOURCE is shown in Figure 5.28.

```
class SOURCE
    Component C;//The behavioral pattern of component is predefined
    //Used by downstream components
    Real varOutFlow1, varOutFlow2, varOutFlow3 (reset = 0.0);
end
```

Figure 5.28: The AltaRica 3.0 code of SOURCE.

By considering capacity constraints, SOURCE has demand inputs and flow outputs. The running scheme considering capacity limit in SOURCE is shown in Figure 5.29.




```
//We assume three downstream components, n=3
//Multi-flux, three types of capacity
FO1 := if state == WORKING then min(DI1, Capacity1) else 0.0;
FO2 := if state == WORKING then min(DI2, Capacity2) else 0.0;
FO3 := if state == WORKING then min(DI3, Capacity3) else 0.0;
//Mono-flux, one capacity
FO1 := if state == WORKING and DI1 > 0 then Capacity*DI1/(DI1+DI2+DI3) else 0.0;
FO2 := if state == WORKING and DI2 > 0 then Capacity*DI2/(DI1+DI2+DI3) else 0.0;
FO3 := if state == WORKING and DI3 > 0 then Capacity*DI3/(DI1+DI2+DI3) else 0.0;
```

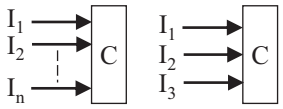
Figure 5.29: Running scheme considering capacity limit in SOURCE.

Known Uses: SOURCE can be used to model wells and turbo generators in offshore plants [129, 90].

Related Patterns: SOURCE usually collaborates with MIMO pattern. SOURCE is also a special case of MIMO.

5.2.5 SINK

Intent: Provides a general way to model the physical structure with only inflows.
Motivation: Consider a tank in a production system, which solely has inflows.



Structure: Dependency graph of the assertion of SINK is shown in Figure 5.30.

- varInFlow1, varInFlow2, varInFlow3: define component inflows.
- varState: indicates the component state.
- volume: monitors inflows. The volume, a flow variable acting as observer, is treated as "outflow" of the component, which relies on its state and inflows.

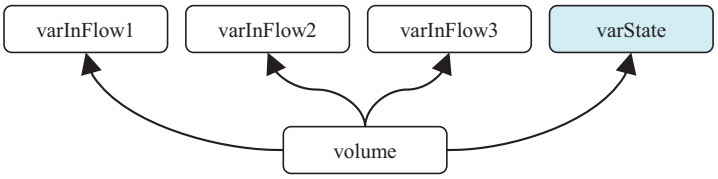


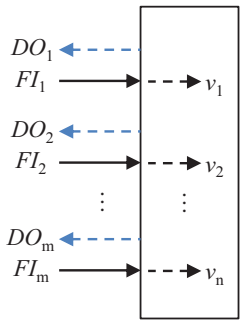
Figure 5.30: Dependency graph of the assertion of SINK.

Implementation: Inflows are usually mono-flux, that is, the materials in the flow are identical. In real cases, SINK structures (e.g. a tank) have outflows. However, when export valves/pipelines are closed, such structures can be modeled using SINK pattern. For example, in a FPSO, the oil inside its cargo tank is transferred to shuttle tankers periodically. Except offloading stage, the oil cargo tank can be modeled with SINK pattern.
Sample Model: The AltaRica 3.0 code of SINK is shown in Figure 5.31.

```
//Mono-flux, assume no capacity limit
class SINK
    Component C;//The behavioral pattern of component is predefined
    Real varInFlow1, varInFlow2, varInFlow3 (reset = 0.0);
    Real volume (reset = 0.0); //Virtual variable, used for monitoring
    assertion//Treat WORKING as the up state
      volume := if C.varState == WORKING then (varInFlow1+varInFlow2+varInFlow3) else 0.0;
end
```

Figure 5.31: The AltaRica 3.0 code of SINK.

By considering capacity constraints, SINK has demand outputs and flow inputs. The running scheme considering capacity limit in SINK is shown in Figure 5.32.
Known Uses: The SINK pattern can be applied to model the tanks in an oil production system [107] and the oil cargo tank in a FPSO system [85].
Related Patterns: SINK is regarded as a special case of MIMO.

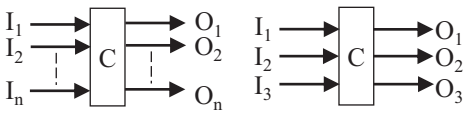



```
//We assume two upstream components, U1, U2, i.e. m=2
//We define new variable DO – Demand Output
//We define v1, v2 for observations
//Capacity-based distribution
DO := if state == WORKING then Capacity else 0.0;
DO1 := if U1.state == WORKING and U2.state == WORKING then DO*U1.Capacity/(U1.Capacity+U2.Capacity)
       else if U1.state == WORKING and U2.state != WORKING then DO
       else 0.0;
DO2 := if U1.state == WORKING and U2.state == WORKING then DO*U2.Capacity/(U1.Capacity+U2.Capacity)
       else if U2.state == WORKING and U1.state != WORKING then DO
       else 0.0;
v1 := min(FI1, DO1);
v2 := min(FI2, DO3);
```

Figure 5.32: Running scheme considering capacity limit in SINK.

5.2.6 MIMO: Multiple-Input-Multiple-Output

Intent: Provides a generic way to model the physical structure that has multiple inflows and multiple outflows.
Motivation: Consider a unit which has m ($m > 1$) inflows and n ($n > 1$) outflows.



Structure: Dependency graph of the assertion of MIMO is shown in Figure 5.33. The varOutFlow1 of MIMO depends on varState and varInFlow1. The way to model varOutFlow2 and varOutFlow3 is similar to varOutFlow1.

- varInFlow1, varInFlow2, varInFlow3: define component inflows.
- varState: represents the component state.
- varOutFlow1, varOutFlow2, varOutFlow3: observe component outflows, which rely on inflows and component sate. All outflows depend on associated inflows and component state.

Implementation: Here are two implementation matters to be considered:

- Judge if inflows are mono-flux or multi-flux.
- Consider using other flow propagation patterns if necessary. For instance, MIMO (e.g. separators) usually works together with SOURCE (e.g. wells).

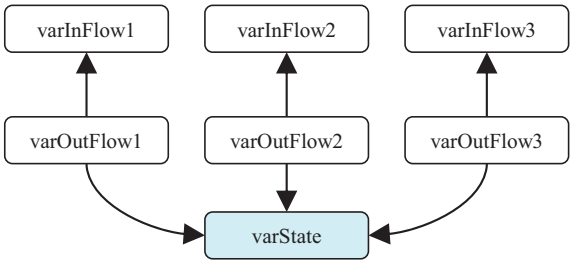


Figure 5.33: Dependency graph of the assertion of MIMO.

Sample Model: The AltaRica 3.0 code of MIMO is shown in Figure 5.34. The AltaRica 3.0 code of mono-flux MIMO is similar to that of SIMO. With regard to multi-flux MIMO, its flow allocation is predetermined.

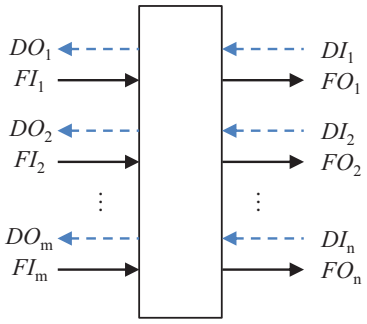
```
//The flow allocation is predetermined, let us assume I1→O1,I2→O2,I3→O3
class MIMO //Multiple-Input-Multiple-Output
    Component C;//The behavioral pattern of component is predefined
    Real varInFlow1, varInFlow2, varInFlow3    (reset = 0.0);
    Real varOutFlow1, varOutFlow2, varOutFlow3 (reset = 0.0);
    assertion//Treat WORKING as the up state
        varOutFlow1 := if C.varState == WORKING then varInFlow1 else 0.0;
        varOutFlow2 := if C.varState == WORKING then varInFlow2 else 0.0;
        varOutFlow3 := if C.varState == WORKING then varInFlow3 else 0.0;
end
```

Figure 5.34: The AltaRica 3.0 code of MIMO.

By considering capacity constraints, MIMO has demand inputs, demand outputs, flow inputs, and flow outputs. The running scheme considering capacity limit in MIMO is shown in Figure 5.35. Note that there are two scenarios in modeling MIMO structures. First, when flows are multi-flux, for example, a separator with oil, gas and water flows, we model each type of flow separately. Capacity of the structure is predefined for each flow. Second, when flows are mono-flux, that is, only one type of flow circulates the component. Demand outputs are assigned according to states and capacities of upstream components.
Known Uses: MIMO can be applied to model a thermal chemical processor in an offshore production system [85], which has two inflows from heat exchangers and three outflows to booster pumps. MIMO can also be used to model a tri-ethylene glycol in an offshore installation [129].
Related Patterns: We can apply MIMO in different situations. First, when inflows have identical media (mono-flux), MIMO can be treated as the similar way of MISO. Second, supposing that inflows have different media (multi-flux), outflows are normally predetermined according to inflows. In addition, SOURCE often works with MIMO pattern. SISO, SIMO, MISO, SOURCE, and SINK can be regarded as special cases of MIMO.

5.2.7 SERIES

Intent: Provides a general way to model series structures which function when all of their components work.

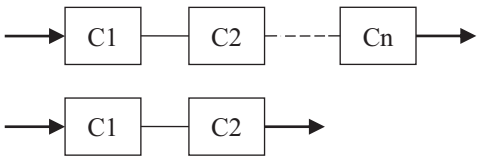



```
//We assume two upstream components, U1, U2, i.e. m=2; two downstream components, n=2
//Multi-flux
DO1 := if state == WORKING then min(DI1, Capacity1) else 0.0;
DO2 := if state == WORKING then min(DI2, Capacity2) else 0.0;
FO1 := min (FI1, DO1);
FO2 := min (FI2, DO2);

//Mono-flux
//Define new variables DO – Demand Output, FO – Flow Output
DO  := if state == WORKING then min(DI1 + DI2, Capacity) else 0.0;
DO1 := if U1.state == WORKING and U2.state == WORKING then DO*U1.Capacity/(U1.Capacity+U2.Capacity)
       else if U1.state == WORKING and U2.state != WORKING then DO
       else 0.0;
DO2 := if U1.state == WORKING and U2.state == WORKING then DO*U2.Capacity/(U1.Capacity+U2.Capacity)
       else if U2.state == WORKING and U1.state != WORKING then DO
       else 0.0;
FO := min(FI1, DO1) + min(FI2, DO2);
FO1 := if DI1 > 0 then FO*DI1/(DI1 + DI2) else 0.0;
FO2 := if DI2 > 0 then FO*DI2/(DI1 + DI2) else 0.0;
```

Figure 5.35: Running scheme considering capacity limit in MIMO.

Motivation: Consider the structure with several components connected in series, which is commonly found in production and safety systems.



The average unavailability of SERIES pattern $\bar{U}_{\text{SERIES}}$ is:

$$\bar{U}_{\text{SERIES}} = 1 - (1-\bar{u}_1)(1-\bar{u}_2)\cdots(1-\bar{u}_n) = 1 - \prod_{i=1}^{n}(1-\bar{u}_i) \tag{5.1}$$

where $\bar{u}_1, \bar{u}_2, \cdots, \bar{u}_n$ are average unavailabilities of components C1, C2, $\cdots$, Cn, respectively.
Structure: Dependency graph of the assertion of SERIES is shown in Figure 5.36.

- C1.varState, C2.varState: indicates the states of C1 and C2, respectively.
- C1.varOutFlow, C2.varOutFlow: define separately the outputs of C1 and C2. The output of C1 depends on its state and inflow. A similar situation occurs for C2.

- varInFlow, varOutFlow: defines inflow and outflow of the structure. The inflow of the SERIES structure is identical to the input of C1. The outflow of the SERIES structure equals to the output of C2.

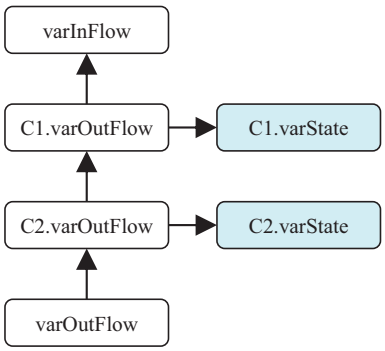


Figure 5.36: Dependency graph of the assertion of SERIES.

Implementation: Consider following matters when implementing this pattern:

- If downstream components have several outputs, they cannot be modeled using SERIES.
- Consider other behavioral patterns if necessary. It means that components in a series structure can be modeled with different behavioral patterns.

Sample Model: The AltaRica 3.0 code of SERIES is shown in Figure 5.37.

```
class SERIES
    Component C1, C2;//Behavioral patterns of components are predefined
    Real varInFlow, varOutFlow (reset = 0.0);
    assertion
        C1.varInFlow := varInFlow;
        C2.varInFlow := C1.varOutFlow;
        varOutFlow   := C2.varOutFlow;
end
```

Figure 5.37: The AltaRica 3.0 code of SERIES.

By considering capacity constraints, SERIES is crossed with demands and flows. The running scheme considering capacity limit in SERIES is shown in Figure 5.38.
Known Uses: In a gas system [68], the separator, compressor, and dehydration units work as a series subsystem. SERIES structures (x-mas tree, flow module, and manifold) can also be found in a subsea gas production system [6].
Related Patterns: SERIES is the series connection of several SISO patterns. NooN, a special case of KooN, is identical to SERIES.

5.2.8 PARALLEL

Intent: Provides a general way to model the parallel structure which functions when at least one component works.

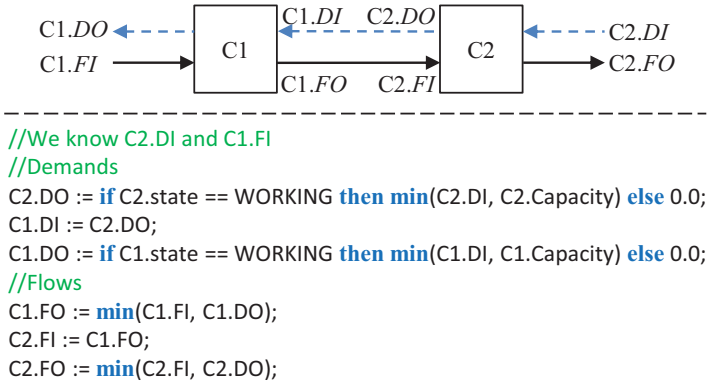
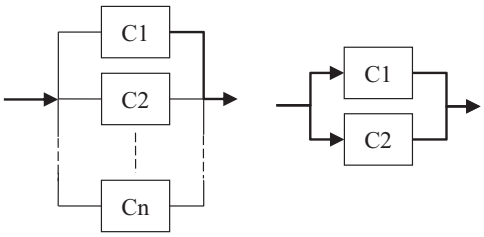


Figure 5.38: Running scheme considering capacity limit in SERIES.

Motivation: Consider a structure which connects components in parallel. The average unavailability of PARALLEL pattern $\bar{U}_{\text{PARALLEL}}$ is:

$$\bar{U}_{\text{PARALLEL}} = \bar{u}_1 \bar{u}_2 \cdots \bar{u}_n = \prod_{i=1}^{n} \bar{u}_i \tag{5.2}$$

where $\bar{u}_1, \bar{u}_2, \cdots, \bar{u}_n$ are average unavailabilities of components C1, C2, $\cdots$, Cn, respectively.



Structure: Dependency graph of the assertion of PARALLEL is shown in Figure 5.39.

- C1.varState, C2.varState: indicate the states of C1 and C2, respectively.
- C1.varOutFlow, C2.varOutFlow: define outputs of C1 and C2, respectively. The output of C1 depends on its component state and structure inflow. The way of modeling C2 is similar to that of C1.
- varInFlow, varOutFlow: define input and output of the structure. The outflow of a parallel subsystem relies on the outputs of its components.

Implementation: Parallel components may not be identical. They may deal with different percentage of structure flows.
Sample Model: The AltaRica 3.0 code of PARALLEL is shown in Figure 5.40.

By considering capacity constraints, PARALLEL is crossed with demands and flows. The running scheme considering capacity limit in PARALLEL is shown in Figure 5.41.
Known Uses: Parallel structures can be frequently found in engineered systems, such as turbo compressors and generators in offshore production systems [68, 36].
Related Patterns:

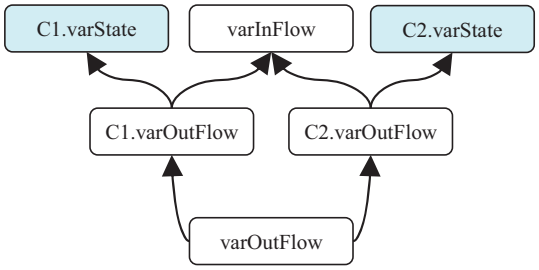
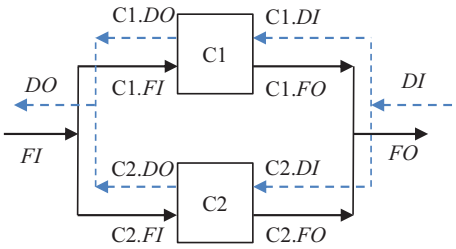


Figure 5.39: Dependency graph of the assertion of PARALLEL.

```
//Assume that each component is capable of processing the inflow of system
class PARALLEL
    Component C1, C2;//Behavioral patterns of components are predefined
    Real varInFlow, varOutFlow (reset = 0.0);
    assertion//Treat WORKING as the up state
     C1.varInFlow := if (C1.varState==WORKING and C2.varState==WORKING) then 0.5*varInFlow
                else if (C1.varState==WORKING and C2.varState !=WORKING) then varInFlow
                else 0.0;
     C2.varInFlow := if (C1.varState==WORKING and C2.varState==WORKING) then 0.5*varInFlow
                else if (C1.varState !=WORKING and C2.varState==WORKING) then varInFlow
                else 0.0;
     varOutFlow   := C1.varOutFlow + C2.varOutFlow;
end
```

Figure 5.40: The AltaRica 3.0 code of PARALLEL.



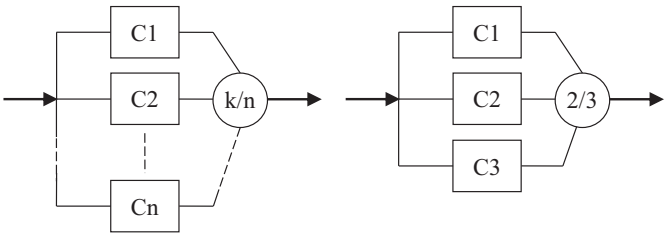

```
//Capacity-based distribution
C1.DI  := if C1.state == WORKING and C2.state == WORKING then DI*C1.Capacity/(C1.Capacity+C2.Capacity)
          else if C1.state == WORKING and C2.state != WORKING then DI
          else 0.0;
C1.DO := min(C1.DI, C1.Capacity);
C2.DI  := if C1.state == WORKING and C2.state == WORKING then DI*C2.Capacity/(C1.Capacity+C2.Capacity)
          else if C2.state == WORKING and C1.state != WORKING then DI
          else 0.0;
C2.DO := min(C2.DI, C2.Capacity);
DO     := C1.DO + C2.DO;
C1.FI  := if C1.DO > 0 then FI*C1.DO/(C1.DO + C2.DO) else 0.0;
C1.FO  := min(C1.FI, C1.DO);
C2.FI  := if C2.DO > 0 then FI*C2.DO/(C1.DO + C2.DO) else 0.0;
C2.FO  := min(C2.FI, C2.DO);
FO     := C1.FO + C2.FO;
```

Figure 5.41: Running scheme considering capacity limit in PARALLEL.

- PARALLEL pattern is parallel connection of several SISO patterns.
- The special case of KooN, 1ooN is identical to PARALLEL.
- If the working of only one component can meet the demand for running the hot standby subsystem, Main unit/Hot standby unit Coordination (MHC) is identical to PARALLEL.

5.2.9 KooN

Intent: Provides a general way to model the subsystem which works when at least k of the total n items must be functioning.
Also Known As: koon; k-out-of-n voting; koon: G ("good")
Motivation: Consider the structure which works when the minimum number of available components is required.



The average unavailability of KooN pattern $\bar{U}_{\text{KooN}}$ is:

$$\bar{U}_{\text{KooN}} = 1 - \sum_{x=k}^{n} \binom{n}{x} (1-\bar{u})^x \bar{u}^{n-x} \tag{5.3}$$

where components in KooN are usually identical, and $\bar{u}$ is the average unavailability of each component.
Structure: Dependency graph of the assertion of KooN is shown in Figure 5.42.

- C1.varState, C2.varState, and C3.varState: indicate states of C1, C2, and C3, respectively.
- varInFlow, varOutFlow: indicate the input and output of the structure, respectively. The structure outflow relies on component states, as well as the structure inflow.

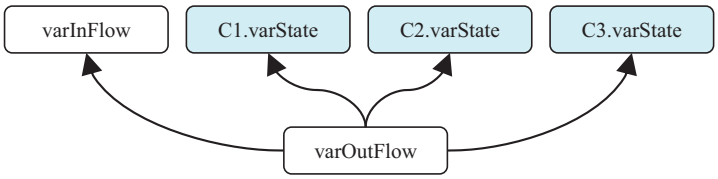


Figure 5.42: Dependency graph of the assertion of KooN.

Sample Model: The graphical representation of GTS for KooN is shown in Figure 5.43. The AltaRica 3.0 code of KooN is shown in Figure 5.44.
Known Uses: KooN can model sensors (2-out-of-3) of pressure protection systems [28, 29].
Related Patterns:

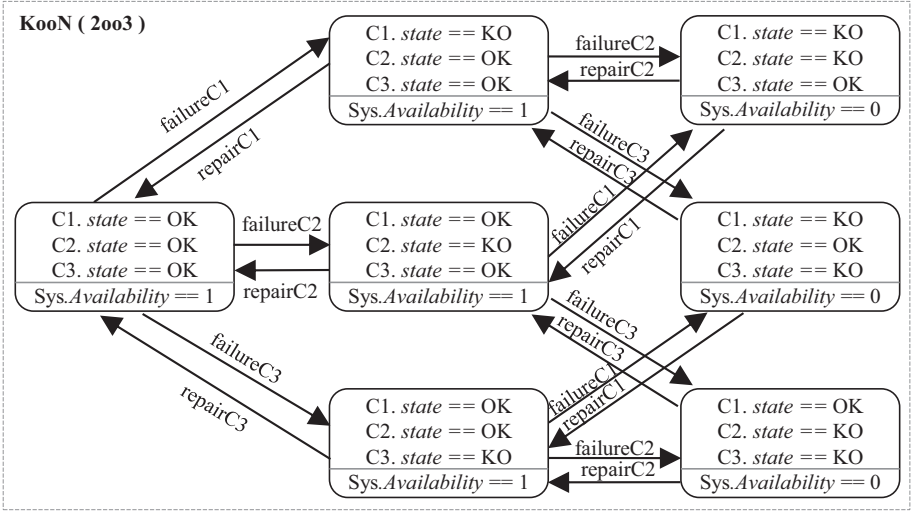


Figure 5.43: Graphical representation of GTS for KooN.

```
class KooN //k-out-of-n
    Component C1, C2, C3;//Behavioral patterns of components are predefined
    Real varInFlow  (reset = 0.0);
    Real varOutFlow (reset = 0.0);
    assertion//Treat WORKING as the up state
        varOutFlow := if C1.varState == WORKING and C2.varState == WORKING then varInFlow
                  else if C1.varState == WORKING and C3.varState == WORKING then varInFlow
                  else if C2.varState == WORKING and C3.varState == WORKING then varInFlow
                  else 0.0;
end
```

Figure 5.44: The AltaRica 3.0 code of KooN.

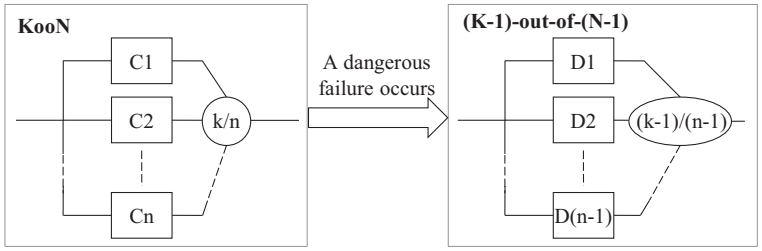
- The special case of KooN, 1ooN is identical to PARALLEL.
- Special case of KooN, NooN is identical to SERIES.
- KooN serves as the basis of SwitchKooN.

5.2.10 SwitchKooN

Intent: Provides a general way to model the structure which switches a KooN (K-out-of-N) structure into (K-1)-out-of-(N-1) structure when a dangerous failure occurs.
Also Known As: Changing logic
Motivation: Consider the structure when the KooN needs to be switched after a failure. Once the failure is repaired, the structure is restored to KooN structure.

This new configuration can increase system availability. If there is no switch, the structure is supposed to work as a K-out-of-(N-1) structure after a failure (similar like in degraded mode). According to Equation (5.3),

$$\bar{U}_{\text{(K-1)-out-of-(N-1)}} = 1 - \sum_{y=k-1}^{n-1} \binom{n-1}{y} (1-\bar{u})^y \bar{u}^{n-y-1} \tag{5.4}$$



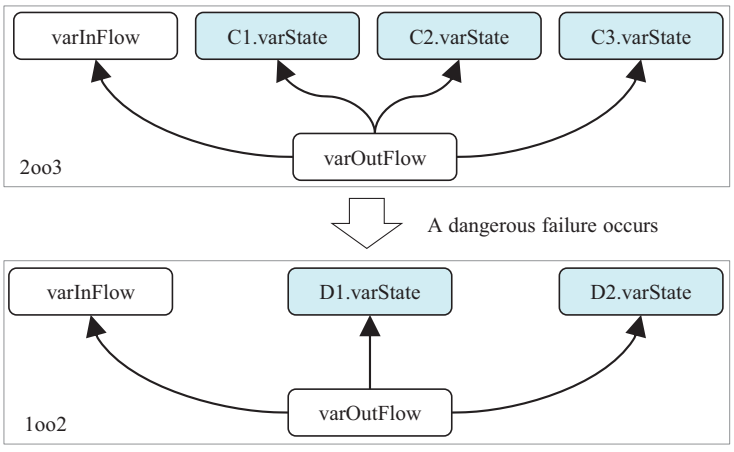
Note: Residual components are renamed as D1, D2, …, D(n-1).

$$\bar{U}_{\text{K-out-of-(N-1)}} = 1 - \sum_{y=k}^{n-1} \binom{n-1}{y} (1-\bar{u})^{y} \bar{u}^{n-y-1} \tag{5.5}$$

$$\bar{U}_{\text{(K-1)-out-of-(N-1)}} - \bar{U}_{\text{K-out-of-(N-1)}} = -\binom{n-1}{k-1} (1-\bar{u})^{k-1} \bar{u}^{n-k} \tag{5.6}$$

where $\bar{U}_{\text{(K-1)-out-of-(N-1)}}$ and $\bar{U}_{\text{K-out-of-(N-1)}}$ are the unavailabilities of (K-1)-out-of-(N-1) and K-out-of-(N-1) structures, respectively. Since $-\binom{n-1}{k-1}(1-\bar{u})^{k-1}\bar{u}^{n-k}$ is a negative number, thus $\bar{U}_{\text{(K-1)-out-of-(N-1)}} < \bar{U}_{\text{K-out-of-(N-1)}}$. That is, availability of the (K-1)-out-of-(N-1) structure is higher than that of K-out-of-(N-1) structure.

Structure: Dependency graph of the assertion of SwitchKooN is shown in Figure 5.45. Consider a 2oo3 structure as an example, if a dangerous failure occurs in 2oo3, the logic is reconfigured into 1oo2. In 2oo3, the system becomes unavailable when two components are failed. However, after switch, the system is still available if there are two failed components. Regarding 1oo2 structure, the system can work with only one working component.



Note: The residual two components are renamed as D1 and D2.

Figure 5.45: Dependency graph of the assertion of SwitchKooN.

Implementation: There are two matters to be considered when applying this pattern:

- 2oo3 structures can be commonly found in safety systems.
- Flow variable in the assertion of GTS can be updated using “if-then-else” or “switch-case” instructions.

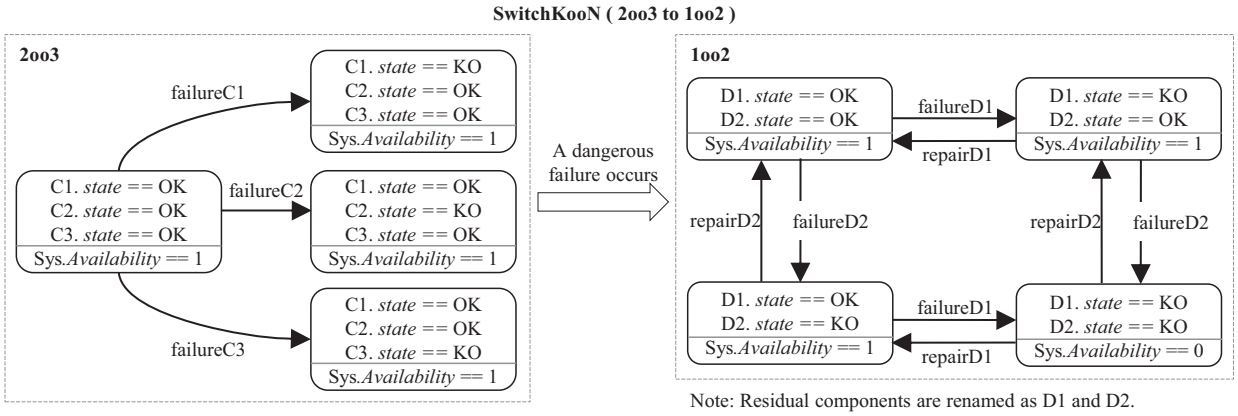


Figure 5.46: Graphical representation of GTS for SwitchKooN.

Sample Model: The graphical representation of GTS for SwitchKooN is shown in Figure 5.46. The AltaRica 3.0 code of SwitchKooN is illustrated in Figure 5.47.

```
class SwitchKooN
    Component C1, C2, C3;//Behavioral patterns of components are predefined
    assertion //Components from 2oo3 to 1oo2
        SystemAvailable := if (C1.varState==OK and C2.varState==OK and C3.varState==OK)
                           then ( 1 - ( 1 - C1.Available * C2.Available ) *
                                      ( 1 - C1.Available * C3.Available ) *
                                      ( 1 - C2.Available * C3.Available ) )
                           else ( 1 - ( 1 - C1.Available ) *
                                      ( 1 - C2.Available ) *
                                      ( 1 - C3.Available ) );
end
```

Figure 5.47: The AltaRica 3.0 code of SwitchKooN.

Known Uses: SwitchKooN can be used to model the sensors (i.e. from 2oo3 to 1oo2) in typical applications 3-6 and 5 in ISO/TR 12489 [60].
Related Patterns: SwitchKooN is based on KooN.

5.2.11 SequentialWork

Intent: Provides a general way to model multiple safety systems working in a sequential order.
Motivation: Consider the structure when its subsystems work sequentially, for example, a multiple (multi-layers) safety system comprises several subsystems operating one after the other when prior ones fail.
Structure: The graphical representation of GTS for SequentialWork is shown in Figure 5.48. The failure of former subsystem behaves as the demand for the working of following subsystem. The failure of previous subsystem $i-1$ triggers (with startDemand) subsequent subsystem i. This one is initially out of work ($subSystemState == 0$). If the trigger action (startDemand) from subsystem $i-1$ succeeds, subsystem i becomes working ($subSystemState == 1$). If subsystem i fails ($subSystemState == 2$), it can be used to trigger the working of subsystem $i+1$.
Implementation: There are two matters to be considered when using this pattern:

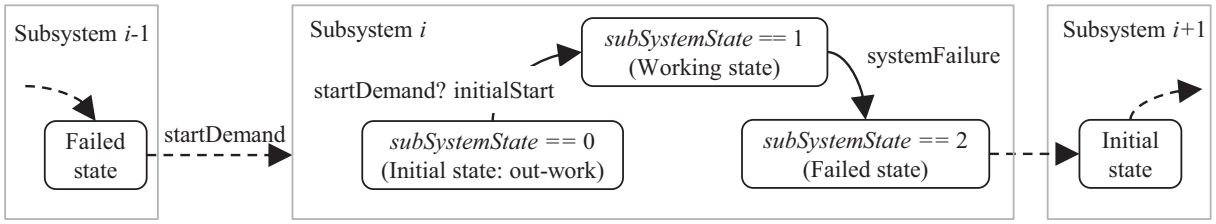


Figure 5.48: Graphical representation of GTS for SequentialWork.

- Define failure mechanisms of subsystems. subSystemState needs to be determined on the basis of component states.
- Assign probabilities of failure on demand of trigger events. initialStart may not be perfect. It can be triggered with a probability of failure on demand.

Sample Model: The AltaRica 3.0 code of SequentialWork is shown in Figure 5.49.

```
class subSystem
    Boolean startDemand ( reset = false );
    Integer subSystemState ( init = 0 ); //0:Initial state(out-work); 1:Working; 2:Failed.
    Boolean subSystemAvailable ( init = true ); //Availability of subsystem
    event initialStart (delay = 0); //Launch system
    event subSystemFailure (delay = 0); //Indicates the failure of subsystem
    transition
        initialStart:  subSystemState == 0 and startDemand == true
                    -> subSystemState := 1;
        systemFailure: subSystemState == 1 and subSystemAvailable == false
                    -> subSystemState := 2;
end

class SequentialWork
    subSystem S1 ( subSystemState.init = 1 ); //S1 works initially
    subSystem S2 ( subSystemState.init = 0 ); //S2 works after S1 fails
    subSystem S3 ( subSystemState.init = 0 ); //S3 works after S2 fails
    assertion
        S2.startDemand := ( S1.subSystemState == 2 );
        S3.startDemand := ( S2.subSystemState == 2 );
end
```

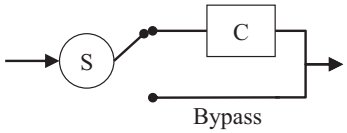
Figure 5.49: The AltaRica 3.0 code of SequentialWork.

Known Uses: SequentialWork can be applied to model a multiple safety system (i.e. typical application 4) in ISO/TR 12489 [60].

5.2.12 BYPASS

Intent: Provides a generic way to model the bypass structure which is activated when the main component fails. In this situation, the quality of the flow decreases (e.g. the gas is not purified or processed), but the quantity of the flow holds.

Motivation: A bypass is the temporary replacement or bypassing of equipment such as a reactor or heat exchanger with a length of pipe [83]. Consider a structure, its system unavailability decreases with a bypass line.



Structure: Dependency graph of the assertion of BYPASS is shown in Figure 5.50. varInFlow and varOutFlow indicate the input and output of the structure, respectively. The structure outflow relies on its inflow.

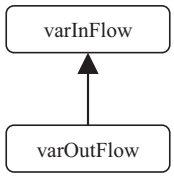


Figure 5.50: Dependency graph of the assertion of BYPASS.

Implementation: Consider following implementation issues when applying the pattern:

- *Different from cold standby.* The cold standby component maintains both flow quantity (except possible losses in adapt time) and quality. However, BYPASS gets decreased quality when compared with cold standby structure.
- *Outflow differs according to protected component.* If the protected (main) component is non-critical, the outflow equals to the inflow when using bypass line. In this situation, the bypass outflow is acceptable (or after treatment) for downstream components.
- We usually assume that the bypass line and the switch are perfect.

Sample Model: The AltaRica 3.0 code of BYPASS is shown in Figure 5.51.

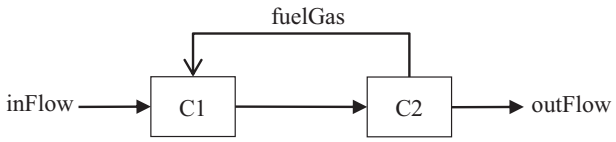
```
//Quality of the flow is reduced, but the quantity keeps
//Assume a perfect switch
class BYPASS
    Component C;//The behavioral pattern of component is predefined
    Real varInFlow (reset = 0.0);
    Real varOutFlow (reset = 0.0);
    assertion
        varOutFlow := varInFlow;
end
```

Figure 5.51: The AltaRica 3.0 code of BYPASS.

Known Uses: A dummy bypass can be found in a flow network in oil and gas industry [4]. A make-up compressor in a gas system [68] can also be modeled using BYPASS pattern.

5.2.13 LOOP

Intent: Provides a general way to model the loop structure whose components depend on each other.
Also Known As: Logical loops, circular logics



Motivation: Consider the loop structure with several components depending on themselves.
Structure: Dependency graph of the assertion of LOOP is shown in Figure 5.52. This dependency graph is a Directed Cyclic Graph (DCG). There is one circle inside, that is, "`C1.outFlow` → `fuelGas` → `C2.outFlow` → `C2.inFlow` → `C1.outFlow`".

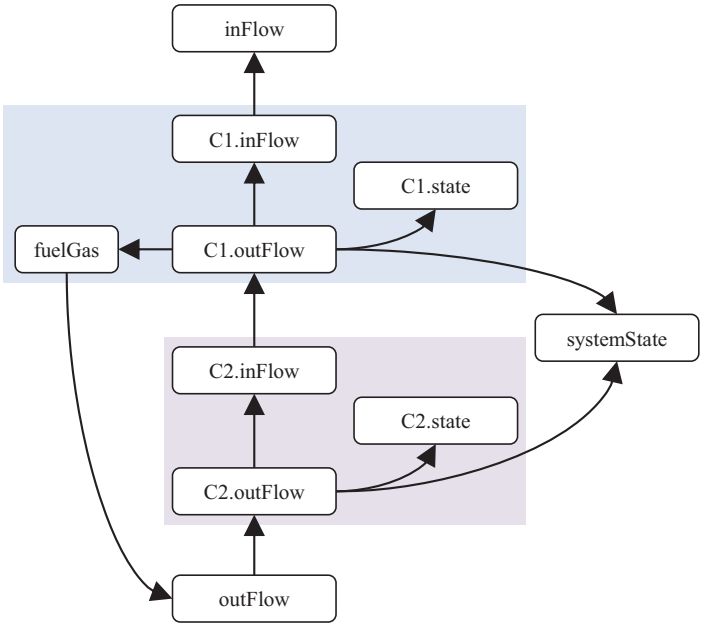


Figure 5.52: Dependency graph of the assertion of LOOP.

- systemState: indicates the system state. When the system is up, systemState (state variable) is TRUE, otherwise FALSE.
- C1.state, C1.inFlow, and C1.outFlow: indicate the state, input, and output of C1, respectively. The way to model C2 is similar to C1. The outflow of C1 depends on its state, inflow, systemState, and outflow of C2. The outflow of C2 relies on its state, systemState, and outflow of C1.
- inFlow and outFlow: indicate input and output of the structure, respectively. The structure outflow relies on systemState, fuelGas, and outflow of C2.

Implementation: Consider following issues when implementing LOOP pattern:

- The returned flow (e.g. fuelGas) is predetermined in a system.
- LOOP is required to initialize subsystem at the beginning. A running scheme is used for loop structures. Consider the two-component structure in Motivation part, its running scheme is shown in Figure 5.53.

- systemState becomes complicated once the targeted system is complex. We need to predefine the out-of-work state (the structure is unavailable) and working state (the structure works). It means that even if some (unimportant) components have failed, the structure is still available.

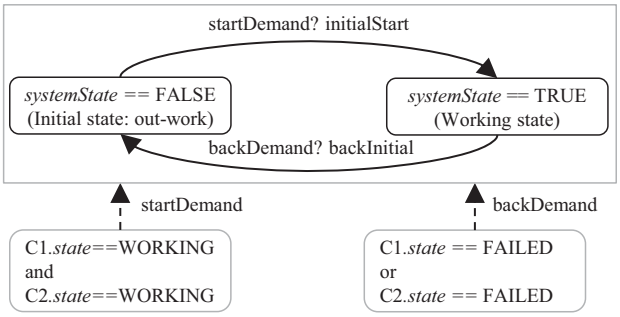


Figure 5.53: Running scheme of LOOP.

In Figure 5.53, a state variable systemState is defined. systemState==FALSE means that the system is at the initial stage and systemState==TRUE indicates that the system is out of the initial state. The scheme covers the following steps:

(1) When both C1 and C2 are in WORKING state, the startDemand becomes true. Associated with startDemand, the system is launched when systemState is in its initial state. Once transition initialStart occurs, systemState changes from FALSE to TRUE, and startDemand is set to false.

(2) Either C1 or C2 is out of WORKING state, backDemand becomes true. Together with backDemand, the system returns to the initial stage, that is, the transition backInitial occurs. In this situation, systemState transfers from TRUE to FALSE.

(3) If both components are repaired, that is, the startDemand is fulfilled, the process enters into step (1) again. Note that transitions initialStart and backInitial are perfect, that is probabilities of failure on demand are 0.

Sample Model: The AltaRica 3.0 code of LOOP is shown in Figure 5.54.

Known Uses: LOOP can be used in an offshore installation [129], where the gas flows from compressors to a dehydration unit. Subsequently, the gas from the dehydration unit returns to support the work of compressors. Similar structures can also be found in an oil and gas separation plant [90].

5.3 Composition Patterns

Composition Patterns (CP) represent cooperations in systems such as the main and standby units.

```
class LOOP
    Component C1, C2;//Behavioral patterns of components are predefined
    Boolean startDemand (reset = false);
    Boolean backDemand  (reset = false);
    Boolean systemState (init = false);
    parameter Real inFlow = 10.0;//Inflow of structure
    parameter Real fuelGas = 1.0;//Return inflow for C1, be treated as a condition
    Real outFlow (reset = 0.0);//Outflow of structure
    event initialStart  (delay = 0);//To start the structure
    event backToInitial (delay = 0);//The structure returns to initial phase
    transition
        initialStart:systemState == false and startDemand -> systemState := true;
        backToInitial:systemState == true and backDemand  -> systemState := false;
    assertion
        startDemand := C1.varState == WORKING and C2.varState == WORKING;
        backDemand  := C1.varState == FAILED  or  C2.varState == FAILED;
        C1.inFlow   := inFlow;
        C1.outFlow  := if systemState == true then C1.inFlow else 0.0;
        C2.inFlow   := C1.outFlow;
        C2.outFlow  := if systemState == true then C2.inFlow else 0.0;
        outFlow     := if systemState == true then (C2.outFlow - fuelGas) else 0.0;
end
```

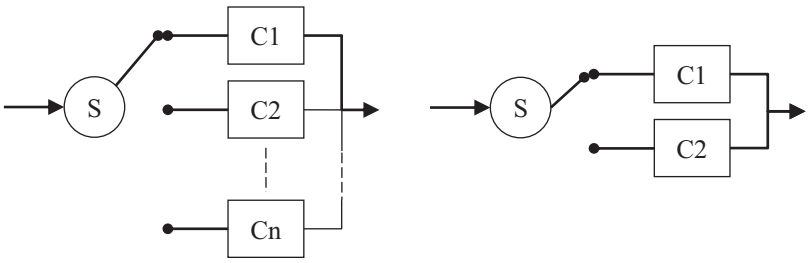
Figure 5.54: The AltaRica 3.0 code of LOOP.

5.3.1 Main unit/Cold standby unit Coordination (MCC)

Intent: Provides a general way to model components which are waiting for *passively* substituting failed components, thus to maintain higher system availability.
As Known As: Passive redundancy, cold redundancy, cold spare
Motivation: A standby redundant system requires an additional equipment, called a fault detector/switch, to identify the failure of operating components and transfer their functions to some other components that act as substitutes [71].

When an on-line unit becomes faulty, it is removed from the operation and replaced with a standby unit. The standby units can be in either a hot or a cold mode. A hot standby unit operates in synchrony with the (on-line) main unit, and is ready to take over at any time. A cold standby unit is unpowered until it is required to replace the faulty main unit. Hot standby systems are typically used when the *reconfiguration time* is required to be minimized, while cold-standby systems are usually used in applications where *energy consumption* is more crucial [75]. Hot standby is discussed in the following section.

Consider the cold standby unit waiting for passively replacing main units.



Structure: Graphical representation of GTS for cold standby unit is shown in Figure 5.55. The failure of the main unit, acting as a demand, triggers the cold standby unit. The initial state of

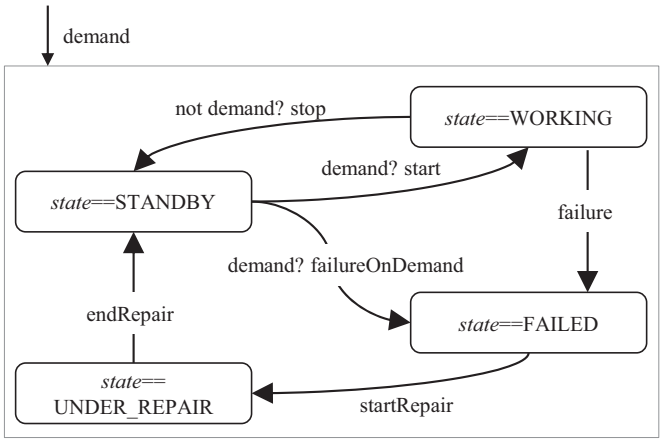


Figure 5.55: Graphical representation of GTS for cold standby unit.

the component is STANDBY. If there is a demand, the component becomes WORKING. If the switch is perfect after the demand, the component works. Otherwise, the component fails. The component returns to STANDBY state after repair.

Implementation: The fault detector/switch may be perfect. In this situation, event failureOnDemand can be omitted.

Sample Model: The AltaRica 3.0 code of Main unit/Cold standby unit Coordination (MCC) is shown in 5.56.

```
class MCC//Main unit/ ColdStandby unit coordination
    Component C (init = STANDBY);//Behavioral patterns of components are predefined
    parameter Real Lambda = 1.0e-3;
    parameter Real Mu = 1.0e-1;
    parameter Real Gamma = 1.0e-2;
    Boolean demand (reset = false);
    event start           (delay = 0, expectation = 1-Gamma);
    event failureOnDemand (delay = 0, expectation = Gamma);
    event stop            (delay = 0);
    event failure         (delay = exponential(Lambda));
    event startRepair     (delay = 0);
    event endRepair       (delay = exponential(Mu));
    transition
        start:           C.varState == STANDBY and demand    -> C.varState := WORKING;
        failureOnDemand: C.varState == STANDBY and demand    -> C.varState := FAILED;
        stop:            C.varState == WORKING and not demand-> C.varState := STANDBY;
        failure:         C.varState == WORKING               -> C.varState := FAILED;
        startRepair:     C.varState == FAILED                -> C.varState := UNDER_REPAIR;
        endRepair:       C.varState == UNDER_REPAIR          -> C.varState := STANDBY;
end
```

Figure 5.56: The AltaRica 3.0 code of Main unit/Cold standby unit Coordination (MCC).

Known Uses: MCC can be used to model two cold-standby treatment units in an oil production system [107].

Related Patterns: MCC can be obtained by adding cold standby-related transitions and states to PERFECT, NonRepairable, CorrectiveMaintenance, and DEGRADATION patterns.

5.3.2 Main unit/Hot standby unit Coordination (MHC)

Intent: Provides a generic way to model components which are waiting for *actively* substituting failed components, thus to assure higher system availability.
Also Known As: Active redundancy
Motivation: Similar as in PARALLEL.
Structure: The dependency graph of the assertion of MHC is similar to that of PARALLEL (see Figure 5.39 on page 60).
Implementation: MHC is similar to, but different from PARALLEL pattern. Parallel structure operates when at least one of its components works. However, the structure with hot standby behavior may require more components to work simultaneously.
Sample Model: The sample model of MHC is similar to the one in PARALLEL (see Figure 5.40).
Known Uses: MHC can be used to model several treatment units in an oil production system [107] and the pre-heaters in a FPSO system [85].
Related Patterns: If the system can work with only one working component, MHC is identical to PARALLEL.

5.3.3 Repairable unit/Repair crew Coordination (RRC)

Intent: Provides a general way to model the situation that the number of repair crew is limited.
Motivation: Consider the behavior when repair resources are limited.
Structure: Graphical representation of GTS for Repairable unit/Repair crew Coordination (RRC) is shown in Figure 5.57. The repairable unit can be modeled with the corresponding repairable pattern.

- RepairCrewWork: indicates the work state of repair crew. The working state of the repair crew (*RepairCrewWork*) is initially FALSE.
- numberBusyCrew: represents the number of busy repair crews.
- totalNumberCrew: stands for the total number of repair crews. If *numberBusyCrew* is smaller than *totalNumberCrew*, the repair is started. Simultaneously, one (1) is added to *numberBusyCrew*. Adversely, one (1) is decreased to *numberBusyCrew* when the repair is completed.

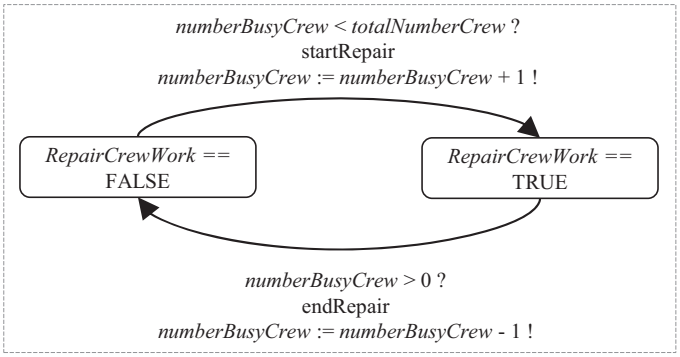


Figure 5.57: Graphical representation of GTS for repair crew.

Implementation: The startRepair of RRC needs to synchronize together with the startRepair of repairable components (see e.g. Figure 5.5 on page 40).
Sample Model: The AltaRica 3.0 code of Repairable unit/Repair crew Coordination (RRC) is shown in 5.58. Synchronization mechanisms are applied to compel several events simultaneously. In RRC, startRepair events of CorrectiveRepairCrew and component C are synchronized through event C_startRepair. endRepair events of CorrectiveRepairCrew and component C are synchronized through event C_endRepair.

```
class RRC//Repairable unit/ Repair crew coordination
    Integer numberBusyCrew (init = 0);//Number of busy repair crew
    parameter Integer totalNumberCrew = 1;//Total number of repair crew
    parameter Real MobilizationTime = 0.0;//Mobilization time for repair crew (default = 0)

    event startRepair (delay = MobilizationTime);
    event endRepair   (delay = 0);
    transition
      startRepair:numberBusyCrew < totalNumberCrew -> numberBusyCrew := numberBusyCrew + 1;
      endRepair:  numberBusyCrew > 0               -> numberBusyCrew := numberBusyCrew - 1;
end

block system
    CorrectiveMaintenance C;//Component C is modeled using CorrectiveMaintenance pattern
    RRC CorrectiveRepairCrew;
    //sychronization
    event C_startRepair (delay = 0);
    event C_endRepair   (delay = exponential(C.Mu));
    transition
        C_startRepair: !CorrectiveRepairCrew.startRepair & !C.startRepair;
        C_endRepair:   !CorrectiveRepairCrew.endRepair   & !C.endRepair;
end
```

Figure 5.58: The AltaRica 3.0 code of Repairable unit/Repair crew Coordination (RRC).

Known Uses: This pattern can be used in many production and safety systems. For instance, it is employed to model limited repair crews in typical applications 3-5 and 3-6 in ISO/TR 12489 [60].
Related Patterns: RRC can work with any repairable components when the number of repair crew is limited.

5.4 Relationships between Modeling Patterns

Modeling patterns are not independent. There are not only interactions inside each category, but also relationships between the three categories. Figure 5.59 depicts relationships between modeling patterns graphically. They are:

- NonRepairable pattern is obtained based on PERFECT pattern together with failure behaviors.
- CorrectiveMaintenance pattern can be gained by adding repair behaviors on NonRepairable pattern.
- PreventiveMaintenance can be obtained by adding preventive maintenance behaviors to CorrectiveMaintenance and DEGRADATION.

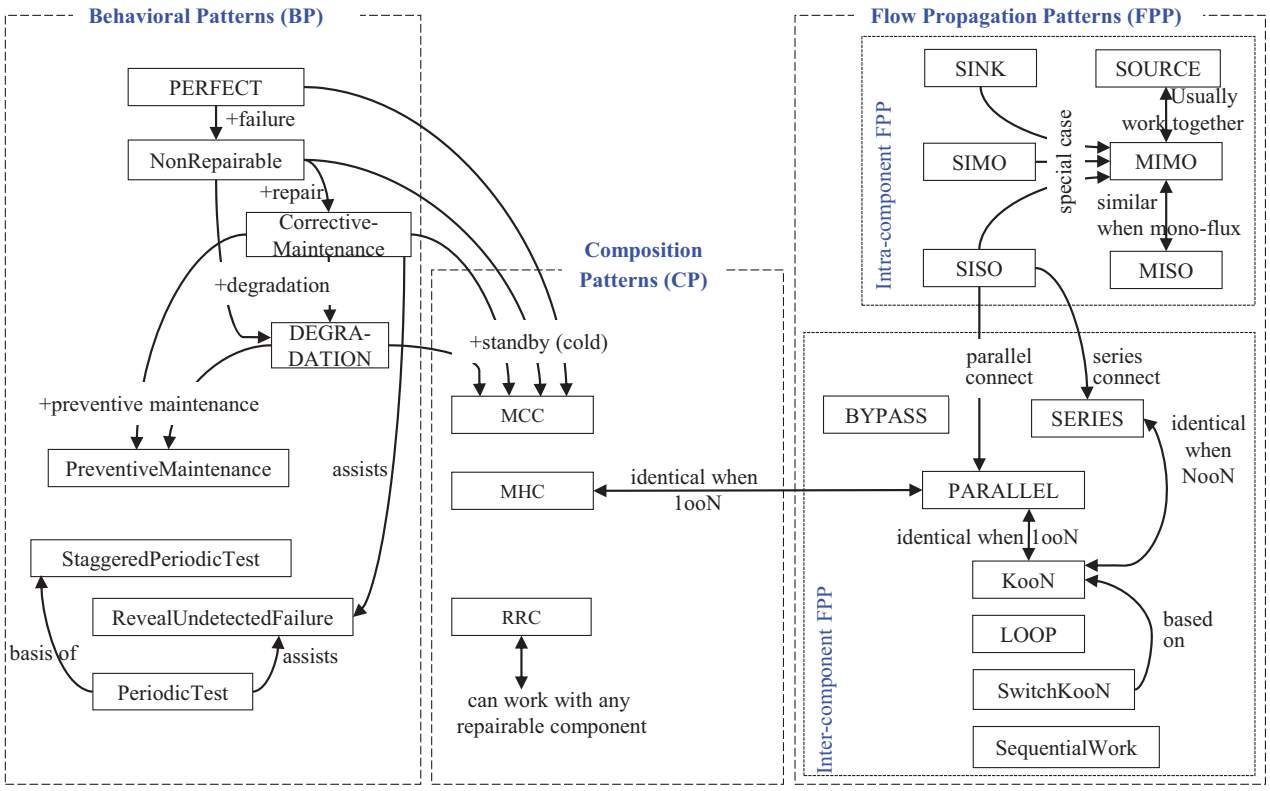


Figure 5.59: Modeling pattern relationships.

- DEGRADATION pattern is proposed with adding degradation behavior on either NonRepairable or CorrectiveMaintenance pattern.
- PeriodicTest is the basis of StaggeredPeriodicTest.
- PeriodicTest works with RevealUndetectedFailure to detect dangerous undetected failures.
- RevealUndetectedFailure is proposed based on PeriodicTest and CorrectiveMaintenance patterns.
- When inflows are mono-flux, MISO and MIMO can be modeled in a similar way.
- SOURCE usually collaborates with MIMO pattern.
- SISO, SIMO, MISO, SOURCE, and SINK patterns are special cases of MIMO.
- SERIES and PARALLEL are series and parallel connections of several SISO, respectively.
- When K =N, NooN is identical to SERIES.
- When $K = 1$, 1ooN and PARALLEL are identical.
- SwitchKooN is based on KooN.
- If the working of only one component can meet the demand for the work of the hot standby subsystem, PARALLEL is identical to Main unit/Hot standby unit Coordination (MHC).
- MCC can be obtained by adding cold standby-related transitions and states to PERFECT, NonRepairable, CorrectiveMaintenance, and DEGRADATION patterns.
- RRC can work with any repairable components when the number of repair crews is limited.

5.5 Modeling Patterns Reuse

After obtaining a catalog of modeling patterns, it is essential to figure out how to model a system using the catalog. In particular, we need to select appropriate modeling patterns in a modeling mission. In addition, we have to point out the methodology of reusing those modeling patterns. In this section, we state the issues and provide some ideas of what a methodology could be.

Two issues related to modeling patterns reuse have to be considered: First, how modeling patterns behave independently and cooperatively? That is, regarding a specific component or subsystem, we would like to know whether the proposed catalog can be used directly or not. Second, how we reuse modeling patterns step by step in a realistic system? That is, we want to figure out the methodology of how to classify modeling patterns, model a system, and obtain results.

We propose some ideas to address aforementioned questions. Firstly, modeling patterns can be used independently. That is, it is straightforward to apply them as in the catalog of modeling patterns. Adversely, in some situations, they have to work together to model a particular component or subsystem. It happens inside behavioral patterns themselves, as well as between behavioral and flow propagation patterns. For example, the repairable component with one inflow and one outflow can be commonly found in production and safety systems. We can apply SISO and CorrectiveMaintenance patterns to model such a component. The component outflow relies on its inflow and state. The inflow and outflow are assigned by SISO. The component state is provided by CorrectiveMaintenance.

Subsequently, we need to point out the way to reuse suitable modeling patterns. The methodology to model target systems via reusing modeling patterns could be composed of four steps:

(1) *Classification*: In this step, we identify units to be modeled and select corresponding modeling patterns. The target system is initially decomposed into components and subsystems. We select modeling patterns that are capable of modeling components and subsystems. In particular, we classify behavioral patterns at component level and flow propagation modeling patterns at component/subsystem level. We can use some features to determine suitable modeling patterns. Those features may be the cardinalities (sizes) of the inflow, outflow, and transition set of a component.

(2) *Pattern-based model*: Based on classification results, a model employing modeling patterns is established. Associated modeling patterns are assigned for each component and subsystem in a block diagram. The pattern-based model simplifies the task of establishing the AltaRica 3.0 model.

(3) *AltaRica 3.0 model*: On the basis of the pattern-based model, the corresponding AltaRica 3.0 model of the system can be built. Modeling patterns are firstly presented in AltaRica environment. Subsequently, the AltaRica 3.0 model of a system is constructed with identified modeling patterns.

(4) *Experimental results*: We acquire experimental results to quantitatively evaluate the benefit of using modeling patterns. We can obtain numerical results using AltaRica 3.0 tools. We compare results obtained using modeling patterns with those reported in the literature.

5.6 Summary

In this chapter, we present a catalog composed of twenty-four (24) modeling patterns for performance analysis of production and safety systems. We discuss each pattern with a set of structured items. The relationships between modeling patterns are presented. Some ideas of how to reuse modeling patterns are put forward. In order to test proposed modeling patterns, several typical production and safety systems are modeled using modeling patterns in the following chapter.

Chapter 6

Experimental Studies

In this chapter, we evaluate the applicability of proposed modeling patterns with a set of production and safety systems. Production systems are declared to cover most modeling difficulties for production-performance analysis. Safety systems are composed of a series of systems in ISO/TR 12489 [60], which encompass the majority of reliability issues of safety systems. In each experimental study, we provide the system description, identify involved modeling patterns, and compare experimental results obtained using modeling patterns and those reported in the literature. Note that AltaRica results are obtained using stochastic simulator of the AltaRica 3.0 language.

6.1 Production Systems in Process Industry

Leveraging the modeling patterns, we modeled several production systems from [68, 107, 129, 85]. The system in [68] is a concise production facility; the system in [107] is an oil production system; the system in [129] is an offshore installation; and the system in [85] is a FPSO (Floating Production Storage and Offloading) system.

6.1.1 A Production Facility

A production facility consisting of eight units is proposed in [68]. Figure 6.1 shows a flow diagram of this production facility. Gas separated from the well fluid at the upstream side is fed to the facility, treated through separators and dehydrators, and led to compressors (CMP-A and CMP-B). A make-up compressor (MUP) is installed to enable CMP-A and CMP-B to discharge gas with full flow rate even if some gas treatment units (HPS or DEH) are failed. It is assumed that MUP is equipped with dedicated gas treatment units.

The maximum throughput capacity for each unit is shown on blocks (in percentage form) in Figure 6.1. The maximum throughput capacity means that the unit has a potential to deal with the throughput volume, which does not mean that the unit is always operated at that condition. For simplicity, it is assumed that these units are stochastically independent with constant failure rates. It is further assumed that all units are independently repaired with constant repair rates.

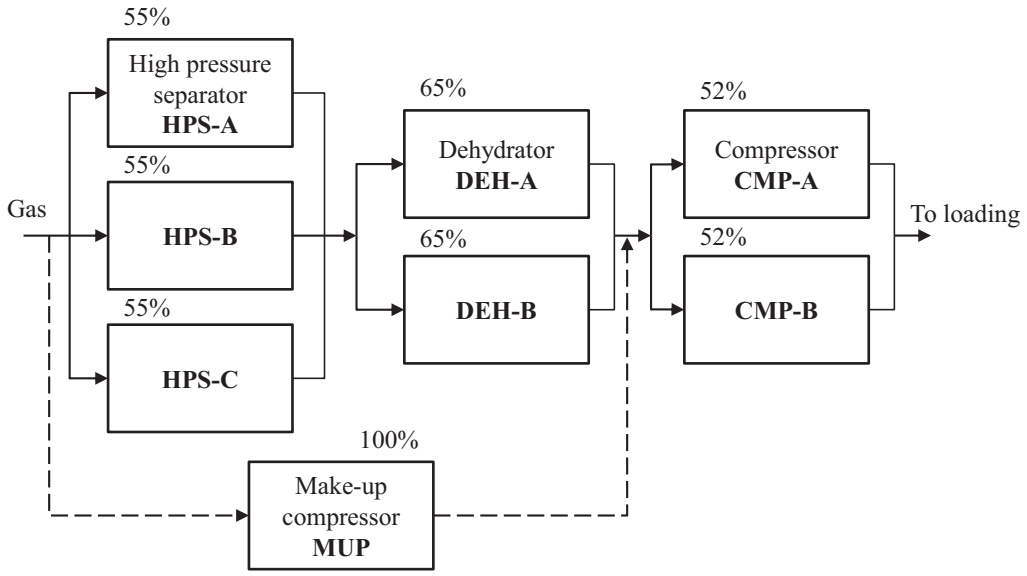


Figure 6.1: A production facility [68].

Modeling Patterns

We identify modeling patterns to model systems. We show how to construct this production system using modeling patterns (see Table 6.1). First, components and subsystems in this production facility are decomposed. Second, modeling patterns are assigned to decomposed units. For instance, High Pressure Separator A (HPS-A) can be modeled using SISO and CorrectiveMaintenance patterns. HPS (a three-component subsystem) can be constructed using PARALLEL pattern. {HPS, DEH} represents the SERIES connection of these two subsystems. The remaining units can be understood in an analogous way.

Table 6.1: Modeling patterns classification for the system in Figure 6.1.

Components/Subsystems	Modeling patterns
• HPS-A, HPS-B, HPS-C, DEH-A, DEH-B, CMP-A, CMP-B	SISO, CorrectiveMaintenance
• MUP	SISO, MCC
• HPS, DEH, CMP	PARALLEL
• {HPS, DEH}, {UPSTREAM, CMP}	SERIES

Experimental Results

After constructing the system with modeling patterns, we obtain numerical results, as shown in Table 6.2. Results attained using proposed modeling patterns agree well with those obtained by applying Markov approach and the UNIRAM software [68]. In our experiments, the mission time (length of histories) of stochastic simulator is 8.76×10^4 h (10 years); the number of Monte Carlo simulations (number of histories) of the system is 10^6. Note that the sum of each result column is 1.

Table 6.2: Probabilities comparison for the production facility.

System throughput capacity level (%)	Results in [68] (Markov Chains)	Results in [68] (UNIRAM software)	Our results
0	4.58E-5	5.48E-5	4.64E-5
52	1.34E-2	1.34E-2	1.34E-2
55	3.28E-6	0	2.12E-6
65	1.52E-5	2.74E-5	1.08E-5
100	9.86E-1	9.87E-1	9.87E-1

6.1.2 A Floating Production Storage and Offloading System

The FPSO (Floating Production Storage and Offloading) system is one of the most commercially viable systems in offshore production activities. The FPSO system in our case study includes a crude oil processing system, a single point mooring system, a crude oil storage and ballast system, a fire protection and lifesaving system, and a power and instrumental system [85]. We focus here on the production availability of the crude oil processing system.

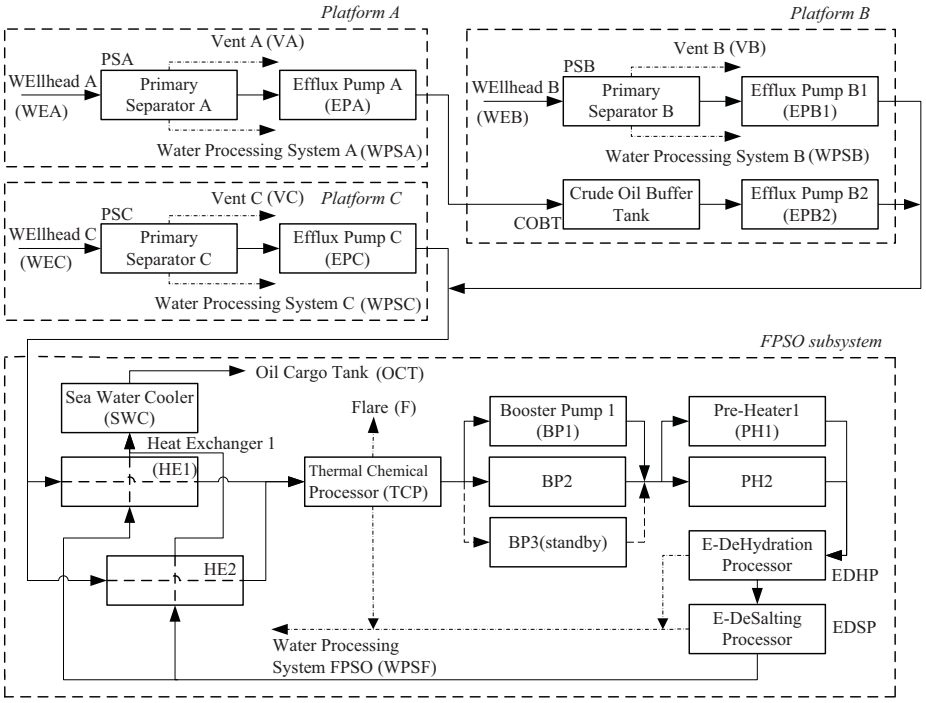


Figure 6.2: A Floating Production Storage and Offloading (FPSO) system [85].

As shown in Figure 6.2, the FPSO system consists of four sub-systems: platforms A, B, and C, and the FPSO subsystem. Platform A transfers the oil to a buffer tank in plaform B. Together with the output oil of platforms B and C, the overall oil is transported to heat exchangers on the FPSO subsystem. In Figure 6.2, the arrows with solid lines represent main streams of the system, that is, crude oil flows; the dashed arrows stand for the flows of separated gas and waste water. The required data (i.e. expected crude oil outputs, preventive maintenance intervals/durations, and failure/repair rates) is available in [85].

Modeling Patterns

Classification of modeling patterns for the FPSO system is given in Table 6.3. The FPSO system is firstly decomposed into components and subsystems. Take PSA as an example, it can be modeled using SISO, CorrectiveMaintenance, and PreventiveMaintenance patterns. With regard to {BP1, BP2, BP3} (a subsystem), it can be model with applying KooN pattern. In detail, this subsystem is a 2-out-of-3 structure. The rest of components/subsystems can be explained in a similar way.

Table 6.3: Modeling patterns classification in the FPSO system.

Components/Subsystems	Modeling patterns
• PSA, PSB, PSC, HE1, HE2	SISO, CorrectiveMaintenance PreventiveMaintenance
• EPA, EPB1, EPB2, EPC, COBT, PH1, PH2	SISO, CorrectiveMaintenance
• BP1, BP2, BP3	SISO, CorrectiveMaintenance PreventiveMaintenance, MCC
• EDSP	SIMO, CorrectiveMaintenance
• TCP	MIMO, CorrectiveMaintenance PreventiveMaintenance
• EDHP, SWC	MISO, CorrectiveMaintenance
• {PSA, EPA, COBT, EPB2}, {PSB, EPB1}, {PSC, EPC}	SERIES
• {HE1, HE2}, {PH1, PH2}	PARALLEL
• {BP1, BP2, BP3}	KooN

The FPSO system can be represented utilizing modeling patterns, as shown in Figure 6.3.

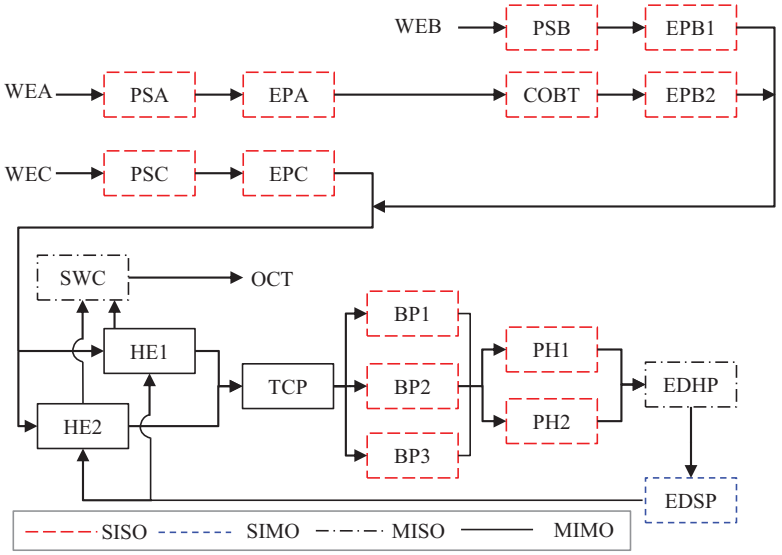


Figure 6.3: Modeling patterns-based presentation of the FPSO system in Figure 6.2.

Experimental Results

The production availability of the FPSO system is mainly affected by failure rates and repair rates of the system. Failure rates and repair rates used in experiments are multiplied with practical values in [85] and a coefficient in {0.5,1.0,1.5,2.0}. In our experiments, we take 8.76×10^4h (10 years) as the mission time and 2×10^4 as simulation histories (Monte Carlo simulations) [85].

Similar results are obtained via AltaRica models and corresponding Stochastic Petri Nets (SPN) model using GRIF software. Figures 6.4 and 6.5 show production availabilities of the FPSO system under different failure and repair rates. Results show that the production availability of the FPSO system decreases as the failure rates arise. Simultaneously, the production availability of the FPSO system rises as the repair rates increase. The reference point (second point on blue line with circle) of our test is the time when both the failure rate and repair rate are multiplied by 1.0.

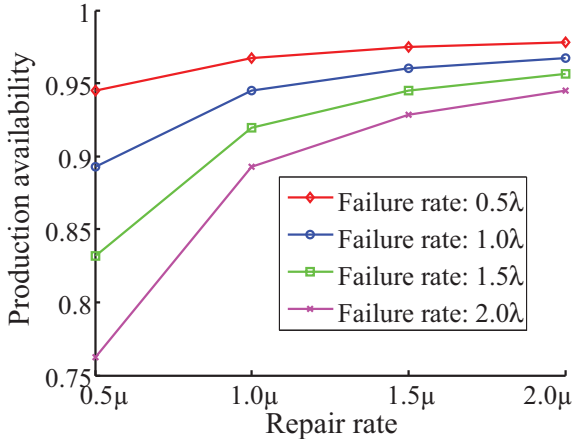


Figure 6.4: Production availabilities (AltaRica).

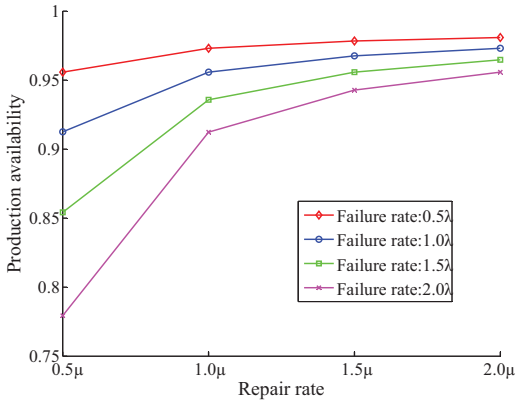


Figure 6.5: Production availabilities (SPN).

Table 6.4 shows that the result (at reference point) of production availability obtained with AltaRica is comparable with that reported in [85] using SPN. Nevertheless, SPN are difficult to master when the system under study becomes complex. The AltaRica 3.0 language is powerful in terms of modeling flexibility and understandability, which is convenient to update and revise.

Table 6.4: Comparison of production availabilities of the FPSO system.

Cases	Production availability
SPN model [85]	0.9636
AltaRica model	0.9445

6.1.3 An Oil Production System

An oil production system proposed in [107] is shown in Figure 6.6. This system represents a part of an oil extraction installation. This test case has been designed to concentrate most modeling difficulties of the assessment of production availability [107]. Details of this system are listed as follows:

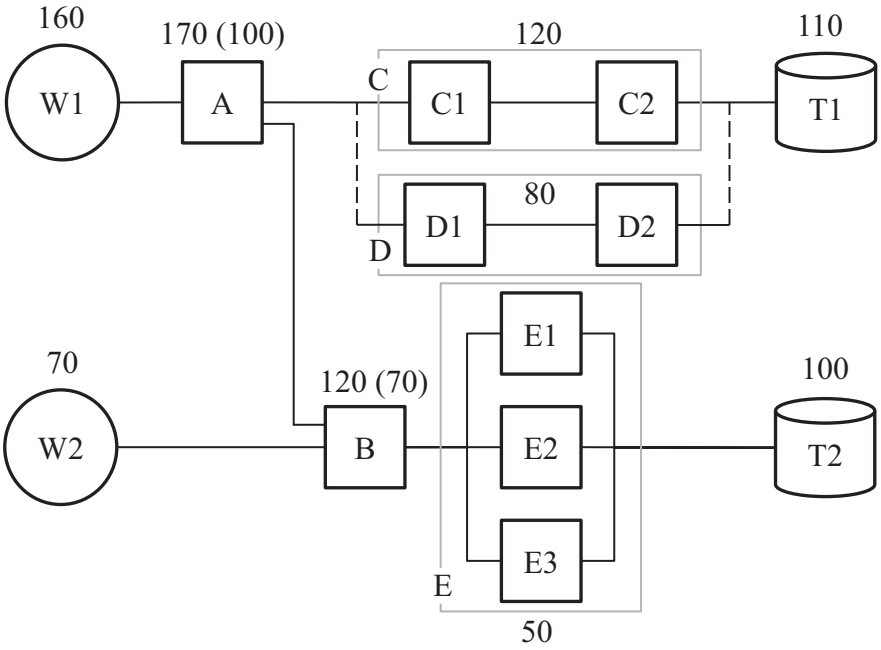


Figure 6.6: An oil production system [107].

- W1 and W2 are wells. Their production capacities are, respectively, 160 and 70 (barrels/day). When they are failed, the production is stopped.
- T1 and T2 are tanks. They are assumed to be perfectly reliable. Their storage capacities are, respectively, 110 and 100 (barrels/day).
- A, B are two treatment units. They have two failure modes: a failure that decreases their capacities from 170 to 100 (resp. 120 to 70), and a severe failure that stops the treatment. A severe failure may occur when the unit is either working correctly or in a degraded mode.
- Components C_i, D_i (i = 1, 2) and E_j (j = 1, 2, 3) are treatment units. Their capacities are, respectively, 120, 80 and 50. For all of these units, a failure stops the treatment. Components of bloc E are in hot redundancy.
- Line D is in cold redundancy with line C. As soon as line C is repaired, line D is stopped. If C1 fails, then C2 is stopped and vice-versa.
- Two repair crews are available in this system, that is, at most two components can be repaired simultaneously. A component is not able to treat the production during a repair.
- The production entering component A is split into two: a fraction goes to tank T1 through the top line and the remainder goes to tank T2 through the bottom line. This requires defining a splitting policy. We adopt the following one. The production of W1 goes preferably to the top line. The production of W2 goes to the bottom line in preference (priority). If needed, what remains available from W1 goes to the bottom line.

The problem is to assess the average production of two wells and the average storage in two tanks, that is, the mathematical expectation of these quantities throughout the mission time.

Modeling Patterns

Modeling patterns classification for the oil production system is given in Table 6.5. Take W1 as an example, it can be modeled with SOURCE and CorrectiveMaintenance patterns. That is, the applied flow propagation modeling pattern for W1 is SOURCE. The utilized behavioral pattern for W1 is CorrectiveMaintenance. {C1, C2} subsystem can be modeled with SERIES pattern. Remaining components/subsystems can be interpreted in a similar way.

Table 6.5: Modeling patterns classification in the oil production system.

Components/Subsystems	Modeling patterns
• W1, W2	SOURCE, CorrectiveMaintenance
• A	SIMO, CorrectiveMaintenance, DEGRADATION
• B	MISO, CorrectiveMaintenance, DEGRADATION
• C1, C2	SISO, CorrectiveMaintenance
• D1, D2	SISO, CorrectiveMaintenance, MCC
• E1, E2, E3	SISO, CorrectiveMaintenance, MHC
• T1,T2	SINK, PERFECT
• {C1, C2}, {D1, D2}	SERIES
• {E1, E2, E3}	PARALLEL

Experimental Results

After modeling the system with the AltaRica 3.0 language, we obtain experimental results, as shown in Table 6.6. Results obtained using modeling patterns meet well with those given in [107]. Note that the sum of the productions from W1 and W2 equals to the storages in T1 and T2. The Matrix Exponential Method is employed to obtain results in [107]. The mission time is 8.76×10^3 h (1 year). The number of Monte Carlo simulations of the system is 10^5.

Table 6.6: Results comparison for the oil production system.

Quantities	Results in [107]	Our results
Production of W1	83.6248	82.8309
Production of W2	47.6409	47.7638
Storage in T1	70.0677	69.3603
Storage in T2	61.198	61.2344

6.1.4 An Offshore Installation

We consider an offshore installation system (so-called SAFERELNET) proposed in [111, 129]. The system encompasses majority of modeling difficulties encountered when conducting studies related to production processes.

Figure 6.7 shows the flow diagram of the SAFERELNET system. SEP (Separation unit) divides the flow from the well (WEL) into three types of flows: gas, oil and water. The separated gas firstly flows to two parallel Turbo-Compressors (TC1 and TC2), and then to TEG (Tri-Ethylene

Glycol) for dehydration. Before the final gas exports, there are three gas flows which are employed for fueling and lifting operations. One part of the fuel gas returns to support the work of TCs, while the other part of fuel gas flows assists the electricity generation in TGs (Turbo-Generators). The obtained electricity is utilized by units TEG, EC (Electro-Compressor), OPS (Oil Pumping System), and WPS (Water Pumping System). Finally, the lift gas flow goes back to the well for assisting well production. The isolated `oil` flow is exported through OPS. To maintain the well pressure, the separated `water` is rejected into the well through WPS.

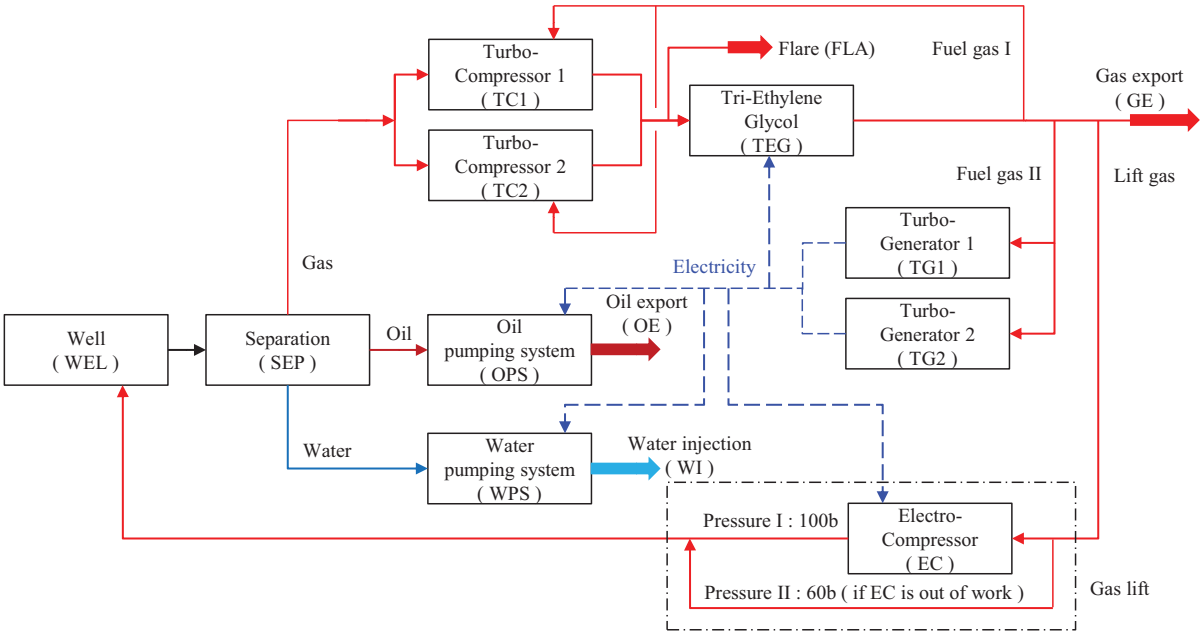


Figure 6.7: An offshore installation, redraw from [111, 129].

Three types of preventive maintenance are considered for TCs and TGs, where preventive intervals are every 2.16×10^3 h, 8.76×10^3 h, and 4.38×10^4 h [129]. Moreover, corresponding preventive maintenance durations are different. The required data for analyzing the production performance of SAFERELNET system can be found in [129].

Modeling Patterns

Classification of modeling patterns for the SAFERELNET system is provided in Table 6.7. Take WEL as an example, it can be modeled using SOURCE and PERFECT patterns. In simplicity, WEL can be constructed only with SOURCE pattern. Since the inflow (gas) and outflow (electricity) are different materials, both TG1 and TG2 (origin of electricity) are modeled with SOURCE pattern. Gas is treated as a condition for their working. The remaining components can be interpreted in a similar way. Regarding {TC1, TC2} subsystem, it can be model with PARALLEL pattern. The same situation comes to subsystem {TG1, TG2}.

The SAFERELNET system can be represented using identified modeling patterns, as shown in Figure 6.8.

Table 6.7: Modeling patterns classification for the system in Figure 6.7.

Components/Subsystems	Modeling patterns
• WEL	SOURCE, PERFECT
• TG1, TG2	SOURCE, CorrectiveMaintenance DEGRADATION, PreventiveMaintenance
• OPS, WPS	SISO, PERFECT
• TC1, TC2	SISO, CorrectiveMaintenance, DEGRADATION PreventiveMaintenance
• EC	SISO, CorrectiveMaintenance PreventiveMaintenance, MCC
• TEG	SIMO, CorrectiveMaintenance
• {TC1, TC2}, {TG1, TG2}	PARALLEL

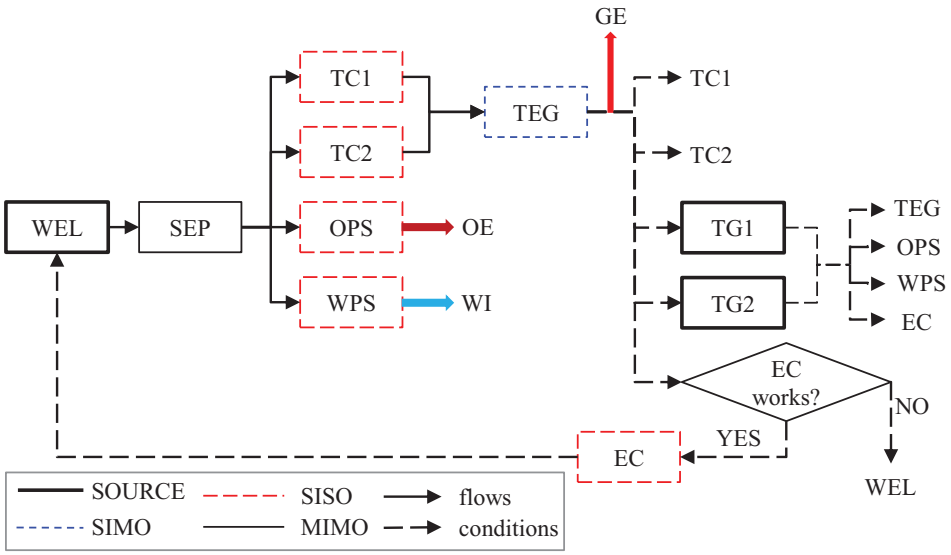


Figure 6.8: Modeling patterns-based presentation of the system in Figure 6.7.

Experimental Results

The AltaRica model of SAFERELNET system is established using proposed modeling patterns. We conduct experiments with three cases. Case A is the system without preventive maintenance. Case B refers to the system as perfect without failure, but the system is operated with preventive maintenance. Case C reflects the real situation in which the system runs with both corrective and preventive maintenances. In experiments, simulation histories (Monte Carlo simulations) are 10^5; the mission time in cases A, B, C are 10^3h, 10^4h, 5×10^5h, respectively [129].

Table 6.8 shows the comparison of production availabilities of SAFERELNET system with results in [129]. Results obtained using modeling patterns and those reported in [129] are similar. The results reveal that the production availabilities in cases A and B are higher than the ones in case C. The outcome is rational, and shares the same tendency as results in [129]. When system components are perfect without failure (no downtime caused by the corrective maintenance), the production availability becomes higher. Thus the production availability in case A is higher

than the ones in case C. Given that components in the system run without preventive maintenance (no downtime generated by the preventive maintenance), the production availability becomes higher. Hence the production availabilities in case B is higher than the ones in case C.

Table 6.8: Production availabilities of the SAFERELNET system.

Cases	Production availability	Gas	Oil	Water
A	Results in [129]	0.9726	0.9974	0.9577
	Our results	0.9570	0.9848	0.9564
B	Results in [129]	0.9763	0.9970	0.9730
	Our results	0.9711	0.9898	0.9758
C	Results in [129]	0.8858	0.9588	0.9573
	Our results	0.8860	0.9577	0.8875

6.2 Safety Systems in Process Industry

We choose safety systems in ISO/TR 12489 as our running examples because these architectures are general enough to cover most safety systems [60]. In addition, these systems are representative of most reliability studies of safety systems performed in petroleum, petrochemical, and natural gas industries, as well as in other industries [60].

Three assumptions are made for all systems in ISO/TR 12489:

- Detected and undetected dangerous failures of a given component are independent, with exception of systems ♯3-2 and ♯3-3.
- Failure rates are constant.
- Components are as good as new after repairs.

Table 6.9 shows the modeling patterns used for modeling SIS in ISO/TR 12489. We take systems ♯1-1 and ♯5 as an example to illustrate the results.

Two components are modeled in system ♯1-1, where the protected system is shut down during periodic tests and repairs. Therefore activities related to the maintenance/repair are negligible when calculating the system unavailability. Since the system unavailability of system ♯1-1 is only generated by DU failures, the logical solver (which only has DD failures) has not been considered. Both the pressure sensor and isolation valve are modeled by RevealUndetectedFailure pattern. These two components work in series, thus SERIES pattern is used as well.

In system ♯5, we employ a 2oo3 structure (S1a, S1b, and S1c) as an example to elaborate the results. Since DD and DU failures of the component are assumed to be independent in EDP system, the DD failure of a component (e.g. S1a) can be modeled using the CorrectiveMaintenance pattern. Since the uncovered DU failure cannot be repaired, it is constructed with the NonRepairable pattern. The covered DU failure by periodic tests is considered to be modeled with the RevealUndetectedFailure pattern. The subsystem composed by these three components, {S1a, S1b, S1c}, can be modeled by both KooN pattern and SwitchKooN pattern. Two groups of 2oo3 structures, {S1, S2}, in the EDP system can be modeled with PARALLEL pattern. Note that S1 and S2 stand for 2oo3 subsystems. The rest of classification results can be interpreted in the similar way.

Table 6.9: Modeling patterns classification for the safety systems in ISO/TR 12489.

System	Components/Subsystems	Modeling patterns
♯1-1	S, V	RevealUndetectedFailure
	{S, V}	SERIES
♯1-2	S, V	RevealUndetectedFailure
	{S, V}	SERIES
♯1-3	S	CorrectiveMaintenance,RevealUndetectedFailure
	LS	CorrectiveMaintenance
	V	RevealUndetectedFailure
	{S, LS, V}	SERIES
♯1-4	S, V	NonRepairable, RevealUndetectedFailure
	{S, V}	SERIES
♯2-1	S1, S2, V1, V2	RevealUndetectedFailure
	{S1, V1},{S2, V2}	SERIES
	{{S1, V1},{S2, V2}}	PARALLEL
♯2-2	S1, S2, V1, V2	RevealUndetectedFailure
	{S1, V1},{S2, V2}	SERIES
	{{S1, V1},{S2, V2}}	PARALLEL
♯2-3	S1, S2	CorrectiveMaintenance,RevealUndetectedFailure
	LS1,LS2	CorrectiveMaintenance
	V1, V2	RevealUndetectedFailure
	{S1, LS1, V1},{S2, LS2, V2}	SERIES
	{{S1, LS1, V1},{S2, LS2, V2}}	PARALLEL
♯2-4	S1, V1	RevealUndetectedFailure
	S2, V2	StaggeredPeriodicTest
	{S1, V1},{S2, V2}	SERIES
	{{S1, V1},{S2, V2}}	PARALLEL
♯3-1	S1, S2, S3, SV1, V1, SV2, V2	RevealUndetectedFailure
	{S1, S2, S3}	KooN
	{SV1, V1},{SV2, V2}	SERIES
	{{SV1, V1},{SV2, V2}}	PARALLEL
♯3-2	S1, S2, S3, SV1, V1, SV2, V2	RevealUndetectedFailure, CorrectiveMaintenance
	LS	CorrectiveMaintenance
	{S1, S2, S3}	KooN
	{SV1, V1},{SV2, V2}	SERIES
	{{SV1, V1},{SV2, V2}}	PARALLEL
♯3-3	S1, S2, S3, SV1, V1, SV2, V2	RevealUndetectedFailure, CorrectiveMaintenance
	LS	CorrectiveMaintenance
	{S1, S2, S3}	KooN,SwitchKooN
	{SV1, V1},{SV2, V2}	SERIES
	{{SV1, V1},{SV2, V2}}	PARALLEL
♯4	S1, S2, S3	CorrectiveMaintenance,RevealUndetectedFailure
	LS1,LS2	CorrectiveMaintenance
	SV1, SV2, SV3, V1, V2	RevealUndetectedFailure
	{SV1, LS1, V1}	SERIES
	All components	CorrectiveMaintenance
♯5	S1a, S1b, S1c, S2a, S2b, S2c	CorrectiveMaintenance, NonRepairable, RevealUndetectedFailure
	{S1a, S1b, S1c},{S2a, S2b, S2c}	KooN, SwitchKooN
	{S1, S2}	PARALLEL
	LS, SV1, V1, SV2, V2	NonRepairable, RevealUndetectedFailure
	{SV1, V1},{SV2, V2}	SERIES

Safety systems in ISO/TR 12489 are modeled using proposed modeling patterns. The mission time of all SIS is 8.76×10^4 h (10 years) except the one of system ♯5 which is 1.314×10^5 h (15 years). The number of Monte Carlo simulations of all systems is 10^6.

The results are listed in Tables 6.10 to 6.16. Note that the unavailability in tables refers to the average unavailability for simplicity. The failure frequency in tables is the average dangerous failure frequency. The Formulae, Fault tree, Markovian, and Petri net approaches are used in ISO/TR 12489. In general, the more complex the systems become, fewer approaches can be utilized. For example, because of the state explosion problem, the Markovian approach is only used in systems ♯1-1, ♯1-2, and ♯2-1.

6.2.1 An Overpressure Protection System with Single Channel

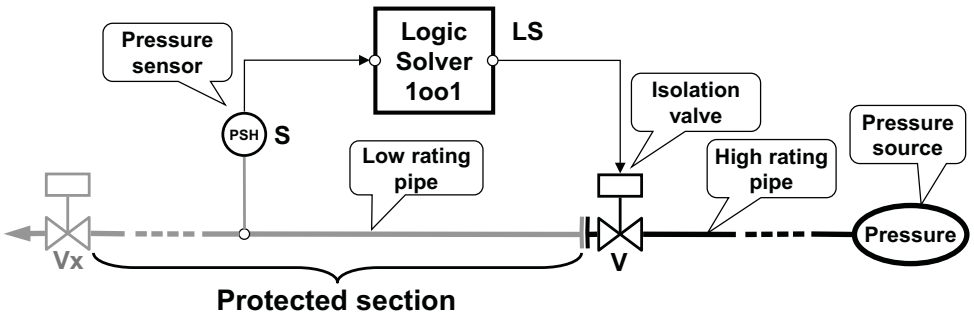


Figure 6.9: An overpressure protection system with single channel (System ♯1) [60].

The simplest architecture of a SIS is illustrated in Figure 6.9, which is composed of a pressure sensor (S), a logic solver (LS), and an isolation valve (V). This system is popular in process industry for common safety loops with low to moderate reliability requirements (Safety Integrity Level: SIL1 to SIL2). When the pressure exceeds the predefined threshold, the sensor sends a signal to the logic solver, which in turn commands the valve to close. According to different assumptions, there are four SIS generated from the system in Figure 6.9. They are numbered from ♯1-1 to ♯1-4.

The assumptions used for system ♯1-1 are:

- Perfect periodic tests are performed simultaneously.
- Installation (protected section) is stopped during periodic tests and repair.

The assumptions applied for system ♯1-2 are identical to system ♯1-1 except that:

- The periodic tests of components are not performed with the same interval.
- In addition, two kinds of periodic tests are performed on the isolation valve:
  - Partial stroking tests to check if the valve is able to move or not;
  - Full stroking tests to check if the valve is tight after closure.

The assumptions assigned for system ♯1-3 are exactly the same as ♯1-1 except that:

- The installation is not shut down during the repair of the sensor and of the logic solver.

- The sensor is periodically tested off line and is no longer available for its safety function during the periodic test.

Assumptions made for system ♯1-4 are just the same as for system ♯1-1, except that coverages of the periodic tests are not 100%. This means that part of the Dangerous Undetected (DU) failure is not covered by periodic tests, and thus will not be detected.

Experimental Results

Experimental results for system ♯1 are tabulated in Table 6.10. In general, results from AltaRica 3.0 models agree well with those reported in ISO/TR 12489.

Table 6.10: Experimental results in system ♯1.

Systems	Approaches	Unavailability	Failure frequency(h^{-1})
♯1-1	Formulae [60]	1.4E-2	3.2E-6
	Fault tree [60]	1.39E-2	3.16E-6
	Markovian [60]	1.39E-2	3.16E-6
	Petri net [60]	1.38E-2	3.15E-6
	AltaRica 3.0	1.39E-2	3.15E-6
♯1-2	Formulae [60]	1.06E-2	–
	Fault tree [60]	1.05E-2	3.17E-6
	Markovian [60]	1.05E-2	3.17E-6
	Petri net [60]	1.05E-2	3.17E-6
	AltaRica 3.0	1.05E-2	3.18E-6
♯1-3	Formulae [60]	1.47E-2	–
	Fault tree [60]	1.46E-2	1.39E-4
	Petri net [60]	1.46E-2	1.442E-4
	AltaRica 3.0	1.45E-2	1.35E-4
♯1-4	Formulae [60]	2.09E-2	–
	Fault tree [60]	2.08E-2	3.158E-6
	Petri net [60]	2.07E-2	3.13E-6
	AltaRica 3.0	2.07E-2	3.13E-6

With regard to system ♯1-1, AltaRica 3.0 results meet well with those given in ISO/TR 12489. Among these four systems, system ♯1-1 serves as a reference for systems ♯1-2, ♯1-3, and ♯1-4.

The average unavailability of system ♯1-2 is lower than the one of system ♯1-1 because two kinds of periodic tests are conducted on the isolation valve in system ♯1-2: the full stroking and partial stoking. The partial stoking is regarded as the main form of periodic test, and the corresponding periodic test interval decreases from 8760h to 4380h. Because DU failures may put a SIS in a down state for a long period until a periodic test is conducted, DU failures are always main contributors to the unavailability of a SIS [79]. Thus DU failures in system ♯1-2 can be discovered timely and less unavailability will be generated.

The average unavailability of system ♯1-3 is slightly higher than the one of system ♯1-1 due to the EUC is not stopped during the repair of the sensor and the logic solver. Thus DD failures of the sensor and the logic solver are considered. In addition, the sensor is periodically tested off

line, therefore the periodic test duration is taken into account. DD failures, repair time, as well as the periodic test duration bring higher, albeit not significant, unavailability of SIS.

The average unavailability of system ♯1-4 is higher than the one of system ♯1-1 because imperfect periodic tests are taken into account. Uncovered DU failures will not be detected and repaired.

6.2.2 An Overpressure Protection System with Dual Channel

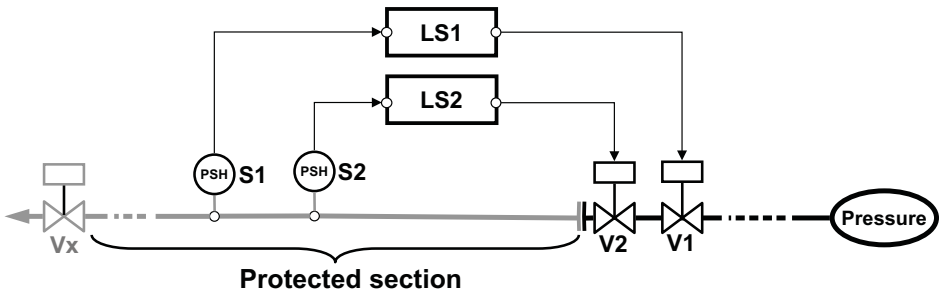


Figure 6.10: An overpressure protection system with dual channels (System ♯2) [60].

Figure 6.10 illustrates an overpressure protection system with dual channels. It is a structure with two channels (S1, LS1, and V1; S2, LS2, and V2) working in parallel. Such dual-channel architecture is commonly used for conforming to high reliability requirements (SIL2 to SIL4). According to various assumptions, four systems (♯2-1, ♯2-2, ♯2-3, and ♯2-4) are generated from the system in Figure 6.10.

Assumptions used for system ♯2-1 are exactly the same as ♯1-1 except that failure rates have been split into an independent part and a common cause failure part (Beta-factor model is applied).

Comparing with system ♯2-1, additional assumptions are added to system ♯2-2. It is the same way as systems ♯1-1 and ♯1-2.

Extra assumptions are assigned to system ♯2-3 when compared with system ♯2-1. It is the same way as systems ♯1-1 and ♯1-3.

Assumptions made for system ♯2-4 are identical to those for ♯2-1, except that:

- Periodic tests of sensors and valves are staggered.
  - The pressure sensor S2 is periodically tested in the middle of the periodic test interval of S1.
  - The isolation valve V2 is periodically tested in the middle of the periodic test interval of V1.
- Each periodic test of a component provides an opportunity to detect related potential common cause failures.

Experimental Results

Results comparison of system ♯2 can be found in Table 6.11. There is a good agreement for results obtained using modeling patterns and those reported in ISO/TR 12489. Since system ♯2 is a dual-channel SIS, system unavailabilities of ♯2 are lower than those of system ♯1.

Table 6.11: Experimental results in system ♯2.

Systems	Approaches	Unavailability	Failure frequency(h^{-1})
♯2-1	Formulae [60]	9.37E-4	–
	Fault tree [60]	9.33E-4	2.39E-7
	Markovian [60]	9.20E-4	2.34E-7
	Petri net [60]	9.30E-4	2.41E-7
	AltaRica 3.0	9.29E-4	2.37E-7
♯2-2	Formulae [60]	6.26E-4	–
	Fault tree [60]	6.45E-4	–
	Petri net [60]	6.46E-4	–
	AltaRica 3.0	6.45E-4	2.19E-7
♯2-3	Fault tree [60]	1.19E-3	–
	Petri net [60]	1.19E-3	–
	AltaRica 3.0	1.18E-3	1.05E-4
♯2-4	Fault tree [60]	4.9E-4	–
	Petri net [60]	4.94E-4	–
	AltaRica 3.0	4.94E-4	2.36E-7

According to the system description and assumptions, relationships between systems ♯2-1, ♯2-2, and ♯2-3 are similar to those between systems ♯1-1, ♯1-2, and ♯1-3. The situation of system ♯2-4 is different.

The staggered test is applied in system ♯2-4. Redundant components (e.g. S1 and S2, V1 and V2) in system ♯2-4 are tested with the same periodic test interval but not simultaneously. This policy decreases the risk that redundant components are unavailable at the same time in test. Thus the average unavailability of system ♯2-4 is significantly lower than that of system ♯2-1.

6.2.3 An Overpressure Protection System with Redundant Architecture

Figure 6.11 depicts an overpressure protection system with redundant architecture. The system is a popular architecture of SIS, which is composed of three parts in series: three pressure sensors (S1, S2, and S3) in a 2oo3 configuration, one logic solver (LS) and two channels of final elements (the solenoid valve SV1 and isolation valve V1; SV2 and V2). Such an architecture is used in process industry for safety loops with conforming to high reliability requirements (SIL3 to SIL4). Based on different assumptions, three systems (♯3-1, ♯3-2, and ♯3-3) are generated from the system in Figure 6.11.

Assumptions used for system ♯3-1 are identical to those for system ♯1-1. Assumptions assigned for system ♯3-2 are:

- Perfect periodic tests are performed simultaneously.
- The production is not shut down during the repairs of a sensor, the logic solver and solenoid valves.
- This system may be used as a subsea High Integrity Pressure Protection System (HIPPS), which is difficult to be maintained. A maintenance rig (carrying the repair crew) is required to be mobilized for repair operations and the production is not shut down while

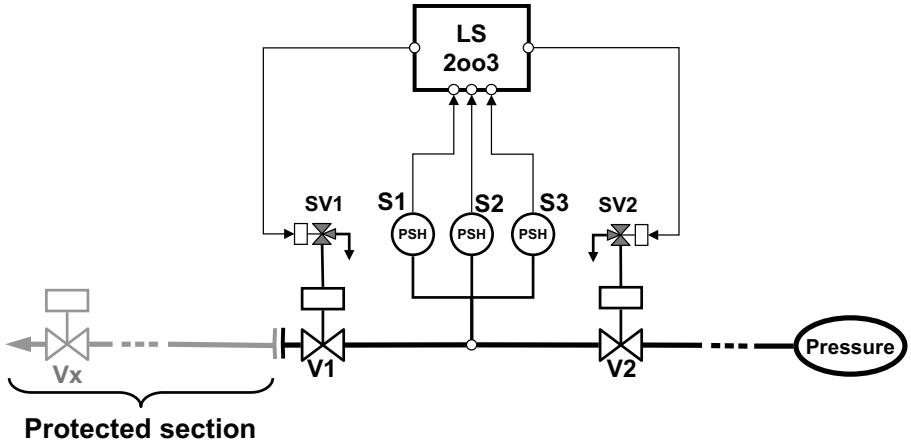


Figure 6.11: An overpressure protection system with redundant architecture (System ♯3) [60].

waiting for the maintenance rig.

- Sensors and solenoid valves are periodically tested off line and are no longer available for their safety function during periodic tests.
- The production is stopped during the maintenance of isolation valves.

Assumptions considered for system ♯3-3 are the same as those made for ♯3-2, except that when a dangerous failure of one sensor is detected, remaining sensors are reorganized from 2oo3 to 1oo2. This extra assumption makes system ♯3-3 more available than ♯3-2 when a dangerous failure is detected.

Experimental Results

Experimental results of system ♯3-1 are listed in Table 6.12. Average unavailabilities obtained using modeling patterns and those provided in ISO/TR 12489 give very similar results.

Table 6.12: Experimental results in system ♯3-1.

Approaches	Unavailability
Formulae [60]	9.0E-4
Fault tree [60]	9.0E-4
Petri net [60]	9.0E-4
AltaRica 3.0	9.1E-4

Experimental results of system ♯3-2 are tabulated in Table 6.13. Average unavailabilities obtained using AltaRica models are in good agreement with those in ISO/TR 12489.

Experimental results of system ♯3-3 are listed in Table 6.14. The average unavailability obtained using the AltaRica model agrees well with that in ISO/TR 12489. Note that in system ♯3-3, the mobilization time of the maintenance rig is 720h and the periodic test duration is 2h. Because of applying the SwitchKooN pattern (see section 5.2.10 on page 62), the average unavailability of system ♯3-3 is obviously lower than the one of ♯3-2.

Table 6.13: Unavailability results comparison (system ♯3-2).

Mobilization time (h)	Periodic test duration (h)	Petri net [60]	AltaRica 3.0
0	0	9.76E-4	9.78E-4
0	2	1.14E-3	1.17E-3
720	0	3.95E-3	3.93E-3
720	2	4.09E-3	4.14E-3

Table 6.14: Experimental results in system ♯3-3.

Approaches	Unavailability
Petri net [60]	2.70E-3
AltaRica 3.0	2.74e-3

6.2.4 A Multiple Safety System

Figure 6.12 represents a multiple safety system. It is comprised of two subsystems working in a predetermined sequence. The first one (S1, LS1, SV1, and V1) can be a safety loop of the BPCS (Basic Process Control System) and the second one (S2, S3, LS2, SV2, SV3, V1, and V2) can be a safety loop of the ESD (Emergency Shut Down system) or a HIPPS.

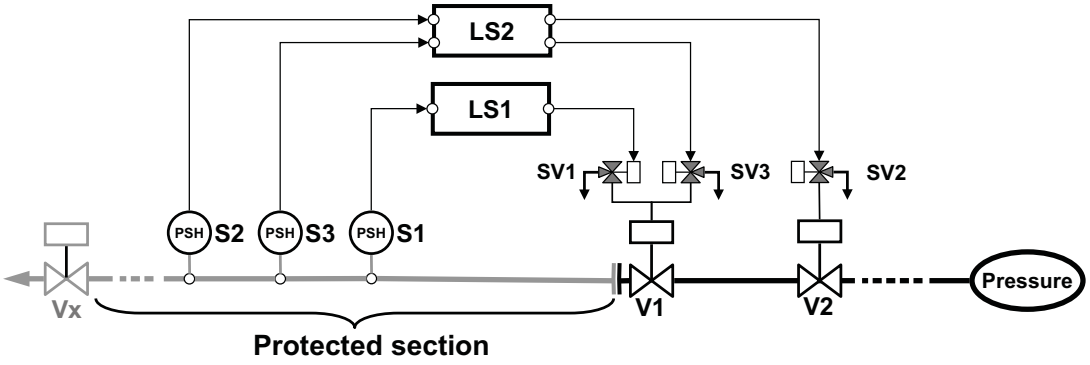


Figure 6.12: A multiple safety system (System ♯4) [60].

Assumptions used for system ♯4 are:

- Perfect periodic tests are performed simultaneously.
- Installation is stopped during repair of valves.
- Periodic test durations are negligible.

Experimental Results

Table 6.15 shows that results obtained using AltaRica agree rather well with those in ISO/TR 12489.

Table 6.15: Experimental results in system ♯4.

Approaches	Unavailability	Failure frequency(h^{-1})
Fault tree [60]	2.96E-4	4.02E-7
Petri net [60]	2.95E-4	4.0E-7
AltaRica 3.0	2.95E-4	3.98E-7

6.2.5 An Emergency Depressurization System of A Hydrocracking Unit

Figure 6.13 represents an Emergency DePressurization (EDP) system of a hydrocracking unit, which comes from the downstream oil and gas industry. This system is composed of two groups of temperature sensors (S1a, S1b, and S1c; S2a, S2b, and S2c) organized in 2oo3, one logic solver (LS) and two isolation valves (V1 and V2) organized in parallel and piloted by two solenoid valves (SV1 and SV2). This safety system aims to quickly depressurize the reactor when the temperature increases and reaches a predetermined threshold, thus to avoid a runaway of the exothermic chemical reaction.

Assumptions used for system ♯5 are:

- Periodic tests are performed when the reactor is stopped.
- Installation is paused during repair of Dangerous Undetected (DU) failures.
- Installation is shut down during periodic tests and repair of the logic solver.
- Failures that are not covered by periodic tests will not be detected and repaired.
- The 2oo3 logic of a group of sensors is switched to 1oo2 in case of one Dangerous Detected (DD) failure in the group.

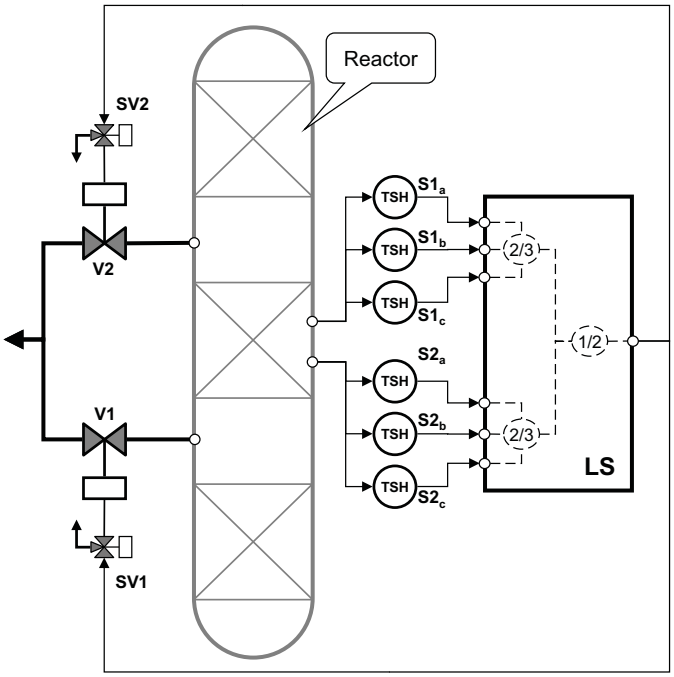


Figure 6.13: An emergency depressurization system of a hydrocracking unit (System ♯5) [60].

Experimental Results

Experimental results comparison of system ♯5 can be found in Table 6.16. The AltaRica 3.0 and ISO/TR 12489 (Fault tree) give almost the same results.

Table 6.16: Experimental results in system ♯5.

Approaches	Unavailability
Fault tree [60]	3.50E-4
AltaRica 3.0	3.46E-4

6.3 Summary

In this chapter, we conduct experimental studies on a set of production and safety systems. With regard to each system, the system description, the involved modeling patterns, and the experimental results are elaborated. Experimental results demonstrate that proposed modeling patterns are applicable to model production and safety systems in process industry, especially in oil and gas industry. Results comparisons show that experimental results obtained using modeling patterns are in good agreement with those from the literature.

Chapter 7

Conclusion and Future Works

In this PhD project, modeling patterns have been discussed in several aspects, that is, the form to capitalize modeling patterns (catalog), the way to develop modeling patterns (methodology), and experimental studies to test modeling patterns.

7.1 Conclusion

We proposed a pattern-based approach for performance analysis of production and safety systems. Modeling patterns are capable of stabilizing the experience from modeling processes. By reusing modeling patterns, a modeling mission can be simplified when analyzing performance of systems. Three main works have been completed in this thesis:

- *We studied modeling patterns in different facets.* We classified modeling patterns according to their purpose, which reflects the function of a specific modeling pattern. Modeling patterns were categorized into behavioral, flow propagation, and composition patterns. Each type of modeling pattern was illustrated using Guarded Transition Systems (GTS). A methodology to develop modeling patterns was proposed.
- *We showed modeling patterns as a catalog.* Based on reviewing numerous production and safety systems, we proposed twenty-four (24) modeling patterns. They are composed of eight (8) Behavioral Patterns (BP), thirteen (13) Flow Propagation Patterns (FPP), and three (3) Composition Patterns (CP). Each pattern was illustrated with a set of structured items (e.g. structure and sample model). Structures of BP and CP are shown with graphical representations of GTS. Structures of FPP are shown with their dependency graphs. Sample models are given with AltaRica 3.0 code. Relationships between modeling patterns were illustrated as well. We also discussed the methodology to reuse modeling patterns.
- *We conducted experimental studies on a set of production and safety systems.* We extracted these systems from the literature. Production systems are declared to cover most of modeling difficulties for production-performance analysis. Safety systems are composed of a group of systems in ISO/TR 12489, which encompass the majority of reliability issues. With regard to each system, we illustrated its system description, involved modeling patterns, as well as experimental results. Experimental results obtained using the modeling

patterns are in good agreement with those reported in the literature. It is shown that proposed modeling patterns are applicable to model production and safety systems in process industry.

7.2 Future Works

Three issues related to the topic of this thesis can be investigated in future works:

- *Modeling patterns can be improved.* Raised modeling patterns are based on a limited set of production and safety systems. Even though they are declared to cover most reliability studies [107, 129, 60], these patterns can be improved and expanded with new system behaviors. Moreover, further research may consider EUC (Equipment Under Control) as an integral part of safety systems. New modeling patterns are also expected with such integrations.
- The proposed pattern-based approach has proven to be applicable to a finite set of production and safety systems from the literature. However, *more experiments are required* to further validate the solidity of the proposed approach.
- The pattern-based approach can simplify modeling missions in a modular way. This approach can be *extended to other performance engineering domains*, such as performance testing and requirements engineering.

Appendix A

Production Availability Analysis using Stochastic Petri Nets

The offshore oil industry holds characteristics of high risk, investment and profit. Increasing attention has been paid to ensure that the production system possesses the capacity to perform its required functions as expected. Hence the performance of offshore production systems is important for stakeholders from both industry and public domains. In this chapter, we show our work of production performance analysis using stochastic Petri nets[1].

We perform the production availability analysis of a FPSO (Floating Production Storage and Offloading) system. The FPSO system is employed for processing and provisionally storing the crude oil from offshore platforms or subsea wells. The oil inside the FPSO cargo tank is transferred to shuttle tankers periodically. We select stochastic Petri nets (SPN) as the modeling tool in this study. In summary, highlights of this work are:

- SPN model of a FPSO system is proposed.
- Production availability of the FPSO system is obtained.
- Different preventive maintenance policies are considered.

[1] H. Meng, L. Kloul, A. Rauzy. Production availability modeling of FPSO system using stochastic Petri nets. In L. Podofillini, B. Sudret, B. Stojadinovic, E. Zio, and W. Kröger, editors, Proceedings of the 25th European Safety and Reliability Conference (**ESREL 2015**), Zurich, Switzerland, Sept. 2015: 2271-2279.

Production Availability Modeling of FPSO System using Stochastic Petri Nets

Huixing Meng
Computer Science Laboratory (LIX), École Polytechnique, Palaiseau, France

Leïla Kloul
Computer Science Laboratory (PRiSM), Université de Versailles, Versailles, France

Antoine Rauzy
Chair Blériot-Fabre, CentraleSupélec/ SAFRAN, Chatenay-Malabry, France
Industrial Engineering Laboratory (LGI), CentraleSupélec, Chatenay-Malabry, France

ABSTRACT: The offshore oil industry holds the characteristics of high risk, investment and profit. Increasing attention has been paid to ensure that the production system possesses the capacity to perform its required functions as expected. Hence the performance and the reliability of the offshore production systems are important for the stakeholders from both the industry and public domains. The objective of our study is to perform the production availability analysis of the FPSO (Floating Production Storage and Offloading) system. The FPSO system is employed for processing and provisionally storing the crude oil from the offshore production platforms or the subsea oil wells directly. The oil inside the FPSO cargo tank is transferred to the shuttle tankers periodically. We selected stochastic Petri nets (SPN) as the modeling tool in this study because they are naturally suitable for the performance evaluation and availability analysis.

1 INTRODUCTION

The life cycle of the offshore oil activities covers the offshore exploration, drilling, production, as well as the decommission processes. Among them, the offshore production period occupies the longer time, which could last from several years to decades, whereas the offshore drilling normally takes several months. The FPSO (Floating Production Storage and Offloading) is now a popular scheme for the offshore production (Lake & Arnold 2007). The FPSO could work in both shallow and deep waters, as well as in rich and poor reserves oil fields. The FPSO can be shifted to another offshore field conveniently and economically after its previous commission.

The FPSOs are widely employed all over the world, but mainly operated in the North Sea, both sides of South Atlantic Ocean, Australia and East Asia waters (Tillie Nutter 2014). FPSOs are complex systems, the operations of them may generate lots of possible uncertainties and failures, as well as reliability and risk difficulties. Lots of incidents and accidents of the FPSOs have been reported (HSE 2003). Recently, an explosion accident happening on the FPSO *Cidade de São Mateus* left 9 fatalities and several injured, which was occurred on February 11, 2015, about 120 km away from the coast of Brazil (BW Offshore 2015).

Long outages are particularly important for the process plants (Mannan 2005). The aforementioned incidents or accidents can lead to the decrease of production availability. In practice, the stakeholders, especially the decision makers from the offshore oil companies are prone to attach greater importance to the production availability at ordinary times, which has a strong economic effect (Boiteau et al. 2006).

The offshore industry involves large capital investment and operational costs, also the profitability of it is extremely dependent on the reliability, availability and maintainability of the facilities systems (Aven and Pedersen 2014). Thus there is the need for evaluating the availability of multi-state, multi-output systems in the oil industry (Zio et al. 2006).

The production availability is the ratio of production to planned production, or any other reference level, over a specified period of time (ISO 2008, Aven 1987). The production availability, production assurance, or production regularity have the similar meaning about how a system could meet the demand for deliveries or performance (Barabady and Aven 2008).

Many complex industrial processes are dynamic in

essence (Li and Peng 2014), including the FPSO system. Therefore Petri nets (PN) based techniques are good candidates for modeling this type of systems (Grunt and Briš 2015), which hold the relative high algorithm efficiency (Kloul et al. 2013).

PN were initially proposed by Dr. Carl Adam Petri in 1960s. The PN are both a graphical and mathematical modeling tool (Murata 1989), which is widely used in automation, communication, chemistry, economy, reliability and safety areas (Rausand 2011).

PN can deal with arbitrary probabilistic distributions. The model size of PN increases linearly associated with the component numbers, which makes it possible to model larger complex systems when compared with the peer methods (ISO 2013). For instance, the scale of the Markov chains increases exponentially along with the rise of the components number.

The stochastic Petri net (SPN) is the extended PN whose transitions are associated with random waiting times (Rausand 2011). The SPN can be utilized to compute reliability measures such as system production losses, production (un)availability, maintenance man-hours (ISO 2008). The PN family has already been applied in several offshore cases (Liu et al. 2015, Zhang et al. 2014).

In this paper, SPN are selected for modeling the production availability of the FPSO system. Our objective is to model and analyze the FPSO production availability, and conduct a realistic case study.

The remainder of this article is organized as follows. We discuss some related works in Section 2. We introduce briefly the FPSO technology in Section 3. We present the Petri nets model we designed in Section 4. Finally, we give some numerical results and discuss them in Section 5.

2 RELATED WORKS

The methods for production availability (regularity) assessment mainly include the simulation and analytical ones. For the easy recalculation of the production regularity, an analytical method based on Markov diagrams and merging rules was put forward (Kawauchi and Rausand 2002).

Zio et al. (2006) investigated the production availability of an offshore production plant using Monte Carlo simulation. The offshore installation is part of European thematic network *SAFERELNET*. Briš and Kochaníčková (2006) employed the SPN with MOCA-RP software to model the production availability of the same shared offshore system.

Zhang et al. (2014) employed the SPN in the production availability analysis of the offshore facilities with the GRIF tool. The work mainly focuses on the subsea part of the FPSO, which directly obtains the oil and gas from the subsea manifold.

3 THE FPSO SYSTEM

Our case study is a realistic FPSO system, which is now serving in an offshore oil field in South China Sea with the water depth of 90m. To analyze the production availability of the FPSO system, we concentrate on the FPSO crude oil processing system.

The FPSO system consists of four sub-systems: platforms A, B, C and the FPSO subsystem, as shown in Figure 1. Platform A transfers the oil to a buffer tank in plaform B. Together with the output oil of platforms B and C, the overall oil is transported to the heat exchangers on the FPSO. In Figure 1, the arrows with solid lines represent the main streams of the system, the crude oil flows, in which we are interested. The dashdotted arrows stand for the flows of the separated gas or waste water. The three platforms work in parallel, and are relatively independent. The system functions of the three platforms and the FPSO system are presented in the following parts.

3.1 *System description of the FPSO system*

The goal of platform A is to conduct the prefractionation and transportation of the oil from the WEllhead A (WEA). There are two main components on platform A: the Primary Separator A (PSA) and the Efflux Pump A (EPA). The PSA separates the crude oil into three parts: the gas is transported to Vent A (VA), the water is delivered to the Water Processing System A (WPSA), and the more purified crude oil flows into the EPA. The provisional destination of the crude oil from platform A is the Crude Oil Buffer Tank (COBT) on platform B.

The objective of platform B is to purify and transport the crude oil from the WEllhead B (WEB). Platform B covers the Primary Separator B (PSB), COBT, the Efflux Pumps B1 (EPB1) and B2 (EPB2). The PSB separates the crude oil into three parts: the separated gas flows to Vent B (VB), the exporting crude oil is carried to EPB1, and the remaining water is delivered to the Water Processing System B (WPSB). The oil flowing from platform A, is temporarily stored in COBT, before being transferred to platform C through EPB2. The exported oil from EPB1 and EPB2 are integrated together to the one from platform C.

The aim of platform C is to implement the purification and transportation of the initial oil flow from the WEllhead C (WEC). Like Platform A, Platform C includes two components: the Primary Separator C (PSC) and the Efflux Pump C (EPC). The former separates the initial oil into three parts: the gas is transfered to Vent C (VC), the water is delivered to Water Processing System C (WPSC), and the treated crude oil is transferred to EPC. The oil from EPC joins the flows from EPB1 and EPB2 to the FPSO.

The FPSO subsystem is employed to process and store the required crude oil. This subsystem consists of two Heat Exchangers (HEs) working in parallel,

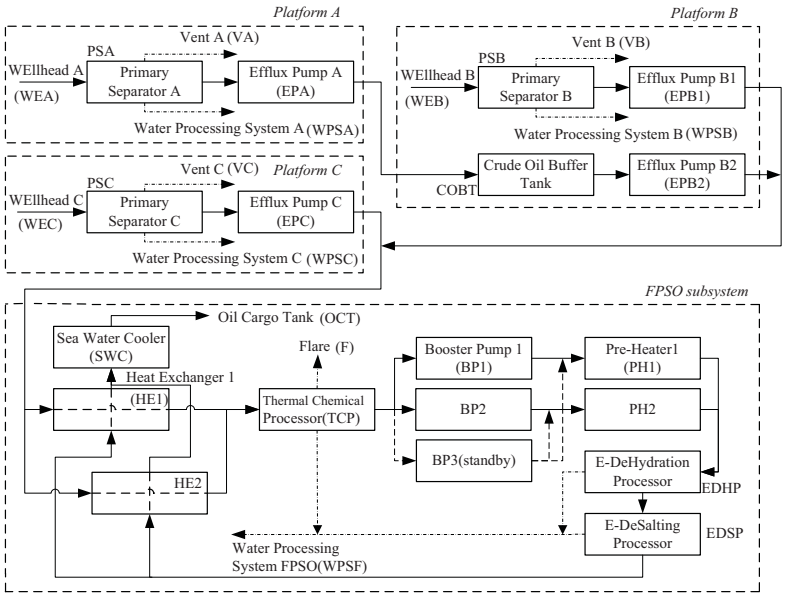


Figure 1: The flow diagram of the FPSO production system.

one Sea Water Cooler (SWC), three parallel Booster Pump (BPs), two parallel PreHeaters (PHs), as well as three processors. The main function of the FPSO is to process the primarily separated oil to become up to the standard. The overall oil flowing from platforms is initially delivered to HEs to increase its temperature.

After HEs, the flow goes to the Thermal Chemical Processor (TCP) for separating the oil from the remaining gas and water. The gas is sent to the flare for burning, and the water is delivered to the Water Processing System of the FPSO (WPSF). Regarding the oil flow, it goes to PreHeaters (PHs) through BPs. BP3 is a standby pump for redundancy, which has three states (work, failure and standby). BP3 carries load once BP1 or BP2 fails. The FPSO requires that at least two pumps work simultaneously. After the PHs, the oil flow is transferred to the Electric DeHydration Processor (EDHP). The EDHP is utilized for further dehydration, using the mixed electric field generators by the alternative and direct currents.

Normally the crude oil from the oil reservoir contains several chemical materials such as $NaCl$, $MgCl_2$ and $CaCl_2$, which are harmful to the further onshore refinery. Thus an Electro-DeSalting Processor (EDSP) is required. The remaining water from EDHP and EDSP goes to WPSF. From the EDSP, the oil flows to the HEs and SWC to decrease its temperature. The oil is then delivered to the Oil Cargo Tank (OCT), which is transported periodically (normally around once a week) to the onshore refinery using oil barges.

3.2 *Oil yield of the FPSO system*

Table 1 shows the oil yield parameters of the platforms (CNOOC 2007). The oil yields increase at the beginning and decrease to the stable values subsequently.

Table 1: Crude oil outputs of the platforms ($10^4 m^3/a$).

Year/ a	Platform A	Platform B	Platform C
1	12.9	16.7	14.0
2	164.2	213.6	178.7
3	153.8	200.1	167.4
4	83.2	108.2	90.5
5	56.9	74.0	61.9
6	44.8	58.3	48.8
7	37.3	48.5	40.6
8	31.4	40.9	34.2
9	27.2	35.4	29.6
10	21.9	28.4	23.8

3.3 *Maintenance policy of the FPSO system*

The maintenance policy of FPSO system consists of both the corrective maintenance (CM) and preventive maintenance (PM). The former is implemented right after the failures. The clock-based PM is carried out at predefined time points for the FPSO system. The maintenance strategy may include periodic tests and limited maintenance resources, like having only one repair team. In our case, the PSA, PSB, PSC, HEs and TCP are operated with both the CM and periodic PM. The other components run with the CM strategy. Table 2 shows the PM policies of the FPSO system.

4 AVAILABILITY MODELING OF THE FPSO SYSTEM

4.1 *Stochastic Petri nets overview*

Stochastic Petri nets (SPN) is an extension of Petri nets, which possesses the probabilistic time delays of

Table 2: The preventive maintenance policy of the FPSO production system.

PM level	Components	Period/ h	Duration/ h
	PSA, PSB, PSC	2190	36
PM1	HE1, HE2	4380	36
	TCP	8760	72
	PSA, PSB, PSC	2190	72
PM2	HE1, HE2	4380	72
	TCP	8760	144
	PSA, PSB, PSC	4380	36
PM3	HE1, HE2	8760	36
	TCP	17520	72

the transitions or spaces. The delays could represent the failure or repair rates of the components. When analyzing the production availability of the FPSO system, it is essential to calculate the availability of the FPSO system by the combination of the different failure or repair rates/ delays between the places.

SPN can be expressed as (Molloy 1982, Bause & Kritzinger 2002): $SPN =< P, T, A, M_0, \lambda >$ where P is the set of places, graphically represented using circles. T is the set of the transitions between the places, represented as rectangles. A stands for the directed arcs connecting transitions and places. The arcs have weights which by default are equal to 1. The arc may be an inhibitor one, which can be shown as an undirected solid line with a small circle near the transition or a dotted directed line. M_0 is the initial marking, which represents the initial numbers of tokens in different places. λ represents the corresponding transition rates of the transitions in T. Further works (Molloy 1982, Murata 1989, Bause and Kritzinger 2002) are available for the details of the SPN.

4.2 *Stochastic Petri nets model of the FPSO system*

We consider several assumptions for the production availability modeling of the FPSO system as following: The test and maintenance are perfect. Each time after the periodic repair or replacement, the components are regarded as good as new. The repair teams of the corrective and preventive maintenance are not the same. For simplicity, BPs are assumed to be repaired immediately after their failures. The corrective and preventive maintenances of one component cannot take place simultaneously.

We consider the SPN of platform A (Figure 2) as an example to illustrate the modeling of the FPSO system. Component PSA has both the CM and PM policies, while the EPA merely has the CM strategy.

- Let a token in place P1 stand for the working state of platform A and a token in P2 represent the failed state of platform A.
- Let a token in place P3 represent the working state of PSA, a token in P4 stand for the CM waiting state of PSA, and a token in P5 represent the CM repairing state of PSA.
- Let a token in P6 represent the PM waiting state of PSA and a token in P7 stand for the PM testing state of PSA.
- Let a token in place P8 represent the working state of EPA, a token in P9 stand for the CM waiting state of EPA, and a token in P10 represent the CM repairing state of EPA.
- T1 and T2 are the failure transitions caused by PSA and EPA, respectively. T3 is the working restart transition.
- T4, T5 and T6 represent the CM failure, CM start and CM repair transitions of PSA, respectively.
- T7, T8 and T9 are the PM wait, PM start and PM transitions of PSA, respectively.
- T10, T11 and T12 denote the CM failure, start and repair transitions of EPA, respectively.

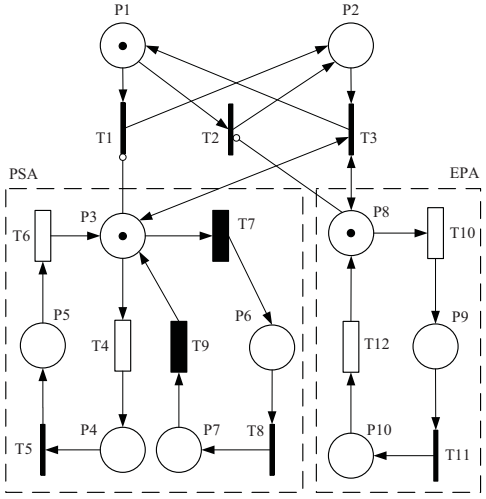


Figure 2: Stochatic Petri nets of platform A.

The firing delays of transitions T7 and T9 are deterministic, whereas T1, T2, T3, T5, T8 and T11 are instantaneous transitions. The remaining transitions (T4, T6, T10, T12) are exponentially distributed. There are also some conditions and assertions for these transitions.

- T1 is enabled if P3 is empty, which means the failure of PSA can lead to the failure of platform A. T2 works alike with T1.
- T3 is enabled if both P3 and P8 have tokens, which implies that platform A can restart work if both of PSA and EPA are working.
- The conditions of firing T4 are that PSA is working (P3 is non-empty) and PSA is not in PM process. After firing T4, the state of PSA is asserted to failed. The availability of PSA is assigned to be 0. T10 operates similar with T4.
- T5 is enabled once the corrective maintenance team is available. After its firing, the corrective maintenance team becomes unavailable. T11 works similar with T5.
- After firing T6, the corrective maintenance team turns into available and the state of PSA is assigned to working. The availability of PSA is assigned to be 1. T12 runs alike with T6.

Table 3: Failure data and maintenance policy of FPSO system.

Components	FR/ h^{-1}	RT/ h	Maintenance
PSA,PSB,PSC	2.63e-4	8.2	PM+CM
EPA,EPB1,EPB2,EPC	5.51e-4	32.7	CM
COBT	1.63e-4	30.3	CM
HE1,HE2	2.00e-4	8.9	PM+CM
TCP,EDHP,EDSP	2.63e-4	8.2	PM+CM
BP1,BP2,BP3	1.03e-3	5.0	CM
PH1,PH2	6.96e-5	42.2	CM
SWC	9.78e-5	70.0	CM

- T7 is enabled if PSA is working and PSA is not in the PM state. After firing T7, PSA is asserted to be in PM state. The availability of PSA is assigned to be 0.
- T8 is enabled once the preventive maintenance team is available. After its firing, the preventive maintenance team becomes unavailable.
- After firing T9, PSA is asserted to be out of PM state, the PM team turns into available. The availability of PSA is assigned to be 1.

The modeling of platforms B and C is similar to platform A. Figures A1 and A2 in the end of the paper show the SPNs of the platforms and the FPSO subsystem, respectively. The three platforms are in parallel relationship, but they are dependent, because they share the same preventive and corrective repair teams. Based on the logic understanding and the preliminary risk assessment of the system, we assume that the failure of PHs, EDHP, EDSP and SWC cannot lead to the failure of the FPSO subsystem.

Finally, we obtain the whole SPN model of the FPSO system with 89 places and 112 transitions. 74 of the transitions are deterministic (with 62 instantaneous) and 38 of them are exponentially distributed.

The predicated oil yield (production) of the FPSO system $Y(t)$ is given by:

$$Y(t) = A_F(A_A Y_A + A_B Y_B + A_C Y_C) \quad \text{for } t > 0 \quad (1)$$

where A_F is the availability of the FPSO subsystem. A_A and Y_A are the availability and oil yield of platform A, respectively. It is similar when it comes to platforms B and C. At each time, we can compute the total yield after attaining the availabilities of the subsystems.

The production availability of the FPSO system $P(t)$ is defined as:

$$P(t) = \frac{Y(t)}{E(t)} \quad \text{for } t > 0 \quad (2)$$

where $E(t)$ is the expected oil yield of the FPSO system, which can be obtained from Table 1.

5 NUMERICAL RESULTS

5.1 *Input data*

Table 3 shows the failure rate (FR), the repair time (RT) and the maintenance policy of the platforms and FPSO subsystem (OREDA Participants 2002). In our experiments, we use the GRIF package, a tool developed by TOTAL (2014). By employing the GRIF-Predicates Petri Nets Module, the SPN model of the FPSO production system can be attained.

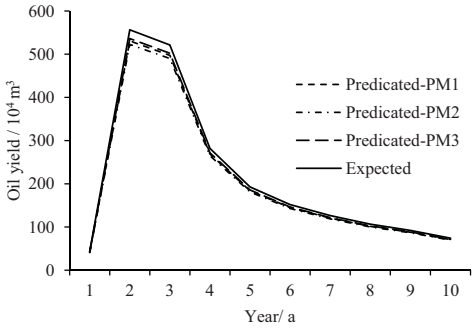


Figure 3: Expected and predicated annual oil yields of the FPSO system .

5.2 *The production availability results*

The number of histories (the number of Monte Carlo simulations) in our case is 20,000. We use Equation 1 to compute the predicated oil production output of the FPSO system.

Figure 3 shows the comparison of the predicated and expected oil production of the FPSO system, under different preventive maintenance policies. The predicated oil yield shares the alike tendency with the expected one. The results show that the increase of the PM duration time can decrease the predicated oil yield, and the growth of the PM period time can bring the contrary results. This figure reveals that the predicated oil yield is less than the expected one, thus the production availability ought to be paid attention to.

Figure 4 demonstrates the cumulative oil losses under three PM policies, which are 99.02, 129.22, 78.66×10^4m^3, respectively. Because the ratio of PM duration time to period time of the PM level 2 is relatively larger than the one of PM1 and PM3, the FPSO system with PM2 policy can generate more oil losses. Although the oil loss seems little than the cumulative oil yield, the economic effect of the oil loss (production unavailability) cannot be ignored.

Table 4 indicates that the availability and the production availability of the FPSO system are different in our case. The system availability is attained from the logic relationship of the components. Whereas the production availability is obtained by the ratio of predicated and expected production outputs. Based on Equation 1, the difference of the two availabilities can be caused by the different oil yields of the platforms. The choice of PM3 policy can attain a higher production availability. From the results, we learned that this FPSO system is highly reliable and available.

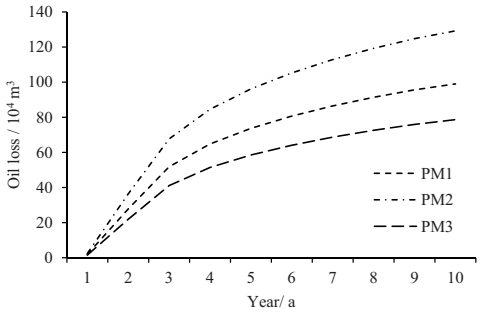


Figure 4: Cumulative oil loss under different preventive maintenance policies.

Table 4: Production availability and system availability under different preventive maintenance policies.

PM policies	PM1	PM2	PM3
Production availability	0.9541	0.9403	0.9636
Availability	0.9943	0.9920	0.9964

6 CONCLUSION

Stochastic Petri nets made it possible to assess the production availability of the FPSO system. The results we obtain are very interesting from an industrial standpoint. Our study shows however that SPN are quite difficult to master when the system under study gets complex. Without our assumptions, the SPN model can become more complex. SPN models are relatively easy to design, but quite hard to update and revise. It makes us a bit worrying since the models we shall design in the next phases of our study will get even more complex so to take into account interactions and working conditions.

Therefore, we think about moving to other modeling formalisms. For instance, the reliability block diagrams (RBD) driven Petri nets is regarded as effective way to improve the readability and the understandability of PNs (Signoret et al. 2013).

We are also planning to use a higher level modeling language like AltaRica (Boiteau et al. 2006), which is considered as more powerful when it comes to the modeling flexibility, efficiency and understandability.

REFERENCES

Aven, T. (1987). Availability evaluation of oil/gas production and transportation systems. *Reliability Engineering 18*(1), 35–44.

Aven, T. & L. M. Pedersen (2014). On how to understand and present the uncertainties in production assurance analyses, with a case study related to a subsea production system. *Reliability Engineering & System Safety 124*, 165–170.

Barabady, J. & T. Aven (2008). A methodology for the implementation of production assurance programmes in production plants. *Journal of Risk and Reliability 222*, 283–290.

Bause, F. & P. S. Kritzinger (2002). *Stochastic Petri Nets: An Introduction to the Theory*. Vieweg+Teubner Verlag.

Boiteau, M., Y. Dutuit, A. Rauzy, & J.-P. Signoret (2006). The AltaRica data-flow language in use: modeling of production availability of a multi-state system. *Reliability Engineering & System Safety 91*(7), 747–755.

Briš, R. & M. Kochaníčková (2006). Stochastic Petri net approach to production availability evaluation of special test case. *Safety and Reliability for Managing Risk 2*, 1569–1575.

BW Offshore (2015). All missing recovered on Cidade de São Mateus. http://www.bwoffshore.com/news1/all-missing-recovered-on-cidade-de-sao-mateus/. [accessed 26-May-2015].

CNOOC (2007). FPSO manual for operation and maintenance.

Grunt, O. & R. Briš (2015). SPN as a tool for risk modeling of fires in process industries. *Journal of Loss Prevention in the Process Industries 34*, 72–81.

HSE (2003). Analysis of accident statistics for floating monohull and fixed installations.

ISO (2008). ISO 20815, Petroleum, petrochemical and natural gas industries: Production assurance and reliability management.

ISO (2013). ISO/TR 12489, Petroleum, petrochemical and natural gas industries: Reliability modelling and calculation of safety systems.

Kawauchi, Y. & M. Rausand (2002). A new approach to production regularity assessment in the oil and chemical industries. *Reliability Engineering & System Safety 75*(3), 379–388.

Kloul, L., T. Prosvirnova, & A. Rauzy (2013). Modeling systems with mobile components: a comparison between AltaRica and PEPA nets. *Journal of Risk and Reliability 227*(6), 599–613.

Lake, L. W. & K. E. Arnold (2007). *Petroleum Engineering Handbook: Facilities and Construction Engineering*. Society of Petroleum Engineers.

Li, Y. & R. Peng (2014). Availability modeling and optimization of dynamic multi-state series–parallel systems with random reconfiguration. *Reliability Engineering & System Safety 127*, 47–57.

Liu, Z., Y. Liu, B. Cai, X. Li, & X. Tian (2015). Application of Petri nets to performance evaluation of subsea blowout preventer system. *ISA transactions 54*, 240–249.

Mannan, S. (2005). *Lees' Loss prevention in the process industries: Hazard identification, assessment and control*. Elsevier Butterworth-Heinemann.

Molloy, M. K. (1982). Performance analysis using stochastic petri nets. *Computers, IEEE Transactions on 100*(9), 913–917.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE 77*(4), 541–580.

OREDA Participants (2002). Offshore reliability data handbook.

Rausand, M. (2011). *Risk assessment: Theory, methods, and applications*. John Wiley & Sons.

Signoret, J.-P., Y. Dutuit, P.-J. Cacheux, C. Folleau, S. Collas, & P. Thomas (2013). Make your Petri nets understandable: Reliability block diagrams driven Petri nets. *Reliability Engineering & System Safety 113*, 61–75.

Tillie Nutter (2014). 2014 Worldwide survey of Floating Production, Storage and Offloading (FPSO) units. http://www.offshore-mag.com/content/dam/offshore/print-articles/volume-74/08/2014FPSO-072214-Ads.pdf. [accessed 26-May-2015].

TOTAL (2014). User manual of the GRIF- Petri nets with predicates.

Zhang, H., F. Innal, F. Dufour, & Y. Dutuit (2014). Piecewise deterministic Markov processes based approach applied to an offshore oil production system. *Reliability Engineering & System Safety 126*, 126–134.

Zio, E., P. Baraldi, & E. Patelli (2006). Assessment of the availability of an offshore installation by Monte Carlo simulation. *International Journal of Pressure Vessels and Piping 83*(4), 312–320.

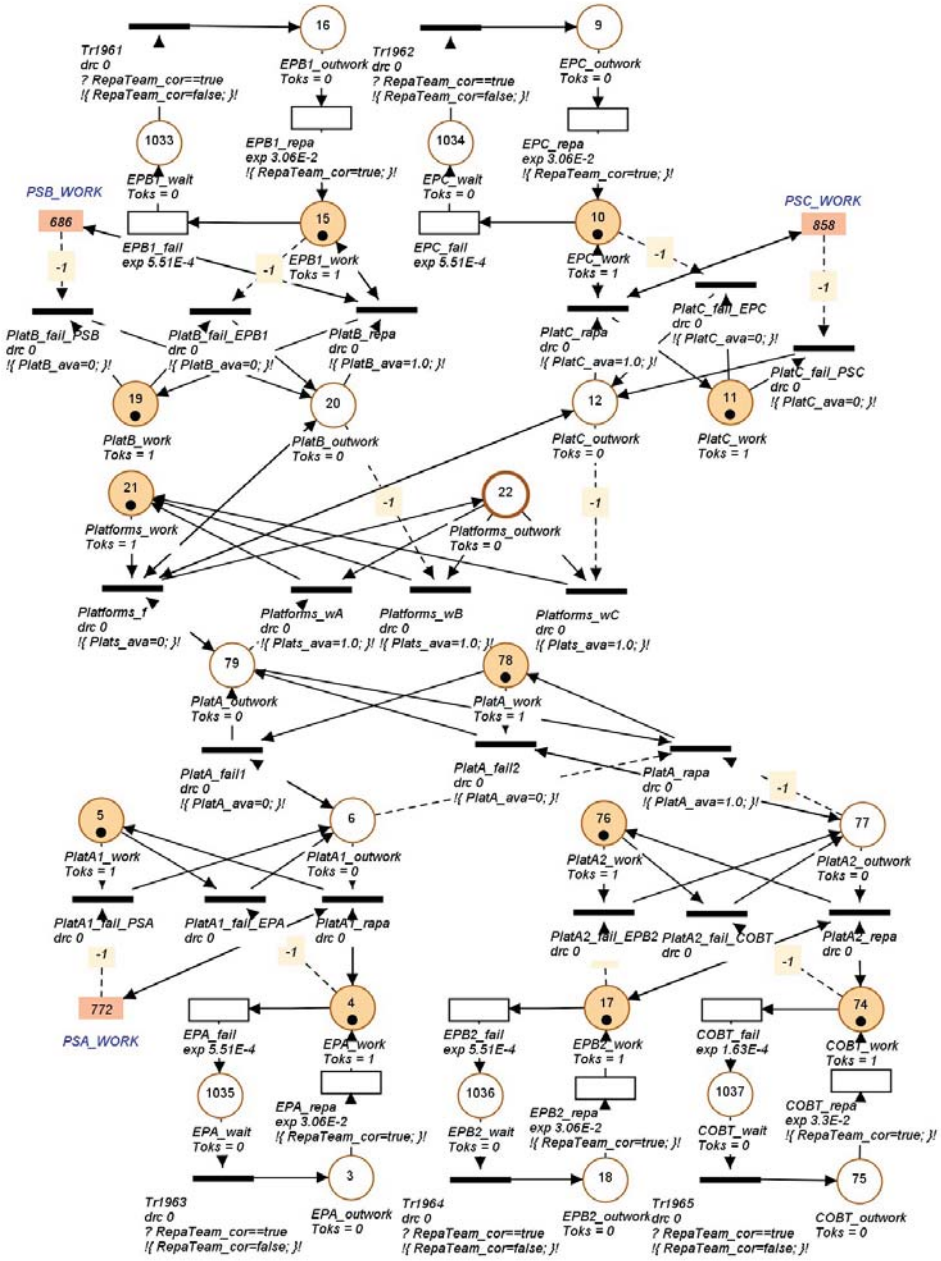


Figure A1: Stochatic Petri nets of three platforms.

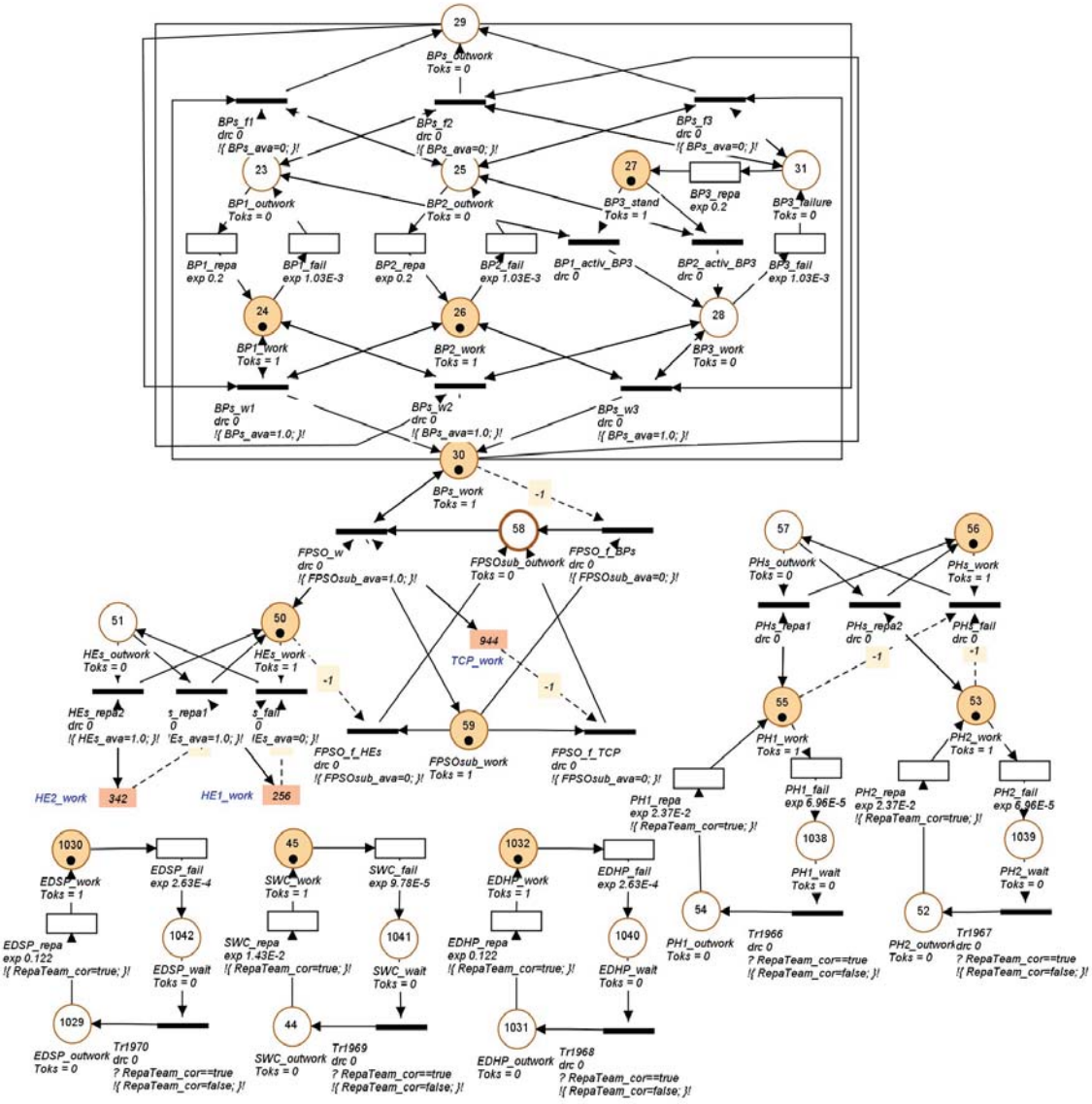


Figure A2: Stochatic Petri nets of the FPSO subsytem.

Appendix B

Modeling Patterns for Production Performance Analysis

Production systems own shared behaviors, so-called modeling patterns, like the single-input-single-output physical structure and the preventive maintenance strategy. In this chapter, we show our work of pattern-based production performance analysis[1].

To ease modeling missions when analyzing the production performance, we propose a set of modeling patterns. We implement modeling patterns, which capitalize the modeling knowledge, with the AltaRica 3.0 modeling language. We apply proposed modeling patterns on a practical offshore installation. Results generated by using modeling patterns agree well with the ones in literature. Our study indicates that it is beneficial to reuse the capitalized knowledge for modeling production systems. In summary, highlights of this work are:

- A set of modeling patterns for production availability analysis is proposed.
- Modeling patterns are defined and classified.
- Modeling patterns are applied on a complex production system.
- Comparison is conducted between results from literature and those from using modeling patterns.

[1] H. Meng, L. Kloul, A. Rauzy. Modeling patterns for performance analyses of offshore production systems, Proceedings of the 27th International Ocean and Polar Engineering Conference, International Society of Offshore and Polar Engineers (**ISOPE 2017**), San Francisco, California, USA, Jun. 2017: 1199-1205.

Modeling Patterns for Performance Analyses of Offshore Production Systems

Huixing Meng[a], Leïla Kloul[b], Antoine Rauzy[c]
a. LIX, École Polytechnique, Palaiseau, France
b. DAVID, Université de Versailles St-Quentin-en-Yvelines, Versailles, France
c. Department of Mechanical and Industrial Engineering, Norwegian University of Science and Technology, Trondheim, Norway

ABSTRACT

To ease the modeling missions when analyzing the production performance, we proposed a set of modeling patterns. The production systems own shared behaviors, the so-called modeling patterns, like the single-input-single-output physical structure and the preventive maintenance strategy. We implement the modeling patterns, which capitalize the modeling knowledge, with the AltaRica 3.0 modeling language. We apply the proposed modeling patterns on a practical offshore installation. The results generated by the modeling patterns agree well with the ones in the literature. Our study indicates that it is beneficial to reuse the capitalized knowledge for modeling the production systems.

KEY WORDS: Modeling patterns; Production performance; AltaRica 3.0 modeling language; Offshore installation.

INTRODUCTION

The performance of the production system is crucial for the process industry, such as the oil and chemical plants. The production facilities in process industry confront two types of risk. First, the low-probability/high-consequence incidents, like the severe accidents (such as the fire and explosion), are constantly attracting high attention from both industry and society. Second, the high-probability/low-consequence incidents (Signoret, 2010), like the production losses, require further focus from the stakeholders of the production systems.

Several definitions for assessing the production system are commonly utilized. *Production performance* is the capacity of a system to meet demand for deliveries or performance (NORSOK,1998; ISO, 2008). *Production-performance analysis* refers to the systematic evaluations and calculations carried out to assess the production performance (NORSOK,1998; ISO, 2008). Conducting production-performance analyses contributes to assess the production losses, thus to check if the system complies with the production requirements. *Production availability* can measure the production performance, which is the ratio of real production to the expected production, or to a reference level, in a period of time (NORSOK,1998; ISO, 2008).

Modeling the production systems is challenging when the systems are complex. It is resource-consuming when the updates of these models are required. The models are expected to be improved by increasing the capability for updating and maintaining the models in the life-cycle of the systems. The high-level modeling languages are alternatives for dealing with these issues.

The formal modeling formalism, such as the AltaRica 3.0 modeling language (Prosvirnova et al., 2013; Prosvirnova, 2014; Lipaczewski et al., 2015; Bateaux et al., 2015), is more near to the targeted systems and easier to be updated in different life phases of the system. AltaRica 3.0 is dedicated to safety and performance analyses (Prosvirnova et al., 2013; Prosvirnova, 2014; Lipaczewski et al., 2015; Bateaux et al., 2015).

An advantage of the formal modeling languages is to reuse the models of components or even subsystems (Prosvirnova, 2014). There are two ways for attaining such an objective: reuse of components (objective-oriented) and modeling patterns (prototype-oriented) (Prosvirnova, 2014). The reuse of components comes directly from programming languages, like C++, or modeling languages, like Matlab/Simulink and Modelica (Fritzson, 2010). The reuse of the modeling patterns is to begin with an existing code, duplicate and adapt it to meet the specific requirements (Prosvirnova, 2014).

The experience shows that it is rewarding to employ the modeling patterns in AltaRica environment (Kehren, 2005). When assessing the performance of a production system, the modeling patterns are expected to relieve the modeling task. The benefits from using modeling patterns are: first, the modeling experience is capitalized; second, the blank can be avoided at the initial stage and the systems can be abstracted at a reasonable level.

The objective of this article is to illustrate how to analyze the production performance using the capitalized modeling knowledge. In this article, we propose the modeling patterns for performance analysis of production systems, especially in process industry. The applicability of the modeling patterns is tested with an offshore production system.

The rest of this paper is structured as follows. First, we review the related works. Second, an offshore production system is illustrated as a running example. The modeling patterns found in the system are listed as well. Third, we propose the formal definitions of the modeling patterns and implement the modeling patterns with AltaRica 3.0 modeling language. Eventually, we present the methodology to recognize the modeling patterns and apply them for modeling the production system.

RELATED WORKS

Production-performance analyses

The standards issued by NORSOK and ISO give basic guidelines for production-performance analyses (NORSOK,1998; ISO, 2008). Analytical and simulation methods are utilized for evaluating the production performance (Aven,1987; Vesteraas, 2008). Simple systems can be evaluated using the analytical methods. In practice, most of the systems are evaluated with simulation methods.

The production availability of an offshore production plant is studied using the Monte Carlo simulation (Zio et al., 2006). This offshore test case is part of an European thematic network SAFERELNET. SPN (Stochastic Petri Nets) is employed to study the production performance of the

same system (Briš et al., 2006).
SPN is also used for calculating the production availability of the topside (Meng et al., 2015) and subsea part (Zhang et al., 2014) of the FPSO (Floating Production, Storage and Offloading unit).
An oil production system is proposed and analyzed using (Rauzy, 2004). The performance of this multi-state production system is also evaluated with AltaRica Data-Flow language (Boiteau et al., 2006), the 2.0 version of the AltaRica language.
Modeling experience can be obtained from the above studies. Few research reports the methodology to reuse such knowledge to analyze the production performance.

Modeling patterns

Pattern can be utilized for reusing the capitalized knowledge, which was formally proposed in civil engineering (Alexander, 1977). The concept was adopted in software engineering afterwards, which was known as *design patterns* (Gamma et al., 1995). The design patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context (Gamma et al., 1995).
Some pattern-related concepts try to provide a general framework of reusing patterns. The pattern based system engineering (PBSE) was proposed, whose procedure includes the pattern definition and system development with patterns (Hamid et al., 2016). The dependability pattern is proposed (Hamid et al., 2016), which is defined as the description of a particular recurring dependability problem that arises in specific contexts and presents a well-proven generic scheme for its solution.
Basing on the modeling experience of several aircraft systems using AltaRica Data-Flow language, the safety architecture patterns (SAP) are proposed to simplify the modeling missions (Kehren, 2005). SAP are the component assemblies used to ensure the safety of the architectures (Kehren, 2005). The application of SAP can be found in the avionics domain (Kehren, 2005; Morel, 2014).
A library of patterns is proposed to capture known solution algorithms and architectural measures/constraints (Khalil et al., 2014). This library focuses on the safety mechanisms in the automotive domain.
The aforementioned pattern-related studies focus on safety domains (Kehren, 2005; Morel, 2014; Khalil et al., 2014), which cannot be applied directly to evaluate the production performance.

RUNNING EXAMPLE

We have modeled several production systems (Kawauchi et al., 2002; Rauzy, 2004; Zio et al., 2006; Meng et al., 2015), with the system in (Kawauchi et al., 2002) is a production facility, the system in (Rauzy, 2004) is an oil production system, the system in (Zio et al., 2006) is a SAFERELNET offshore installation, and the one in (Meng et al., 2015) is a FPSO.
The SAFERELNET system (Signoret, 2000; Zio et al., 2006) encompasses many modeling difficulties encountered when conducting studies related to the production processes (Signoret, 2000). We consider this SAFERELNET production system, an offshore installation, as a running example throughout this study.
Figure 1 shows the flow diagram of the SAFERELNET system. SEP (Separation unit) divides the flow from the well (WEL) into three types of flows: gas, oil and water. The separated `gas` firstly flows to two parallel Turbo-Compressors (TC1 and TC2), and then to TEG (Tri-Ethylene Glycol) for dehydration. Before the final gas export, there are three gas flows which are employed for fueling and lifting operations. One part of the fuel gas returns to support the work of TCs, while the other part of fuel gas flows assists the electricity generation in TGs (Turbo-Generators). The obtained electricity is utilized by units TEG, EC (Electro-Compressor), OPS (Oil Pumping System) and WPS (Water Pumping System). Finally, the lift gas flow goes back to the well for assisting well production. The isolated `oil` flow is exported through OPS. To maintain the well pressure, the separated `water` is sent for water injection through WPS.

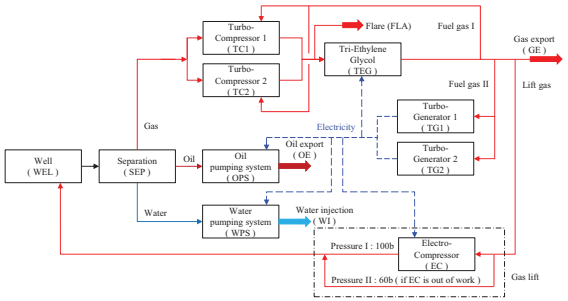


Fig. 1 Flow diagram of the SAFERELNET system, redraw from (Signoret, 2000; Zio et al., 2006).

Three types of preventive maintenance are considered for TCs and TGs, where the preventive intervals are every 2160h, 8760h, and 43800h (Zio et al., 2006). Moreover, the corresponding preventive maintenance durations are different. The required data for analyzing the production performance of SAFERELNET system are too numerous to list here, but can be found in (Zio et al., 2006).

MODELING PATTERNS

A modeling pattern (MP) is a modeling means which allow of modeling recurrent behaviors and structures. In the following, we distinguish between the functional and the physical modeling patterns. These patterns are extracted from the studied systems (Kawauchi et al., 2002; Rauzy, 2004; Zio et al., 2006; Meng et al., 2015).

Functional modeling patterns

Functional modeling patterns capture shared behaviors between the systems. Figure 2 provides the following functional modeling patterns.
(1) `PERFECT` pattern: it stands for the perfect components, which are always working and cannot fail. In the SAFERELNET system (Figure 1), units WEL, SEP, OPS, WPS are modeled using such a pattern.
(2) `CorrectiveMaintenance` pattern: it models the repairable components. The repairable components have three states, which are `WORKING`, `FAILED`, and `UNDER_REPAIR`. The component stays initially in the `WORKING` state. Once a failure occurs, the component turns into the `FAILED` state. If the corrective repair crew is available, the component state becomes `UNDER_REPAIR`. Subsequently, the component returns to `WORKING` state once the repair operation is completed. In the SAFERELNET system, unit TEG can be modeled using this pattern.
(3) `PreventiveMaintenance` pattern: it represents the repairable components with preventive maintenance. This can be seen as the `CorrectiveMaintenance` pattern to which the preventive maintenance action is added. Thus in addition to the transitions described in the `CorrectiveMaintenance` pattern, once the preventive interval is reached and the preventive maintenance crew is available, the component becomes `UNDER_MAINTENANCE`. The component returns to the `WORKING` state once the preventive maintenance is finished. In the SAFERELNET system, unit EC can be modeled using such a pattern.
(4) `DEGRADATION` pattern models the degraded behaviors between the `WORKING` and `FAILED` states. When the component degrades, it becomes less available and more vulnerable. Once there is a degradation, the

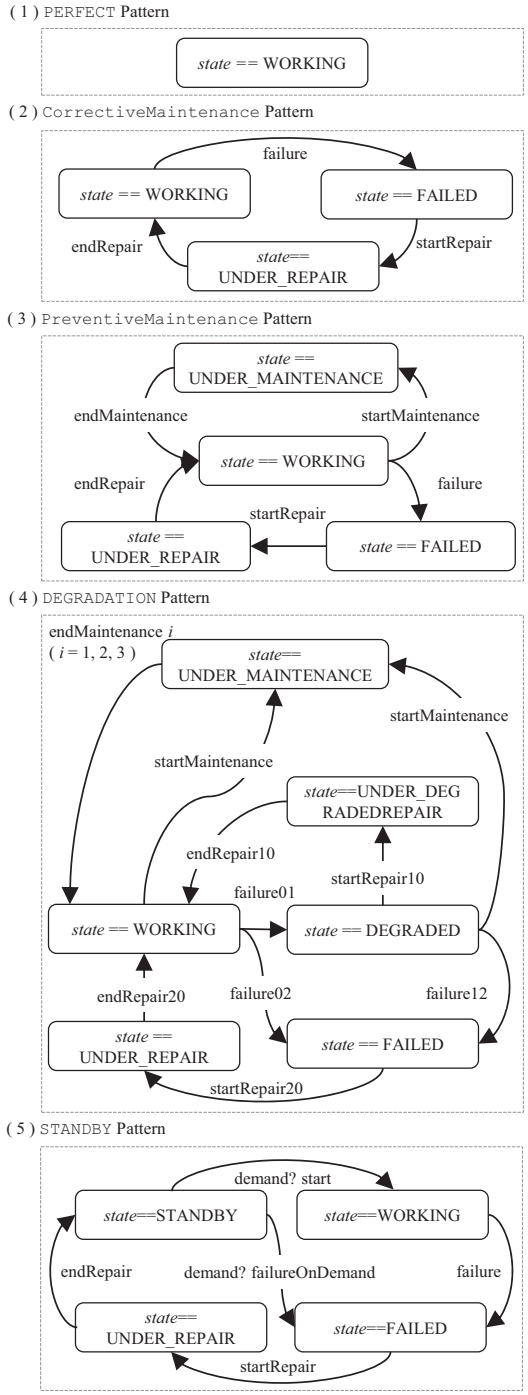


Fig. 2 Functional modeling patterns.

component transfers to DEGRADED state. If the corrective repair crew is available, the component state turns into UNDER_DEGRADEDREPAIR. The degraded component can also fall into the FAILED state. Once the corrective repair crew is available, the component transfers from WORKING to UNDER_REPAIR. The component returns to WORKING once the repair is finished. Once the preventive interval is reached and the preventive maintenance crew is available, the component becomes UNDER_MAINTENANCE. In the SAFERELNET system, units TCs and TGs can be modeled using this pattern.

(5) STANDBY pattern models the standby behaviors. The standby components with corrective maintenance adds the STANDBY state to the CorrectiveMaintenance pattern. The initial state of the component is STANDBY. When there is a demand, the component becomes WORKING. After repair, the component returns to STANDBY state. There is no such a pattern in SAFERELNET system. An example is a standby booster pump in FPSO (Meng et al., 2015).

Physical modeling patterns

The physical modeling patterns capture the structures which are similar in the systems. There are two types of physical modeling patterns. One type of the patterns is based on the input and output flows (component-level). The other patterns are based on the structure interactions between the components (subsystem-level). Figure 3 shows the physical modeling patterns.

The component-level physical modeling patterns mainly include the SOURCE, SISO, MIMO, SIMO, and SINK patterns.

(1) SOURCE pattern has only outputs, which runs under particular conditions. Units WEL, TG1 and TG2 in SAFERELNET system are modeled with such a pattern. WEL produces three outflows (gas, oil and water) under different conditions. TG1 and TG2 generate electricity for the components TEG, EC, OPS and WPS, with the condition that there is gas fueling the generators.

(2) SISO (Single-Input-Single-Output) pattern has one inflow and one outflow. Units TC1, TC2, OPS, WPS and EC in SAFERELNET system are modeled with the SISO pattern. The conditions for the working of TC1 and TC2 are the capacity limit and adequate fuel gas. The prerequisites for running OPS, WPS and EC are sufficient electricity from TG1 and TG2.

(3) MIMO (Multiple-Input-Multiple-Output) pattern has at least two inflows and two outflows. Unit SEP in SAFERELNET system is a MIMO structure. The three inflows come from WEL, and the four outflows turn into TC1, TC2, OPS and WPS. There is a capacity constraint for SEP, which processes the limited oil, gas and water.

(4) SIMO (Single-Input-Multiple-Output) pattern splits one inflow into multiple outflows. Unit TEG in SAFERELNET system is a single-inflow-multi-outflow structure. The inflow comes from the combination of the outflows of TC1 and TC2. Part of the outflows goes to TC1, TC2, TG1 and TG2 as fuel gas. Part of the outflows turns into WEL, may via EC when EC works, for gas lift. The residual is the final output gas of the system.

(5) SINK pattern models the physical structure that only has inflows. There is no such a pattern in the SAFERELNET system. This pattern can be used to model a tank in an oil production system (Rauzy, 2004).

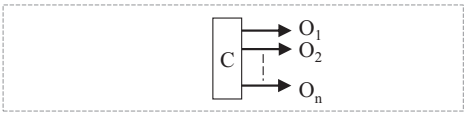
The subsystem-level pattern is composed of several component-level patterns. The subsystem-level physical modeling patterns mainly include the SERIES, PARALLEL, and KooN patterns.

(6) SERIES pattern describes the series structure. SERIES models the series connection of several SISO patterns. The primary separator and efflux pump in FPSO (Meng et al., 2015) are modeled using such a pattern.

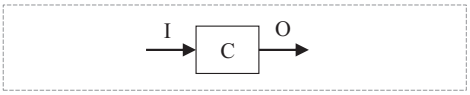
(7) PARALLEL pattern describes the parallel structures. It models the parallel connection of several SISO patterns. In SAFERELNET system, units TCs and TGs can be modeled using this pattern.

(8) KooN (k-out-of-n) pattern describes the structure which works when at least k of the total n items must be functioning. An example is the booster pumps in FPSO, which is a 2-out-of-3 structure (Meng et al.,

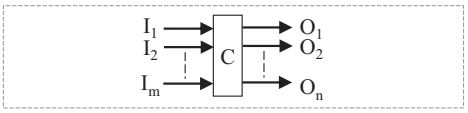
(1) SOURCE Pattern



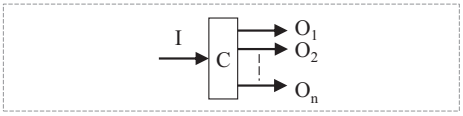
(2) SISO : Single-Input-Single-Output Pattern



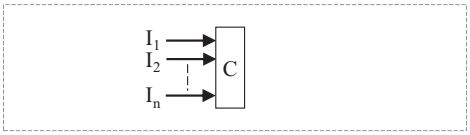
(3) MIMO : Multiple-Input-Multiple-Output Pattern



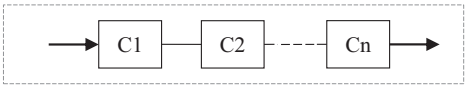
(4) SIMO : Single-Input-Multiple-Output Pattern



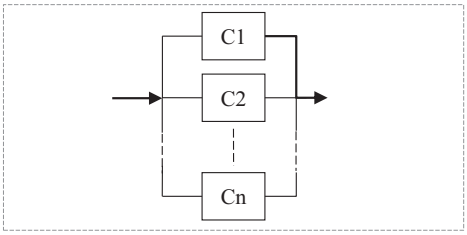
(5) SINK Pattern



(6) SERIES Pattern



(7) PARALLEL Pattern



(8) KooN: k-out-of-n Pattern

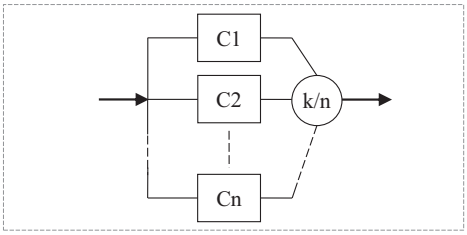


Fig. 3 Physical modeling patterns.

2015).

DEFINITIONS OF MODELING PATTERNS

In this section, we formally define the modeling patterns. The modeling patterns are first defined from the functional and physical viewpoints. Subsequently, the modeling patterns are given by integrating both the functional and physical ones.

A *functional modeling pattern* (FMP) is a shared functional behavior in the production systems. For example, the repairable and degradation behaviors are two independent functional modeling patterns. The pattern resulting from the combination of these patterns is also a functional modeling pattern. FMP construction relies on the finite-state automata of the patterns (see e.g. (Rosen, 2011)). Here we provide the formal definition of FMP first.

Definition 1. A functional modeling pattern is the tuple $\langle \mathcal{S}, \mathcal{S}_0, \mathcal{S}_f, \mathcal{T}, \mathcal{D} \rangle$ where:

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{S}_0$ is a finite set of initial states, $\mathcal{S}_0 \subseteq \mathcal{S}$.
- $\mathcal{S}_f$ is a finite set of final states, $\mathcal{S}_f \subseteq \mathcal{S}$.
- $\mathcal{T}$ is a finite set of state transitions. A transition $t \in \mathcal{T}$ can be immediate or be associated with delayed.
- $\mathcal{D}$ is a finite set of demands. The demand is a Boolean variable.

The set $\mathcal{S}_f$ can be empty. It is nonempty when the component is non-repairable. $\mathcal{T}$ can be omitted when encountering the perfect behavior. $\mathcal{D}$ stands for the pre-conditions for some transitions in cold-standby components.

Example 1: Consider the STANDBY pattern in Figure 2 as an example:

- $\mathcal{S}$ = { STANDBY, WORKING, FAILED, UNDER_REPAIR }.
- $\mathcal{S}_0$ = { STANDBY }. The initial state is STANDBY.
- $\mathcal{S}_f = \emptyset$. There is no final state in STANDBY pattern.
- $\mathcal{T}$ = { start, failureOnDemand, failure, startRepair, endRepair }.
- $\mathcal{D}$ = { demand }. There is a demand in STANDBY pattern.

A *physical modeling pattern* (PMP) is a shared structural behavior in the production systems. For instance, a Single-Input-Single-Output structure is a PMP. Here we give the formal definition of the PMP.

Definition 2. A physical modeling pattern is defined by the tuple $\langle \mathcal{F}_{in}, \mathcal{F}_{out}, \mathcal{S}_c, \mathcal{C}_{cap}, \mathcal{C}_{con}, \mathcal{I} \rangle$ where:

- $\mathcal{F}_{in}$ and $\mathcal{F}_{out}$ are the finite sets of inflows and outflows of the physical structures, respectively. $\mathcal{F}_{in}$ and $\mathcal{F}_{out}$ are disjoint sets, that is, $\mathcal{F}_{in} \bigcap \mathcal{F}_{out} = \emptyset$. The values of the flows can be Boolean, integer or floating-point numbers.
- $\mathcal{S}_c$ stands for the current state of the physical structure, $\mathcal{S}_c \subseteq \mathcal{S}$.
- $\mathcal{C}_{cap}$ is a finite set of capacity constraints on the physical structure
- $\mathcal{C}_{con}$ is a finite set of condition constraints on the physical structure. The condition is a Boolean variable.
- $\mathcal{I}$ is a finite set of indicators with regard to the system stages. An indicator $i \in \mathcal{I}$ is defined by a tuple $\langle i_0, \mathcal{T}_i \rangle$:
  - i_0 is the initial indicator of the system, $i_0 \in \mathcal{I}$.
  - $\mathcal{T}_i$ is a set of transitions between the indicators.

For each PMP, $\mathcal{F}_{in} \neq \emptyset$ or $\mathcal{F}_{out} \neq \emptyset$. In extreme situations, such as SINK pattern, only $\mathcal{F}_{in}$ is required. Similarly, SOURCE pattern has only $\mathcal{F}_{out}$.

The functional modeling pattern provides the current state $\mathcal{S}_c$ for the physical ones. This one represents the connection between both types of modeling patterns.

The set $\mathcal{C}_{cap}$ can be empty because some structures may be assumed to handle unlimited capacities. Similarly, the set $\mathcal{C}_{con}$ can be empty as well. For example, the required gas for the turbo-generator in the offshore installation is treated as a condition constraint, not as an inflow. This is because the materials of inflow and outflow are not identical.

The set $\mathcal{I}$ is often used in the looped structures. Normally we assume that $\mathcal{I}$ = { 0, 1, 2 }. 0 indicates that the system is at the initial stage, which needs to be launched to work. 1 means that the system is failed. 2 represents that the system returns to work after repair, if it is repairable.
Example 2: Consider a `SISO` (Single-Input-Single-Output) pattern, which works with the `CorrectiveMaintenance` pattern:

- $\mathcal{F}_{in}$ = { InFlow }.
- $\mathcal{F}_{out}$ = { OutFlow }.
- $\mathcal{S}_c$ = { `WORKING`, `FAILED`, `UNDER_REPAIR` }. $\mathcal{S}_c$ is provided by the `CorrectiveMaintenance` pattern.
- C_{cap} = { 80% }. We assume the capacity constraint is 80%.
- C_{con} = { true, false }.
- $\mathcal{I}$ = { 0 }. With regard to `SISO` pattern, this is the only indicator and there is no indicator transition.

After defining the physical and functional modeling patterns, we give the definition of the modeling patterns. Modeling patterns can be directly utilized for modeling the targeted production systems. When modeling the targeted systems, functional modeling pattern and physical modeling pattern work jointly to complete the mission. We define the modeling pattern (MP) by integrating the FMP and PMP. For example, a `PreventiveMaintenance` pattern which works with the `SIMO` pattern belongs to MP. Note that FMP and PMP are disjoint sets, that is, $FMP \bigcap PMP = \emptyset$.

Definition 3. A modeling pattern is a tuple $\langle \mathcal{S}, \mathcal{S}_0, \mathcal{S}_f, \mathcal{T}, \mathcal{D}, \mathcal{F}_{in}, \mathcal{F}_{out}, \mathcal{S}_c, C_{cap}, C_{con}, \mathcal{I} \rangle$. The notations are those in Definitions 1 and 2. The set $\mathcal{S}$ in FMP provides current state $\mathcal{S}_c$ in PMP.

MODELING PATTERNS IMPLEMENTATION

In order to implement the library of defined modeling patterns, we consider using the AltaRica 3.0 modeling language. The Guarded Transition Systems (GTS), which is the mathematical foundation of the AltaRica 3.0 modeling language, is formally defined in (Rauzy, 2008). A GTS is a quintuple $\langle V, E, T, A, \iota \rangle$ where V is a finite set of variables, E is a finite set of events, T is a finite set of transitions, A is an assertion and ι is the initial assignment of variables (Rauzy, 2008).
GTS is capable of implementing the elements in the modeling patterns:

- V can model $\mathcal{S}$ and $\mathcal{S}_f$ in FMP, as well as $\mathcal{F}_{in}$, $\mathcal{F}_{out}$, $\mathcal{S}_c$ and $\mathcal{I}$ in PMP.
- E can model $\mathcal{T}$ in FMP and $\mathcal{T}_i$ in PMP.
- T can model $\mathcal{D}$ in FMP, as well as C_{cap} and C_{con} in PMP.
- A is the instruction of variable in V.
- ι can model $\mathcal{S}_0$ in FMP and $\mathcal{I}_0$ in PMP.

We implement the proposed modeling patterns, which are listed in Figures 2 and 3, using AltaRica 3.0 language. Here we use an example to show how to implement the modeling patterns.
Example 3: The repairable components with one inflow and one outflow are commonly found in the production systems. The required `SISO_CorrectiveMaintenance` pattern is implemented using AltaRica 3.0 language, as it is depicted in Figure 4, which can be obtained by integrating `SISO` and `CorrectiveMaintenance` patterns. As it is shown in Figure 2 (2), the events `failure`, `startRepair`, and `endRepair` encode the transitions in Figure 4.

```
// State of the components
domain ComponentState {WORKING, FAILED, UNDER_REPAIR}

// CorrectiveMaintenance - Functional Modeling Pattern
class CorrectiveMaintenance
    ComponentState varState (init = WORKING);

    parameter Real Lambda = 0.001;//failure rate
    parameter Real Mu     = 0.1;  //repair  rate

    event failure     (delay = exponential(Lambda));
    event startRepair (delay = 0);
    event endRepair   (delay = exponential(Mu));
    transition
        failure:     varState == WORKING
                  -> varState := FAILED;
        startRepair: varState == FAILED
                  -> varState := UNDER_REPAIR;
        endRepair:   varState == UNDER_REPAIR
                  -> varState := WORKING;
end

// SISO - Physical Modeling Pattern
class SISO
    Real varInFlow  (reset = 0.0);
    Real varOutFlow (reset = 0.0);
end

// SISO_CorrectiveMaintenance - Modeling Pattern
class SISO_CorrectiveMaintenance

    // Integrate CorrectiveMaintenance
    extends CorrectiveMaintenance;
    // Integrate SISO
    extends SISO;
    assertion
        varOutFlow := if varState == WORKING
                      then varInFlow else 0.0;
end
```

Fig. 4 AltaRica 3.0 code of SISO_CorrectiveMaintenance.

MODELING PATTERNS RECOGNITION

By integrating the pattern recognition and the proposed modeling patterns, we can conduct the *modeling pattern recognition*, which is the automatic process of classifying the components of a targeted production system into different known modeling patterns. This work falls into the domain of *pattern recognition*.
The field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories (Bishop, 2006). Since the input components correspond to the targeted modeling patterns, the modeling pattern recognition is a *supervised learning* problem. The modeling patterns include a finite number of discrete categories, thus the modeling pattern recognition is also regarded as the modeling pattern classification.
We propose a methodology for modeling pattern recognition, as it is shown in Figure 5.

- **(1)** *Pre-processing*: In this step, the functional and physical behaviors of the given production system are identified. Take the SAFERELNET system as an example, the functional and physical modeling behaviors are listed in Figures 2 and 3.
- **(2)** *Feature selection*: *Features* are the characteristics which can describe the targeted systems. Let F be the set of these features, such that $f_1, ..., f_n$, where n is the total number of the features. $F = \{f_1, ..., f_n\} = \bigcup_{i=1}^{n} f_i$. In SAFERELNET system, $F = \begin{pmatrix} f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 & f_8 \end{pmatrix}$.

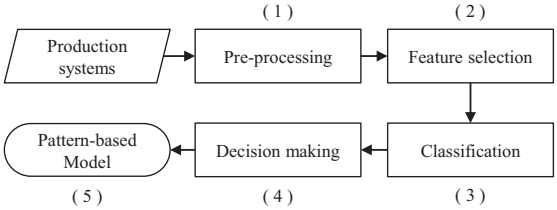


Fig. 5 Methodology for modeling pattern recognition.

- f_1 is the cardinality of the inflow set $\mathcal{F}_{in}$, that is, the number of members in the set.
- f_2 is the cardinality of the outflow set $\mathcal{F}_{out}$.
- f_3 is the cardinality of the set $\mathcal{S}$.
- f_4 is the cardinality of the transition set $\mathcal{T}$.
- f_5 is the cardinality of the demand set $\mathcal{D}$.
- f_6 is the cardinality of the capacity set C_{cap}.
- f_7 is the cardinality of the condition set C_{con}.
- f_8 is the discriminatory feature for modeling patterns. For instance, if we find the `DEGRADED` state in the component, it can be modeled at least with the `DEGRADATION` pattern.

The features that can reflect the inherent characteristics of the production system are selected. After reviewing the features, we found that f_6 and f_7 do not allow to identify the modeling patterns because the capacity and condition constraints are commonly found in the production systems. We select the rest features for classifying the components in the production systems. Here we obtain the selected feature vector $F' = \begin{pmatrix} f_1 & f_2 & f_3 & f_4 & f_5 & f_8 \end{pmatrix}$.

- **(3)** *Classification*: The components are classified into known modeling patterns.
- **(4)** *Decision making*: The modeling pattern is decided based on the classification results. The modeling patterns recognized in SAFERELNET system are listed in Table 1.
- **(5)** *Pattern-based Model*: The production system is modeled using the recognized modeling patterns.

Table 1 Modeling patterns recognized in SAFERELNET system.

Modeling patterns	Components
SOURCE_PERFECT	WEL
SOURCE_DEGRADATION_CorrectiveMaintenance _PreventiveMaintenance	TG1, TG2
SISO_PERFECT	OPS, WPS
SISO_DEGRADATION_CorrectiveMaintenance _PreventiveMaintenance	TC1, TC2
SISO_CorrectiveMaintenance_PreventiveMaintenance	EC
SIMO_CorrectiveMaintenance	TEG
MIMO_PERFECT	SEP

SIMULATION RESULTS

The SAFERELNET system can be represented using the proposed modeling patterns, as it is shown in Figure 6. The AltaRica model of this system is established using the proposed modeling patterns. The used modeling patterns are listed in Table 1. The simulation results are obtained from the stochastic (Monte Carlo) simulation of AltaRica model. The AltaRica model of SAFERELNET system is established using the proposed modeling patterns. The used modeling patterns are listed in

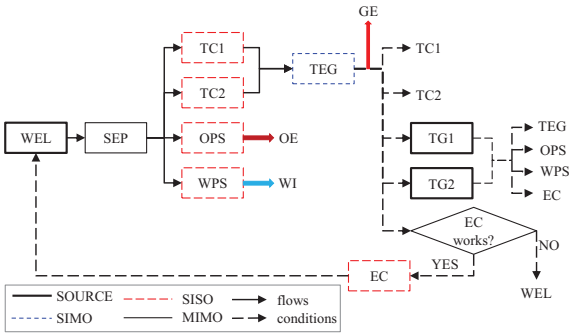


Fig. 6 Modeling patterns-based presentation of SAFERELNET system.

Table 1. The results are obtained from the stochastic simulation (Monte Carlo simulation) of AltaRica model. We conducted experiments with three cases. Case A is the system without preventive maintenance. Case B refers to the system as perfect without failure, but the system is operated with preventive maintenance. Case C reflects the real situation in which the system runs with both corrective and preventive maintenances. In the experiments, the simulation histories (Monte Carlo simulations) are 10^5. The mission time in cases A, B, C are 10^3h, 10^4h, 5×10^5h, respectively (Zio et al., 2006).

Table 2 shows the comparison of the production availabilities of SAFERELNET system with the results in (Zio et al., 2006). The results reveal that the production availabilities in cases A and B are higher than the ones in case C. The outcome is rational, which shares the same tendency as the results in (Zio et al., 2006). When the system components are perfect without failure (no downtime caused by the corrective maintenance), the production availability becomes higher. Thus the production availability in case A is higher than the ones in case C. Given that the components in the system run without preventive maintenance (no downtime generated by the preventive maintenance), the production availability becomes higher. Hence the production availability in case B is higher than the ones in case C.

Table 2 Production availabilities of SAFERELNET system.

Cases	Production availability	Gas	Oil	Water
A	Results in (Zio et al., 2006)	0.9726	0.9974	0.9577
	Our results	0.9570	0.9848	0.9564
B	Results in (Zio et al., 2006)	0.9763	0.9970	0.9730
	Our results	0.9711	0.9898	0.9758
C	Results in (Zio et al., 2006)	0.8858	0.9588	0.9573
	Our results	0.8860	0.9577	0.8875

CONCLUSION

We studied the production-performance issue with a viewpoint of how to reuse the capitalized knowledge. Many approaches have been utilized to assess the production performance, such as the analytical methods, Markov approach, Petri nets, Monte Carlo simulation, and AltaRica language. Currently, few studies have concerned about how to reuse the modeling experience of the production systems.

Our results demonstrate that it is beneficial to employ the capitalized knowledge. We propose the modeling patterns for performance analyses of the production systems. The modeling patterns are implemented with

the AltaRica 3.0 modeling language. An offshore production system is studied. This work indicates that the modeling patterns allow us to model the production systems in a modular way. The methodology in this paper could also serve as a hint for establishing the modeling patterns in other domains.

Since the proposed modeling patterns are based on a limited set of production systems, these modeling patterns can be improved to cover more system behaviors. The modeling patterns here cannot be applied to model the systems with chemical reactions. That is, we generally consider the upstream oil and gas plants, but not the systems like the oil refineries and petrochemical plants.

REFERENCES

Alexander, C (1997). "A pattern language: towns, buildings, construction", *Oxford University Press*.

Aven, T (1987). "Availability evaluation of oil/gas production and transportation systems", *Reliability Engineering*, Vol 18, pp 35–44.

Batteux, M, Prosvirnova, T, and Rauzy, A (2015). "AltaRica 3.0 language specification", *AltaRica Association*.

Bishop, CM (2006). "Pattern Recognition and Machine Learning", *Springer*.

Boiteau, M, Dutuit, Y, Rauzy, A, and Signoret, J-P (2006). "The AltaRica data-flow language in use: modeling of production availability of a multi-state system", *Reliability Engineering & System Safety*, Vol 91, pp 747–755.

Briš, R, and Kochaníčková, M (2006). "Stochastic Petri net approach to production availability evaluation of special test case", *Proceedings of the European Safety and Reliability Conference (ESREL 2006)*, pp 1569–1575.

Fritzson, P (2010). "Principles of object-oriented modeling and simulation with Modelica 2.1", *John Wiley & Sons*.

Gamma, E, Helm, R, Johnson, R, and Vlissides, J (1995). "Design patterns: elements of reusable object-oriented software", *Addison-Wesley Professional*.

Hamid, B, and Perez, J (2016). "Supporting pattern-based dependability engineering via model-driven development: Approach, tool-support and empirical validation", *Journal of Systems and Software*, Vol 122, pp 239–273.

Kawauchi, Y, and Rausand, M (2002). "A new approach to production regularity assessment in the oil and chemical industries", *Reliability Engineering & System Safety*, Vol 75, pp 379–388.

Kehren, C (2005). "Motifs formels d'architectures de systèmes pour la sûreté de fonctionnement", *Ecole nationale superieure de l'aeronautique et de l'espace, France*.

Khalil, M, Prieto, A, and Hölzl, F (2014). "A pattern-based approach towards the guided reuse of safety mechanisms in the automotive domain", *Model-Based Safety and Assessment: 4th International Symposium, IMBSA 2014*, pp 137–151.

Lipaczewski, M, Ortmeier, F, Prosvirnova, T, Rauzy, A, and Struck, S (2015). "Comparison of modeling formalisms for safety analyses: SAML and AltaRica", *Reliability Engineering & System Safety*, Vol 140, pp 191–199.

ISO (2008). "Petroleum, petrochemical and natural gas industries: Production assurance and reliability management", *ISO 20815*.

Meng, H, Kloul, L, and Rauzy, A (2015). "Production availability modelling of FPSO system using stochastic Petri nets", *Proceedings of the European Safety and Reliability Conference (ESREL 2015)*, pp 2271–2279.

Morel, M (2014). "Model-based safety approach for early validation of integrated and modular avionics architectures", *Model-Based Safety and Assessment: 4th International Symposium, IMBSA 2014*, pp 57–69.

NORSOK (1998). "Regularity management & Reliability technology", *NORSOK Z-016*.

Prosvirnova, T, Batteux, M, Brameret, P-A, Cherfi, A, Friedlhuber, T, Roussel, J-M, and Rauzy, A (2013). "The AltaRica 3.0 project for model-based safety assessment", *Proceedings of 4th IFAC Workshop on Dependable Control of Discrete Systems*, pp 127–132.

Prosvirnova, T (2014). "Altarica 3.0: a model-based approach for safety analyses", *Ecole Polytechnique, France*.

Rauzy, A (2004). "An experimental study on iterative methods to compute transient solutions of large markov models", *Reliability Engineering & System Safety*, Vol 86, pp 105–115.

Rauzy, A (2008). "Guarded transition systems: a new states/events formalism for reliability studies", *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, Vol 222, pp 495–505.

Rosen, K (2011). "Discrete Mathematics and Its Applications 7th edition", *McGraw-Hill Science*.

Signoret, J-P (2000). "SAFERELNET–Production Availability Test Case Version 1", *Document of TOTAL–DGEP*.

Signoret, J-P (2010). "Production availability, in: C. Guedes Soares (Ed.), Safety and Reliability of Industrial Products, Systems and Structures", *CRC Press*.

Vesteraas, AH (2008). "Comparison of methods and software tools for availability assessment of production systems", *Norwegian University of Science and Technology, Norway*.

Zhang, H, Innal, F, Dufour, F, and Dutuit, Y (2014). "Piecewise deterministic Markov processes based approach applied to an offshore oil production system", *Reliability Engineering & System Safety*, Vol 126, pp 126–134.

Zio, E, Baraldi, P, and Patelli, E (2006). "Assessment of the availability of an offshore installation by Monte Carlo simulation", *International Journal of Pressure Vessels and Piping*, Vol 83, pp 312–320.

Appendix C

Modeling Patterns for Reliability Analyses of Safety Systems

Safety Instrumented Systems (SIS) act as important safety barriers in industrial systems for preventing hazardous accidents. It is therefore significant to study the reliability of SIS, which have been investigated extensively. As a matter of fact, SIS have common behaviors such as periodic test policies to discover dangerous undetected failures. Modeling patterns capitalize the experience from modeling SIS. By reusing modeling patterns, modeling mission can be simplified when assessing the reliability and availability of systems. Few studies related to SIS have been conducted on patterns for reliability assessment. In this chapter, we show our work of pattern-based reliability analyses of safety systems[1].

This work proposes a pattern-based methodology for reliability assessment of SIS. To demonstrate its applicability, the proposed methodology is applied on an emergency depressurization system provided in an ISO technical report (ISO/TR 12489). The comparison is performed between results obtained using given modeling patterns and the ones from ISO/TR 12489. It is shown that the pattern-based methodology can serve as an effective tool for modeling SIS in a modular way. In short, highlights of this study are:

- A pattern-based methodology for reliability assessment of SIS is put forward.
- A set of modeling patterns for reliability assessment of SIS is proposed.
- Proposed methodology is applied on an emergency depressurization system.

[1] H. Meng, L. Kloul, A. Rauzy. A pattern-based methodology for reliability assessment of safety instrumented systems, Proceedings of the 2017 IEEE International Symposium on Systems Engineering (**ISSE 2017**), IEEE Systems Council, Vienna, Austria, Oct. 2017: 438-443.

A Pattern-based Methodology for Reliability Assessment of Safety Instrumented Systems

Huixing Meng
LIX
École Polytechnique
Palaiseau, France
huixing.meng@polytechnique.edu

Leïla Kloul
DAVID
Université de Versailles
St-Quentin-en-Yvelines
leila.kloul@uvsq.fr

Antoine Rauzy
MTP
Norwegian University of
Science and Technology
antoine.rauzy@ntnu.no

***Abstract*—Safety Instrumented Systems (SIS) act as important safety barriers in industrial systems for preventing hazardous accidents. It is therefore significant to study the reliability issues of SIS. As a matter of fact, SIS have common behaviors such as periodic test policies to discover dangerous undetected failures. Modeling patterns capitalize the experience from modeling SIS. By reusing modeling patterns, modeling mission can be simplified when assessing reliability and availability of systems. Few studies related to SIS have been conducted on patterns for reliability assessment. This paper proposes a pattern-based methodology for reliability assessment of SIS. To demonstrate the applicability, the proposed methodology is applied on an emergency depressurization system provided in an ISO technical report (ISO/TR 12489). The comparison is performed between results obtained using given modeling patterns and the ones from ISO/TR 12489. It is shown that the pattern-based methodology can serve as an effective tool for modeling SIS in a modular way.**

I. Introduction

Safety Instrumented Systems (SIS) play an important role in industrial plants for preventing hazardous accidents. These systems are composed of sensors (e.g. pressure sensors), logic solvers (e.g. programmable logic controllers), and final elements (e.g. isolation valves). Logic solvers translate signals transmitted from sensors into decisions made on final elements. SIS have attracted a lot of attention from various industrial sectors. Associated standards are proposed in specific industries, such as the process industry, the nuclear power industry, the machinery industry, the automotive systems, as well as the railway systems. The main standard is IEC 61508 [1]. Sound performance of SIS is crucial for Equipments Under Control (EUC). It is therefore significant to study the reliability issues of SIS.

Reliability studies of SIS have been conducted tremendously (see e.g. [2]–[5]). Many aspects related to SIS have been investigated, including proof tests, k-out-of-n voting structures, common cause failures, spurious failures, human and organizational factors, uncertainty, and optimization issues.

Models and modeling experience are expected to be capitalized, otherwise, the modeling activity is hardly profitable. Patterns can be utilized for reusing the stabilized knowledge. Reliability studies can benefit from reusing modeling patterns. However, few studies have been carried out on modeling patterns for reliability assessment of SIS.

The pattern was initially proposed in civil engineering [6]. The concept was adopted in software engineering subsequently as the design pattern [7]. This one promotes design reuse, conforms to a literary style, and defines a vocabulary for discussing design [8].

A modeling pattern is a general means allowing to capture the frequently recurrent component and subsystem behaviors in industrial systems. Some researchers try to provide a general framework of reusing patterns. The pattern based system engineering was proposed [9], whose procedure includes the pattern definition and the system development with patterns [10]. The reuse of systems and subsystems is a common practice in safety-critical systems engineering [11]. To reuse system behaviors, we need to standardize the representation of reusable components and figure out the way they exchange information [12]. The whole point of a pattern is to reuse, rather than to reinvent [8].

An advantage of high-level modeling languages, like AltaRica [13], [14], is to reuse models of components or even systems [13]. The AltaRica modeling language is especially well suited for safety analyses [13], [14]. The AltaRica language is introduced in IEC 61508 as a technique for calculating the probabilities of hardware failures in SIS [1]. The language is also mentioned in ISO/TR 12489 [2].

To reuse modeling patterns, a methodology is the prerequisite when assessing reliabilities of SIS. Two benefits are expected with such a particular procedure: first, one can follow steps to analyze the reliability of a SIS via modeling patterns; second, people can propose their own pattern-based methodology in a similar way.

In this article, we develop a pattern-based methodology for reliability assessment of SIS. We classify modeling patterns into different categories. Proposed modeling patterns are implemented with the AltaRica 3.0 modeling language. The methodology is tested with a SIS in ISO/TR 12489.

The rest of this paper is organized as follows. Section II reviews related works. Section III is dedicated to modeling patterns extracted from SIS in ISO/TR 12489. Section IV develops a pattern-based methodology for reliability assessment of SIS. Section V studies an emergency depressurization system to illustrate the application of the proposed methodology. Finally, Section VI concludes this work.

II. Related works

Patterns have been discussed in reliability and safety domains [15]. Patterns related to accident analyses are investigated in traffic domain [16] and industrial plants [17], whereas these studies employ statistical methods to discover patterns of accident causes. The dependability pattern is proposed in [10]. It is defined as the description of a particular recurring dependability problem that arises in specific contexts and presents a well-proven generic scheme for its solution. Resilience design patterns are raised to meet the demand of extreme-scale high-performance computing systems [18].

From the modeling experience of several aircraft systems using AltaRica Data-Flow language, Safety Architecture Patterns (SAP) are proposed to simplify modeling missions [19]. SAP are component assemblies used to ensure the safety of architectures [19]. The application of SAP can be found in the avionics domain [19], [20]. Unlike their work [19], first, we use the AltaRica 3.0 language, which has a different mathematical foundation. The mathematical backgrounds of AltaRica Data-Flow and AltaRica 3.0 are mode automata [21] and guarded transition systems [22], separately. Second, we propose patterns for modeling SIS aiming in process industry. However, their work primarily locates in aviation industry. Third, they mainly proposed the structured collection of redundancy based architecture patterns. But we try to describe behavioral, flow propagation, and coordination characteristics of SIS with modeling patterns.

In a recent work [23], we propose the modeling patterns for production-performance analyses. We apply proposed modeling patterns on a practical offshore installation. The two sets of modeling patterns (in [23] and this article) share some patterns, that is, CorrectiveMaintenance, SERIES, PARALLEL, and KooN (k-out-of-n). However, most of patterns are different, which include patterns for performance analyses of production systems and patterns for reliability assessment of SIS.

Few studies related to patterns of SIS have been conducted. Related works can be found in [2], [24], where the Reliability Block Diagram (RBD) driven Petri Nets (PN) are proposed for reliability analyses. The readability of PN is improved by means of RBD.

III. Modeling patterns

A Modeling Pattern (MP) is a general means allowing to capture the frequently recurrent component and subsystem behaviors in process industry. Modeling patterns can be classified according to their purpose, which reflects what a modeling pattern works for. They can have either behavioral, flow propagation, and coordination purpose. Behavioral Patterns (BP) describe the basic behaviors of a component. For instance, the repairable behavior is regarded as a basic character in SIS. Flow Propagation Patterns (FPP) depict the flow propagations inside or between components. Coordination Patterns (CP) represent cooperations or synchronizations in a system, such as repairable units and repair crews.

We choose SIS in ISO/TR 12489 as our running examples. This is because these architectures are general enough to cover most of safety systems [2]. In addition, these systems are representative of most of reliability studies of safety systems performed in petroleum, petrochemical, and natural gas industries, as well as in other industries [2].

A. Behavioral Patterns

In this part, we introduce five BP, as shown in Figure 1, which mainly capture shared component behaviors.

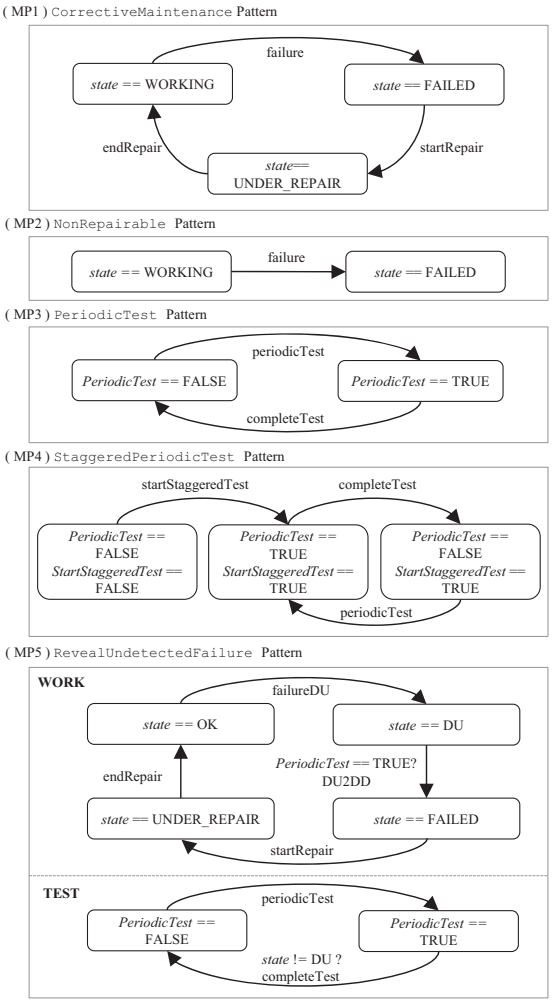


Fig. 1. Behavioral Patterns.

(1) `CorrectiveMaintenance` pattern (Figure 1, MP1): it models components which can be repaired after failure. The component is initially working (*state* == `OK`). Once a failure occurs, the component falls into `FAILED` state. When the corrective maintenance team is available, the component state becomes `UNDER_REPAIR`. Finally, the component returns to the initial state once the repair operation is finished. This pattern is used to model Dangerous Detected (DD) failures. DD failures are detected a short time after their occurrence by automatic diagnostic testing [3].

(2) NonRepairable pattern (Figure 1, MP2): it models components which cannot be repaired after failure. The component is initially in OK state. Once a failure occurs, the component becomes FAILED. In SIS, Dangerous Undetected (DU) failures are preventing activation on demand and can be revealed only by periodic tests (i.e., proof tests) [3]. Part of DU failures cannot be covered by imperfect periodic tests (i.e., the proof test coverage $< 100\%$), such uncovered DU failures can be modeled using this pattern. The rest part of DU failures are covered by periodic tests, which are modeled by means of the following pattern.

(3) PeriodicTest pattern (Figure 1, MP3): it models the periodic test which can detect DU failures. Periodic tests are conducted at predefined intervals and durations.

(4) StaggeredPeriodicTest pattern (Figure 1, MP4): it models the staggered periodic test, which is thought to allow obtaining higher availability than simultaneous tests. Compared with a reference periodic test, the duration of the first test interval in the staggered periodic test is different from the duration of following test intervals. Initially, the *startStaggeredTest* is triggered. Subsequently, the rest of the pattern architecture becomes similar to the PeriodicTest pattern.

(5) RevealUndetectedFailure pattern (Figure 1, MP5): it models the process to detect DU failures and is based on the CorrectiveMaintenance and PeriodicTest patterns. Three issues of this pattern deserve to be underlined: first, DU failures can only be discovered once the *PeriodicTest* is true; second, the periodic test can be completed only after DU failures are detected; third, revealed DU failure works following the CorrectiveMaintenance pattern.

B. Flow Propagation Patterns

Flow Propagation Patterns (FPP) depict flow propagations inside and between components. In the following, we illustrate five FPP, as shown in Figure 2.

(6) SERIES pattern (Figure 2, MP6) describes the series structure, which models series connection of several basic patterns. The average unavailability of the SERIES pattern $\bar{U}_{\text{SERIES}}$ is:

$$\bar{U}_{\text{SERIES}} = 1 - (1 - \bar{u}_1)(1 - \bar{u}_2)\cdots(1 - \bar{u}_n) \tag{1}$$

where $\bar{u}_1$, $\bar{u}_2$, $\cdots$, $\bar{u}_n$ are average unavailabilities of components C_1, C_2, $\cdots$, C_n, respectively.

(7) PARALLEL pattern (Figure 2, MP7) depicts the parallel structures. It models the parallel connection of several SERIES patterns. The average unavailability of the PARALLEL pattern $\bar{U}_{\text{PARALLEL}}$ is:

$$\bar{U}_{\text{PARALLEL}} = \bar{u}_1\bar{u}_2\cdots\bar{u}_n \tag{2}$$

where $\bar{u}_1$, $\bar{u}_2$, $\cdots$, $\bar{u}_n$ are average unavailabilities of components C_1, C_2, $\cdots$, C_n, respectively.

(8) KooN (k-out-of-n: G) pattern (Figure 2, MP8) describes the structure which works when at least k of the total number n of items must be functioning. The average unavailability of the KooN pattern $\bar{U}_{\text{KooN}}$ is:

$$\bar{U}_{\text{KooN}} = 1 - \sum_{x=k}^{n} \binom{n}{x} (1 - \bar{u})^x \bar{u}^{n-x} \tag{3}$$

where components in KooN are usually identical, and $\bar{u}$ is the average unavailability of each component. Some typical configurations of KooN structure are 1oo1 (i.e. single item), 1oo2, 2oo2, and 2oo3 [1].

(9) SwitchKooN pattern (Figure 2, MP9) depicts the behavior of switching a KooN structure into (K-1)-out-of-(N-1) structure when a DD or DU failure occurs. Once the

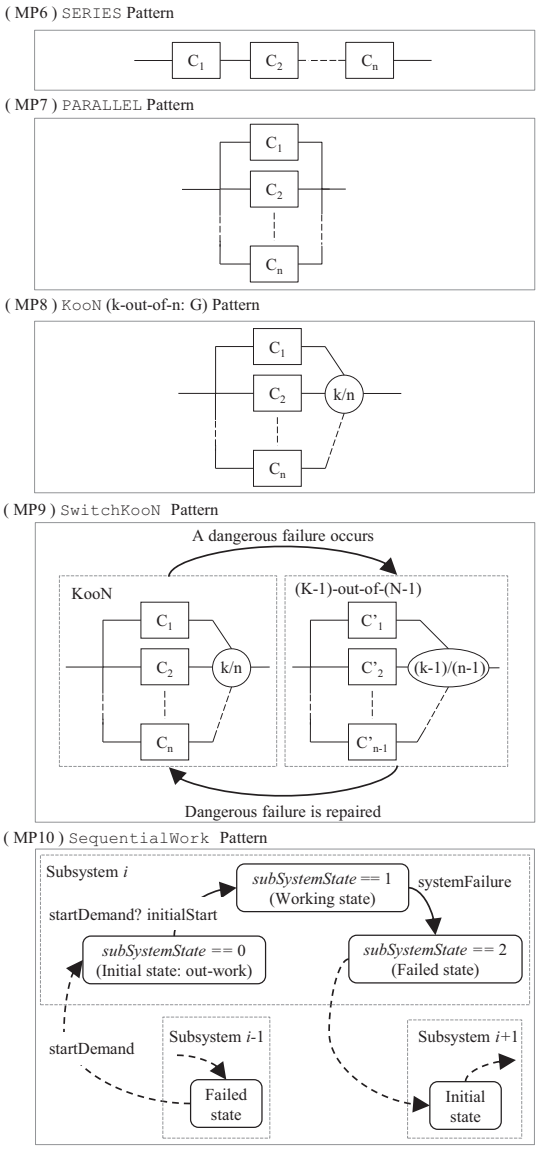


Fig. 2. Flow Propagation Patterns.

failure is repaired, the structure is normally restored to KooN structure.

The switched configuration, (K-1)-out-of-(N-1), can increase the system availability. If there is no such a switch, the structure is supposed to work as a K-out-of-(N-1) structure after a failure. According to Equation (3),

$$\bar{U}_{\text{(K-1)-out-of-(N-1)}} - \bar{U}_{\text{K-out-of-(N-1)}} = -\binom{n-1}{k-1}(1-\bar{u})^{k-1}\bar{u}^{n-k} \tag{4}$$

where $\bar{U}_{\text{(K-1)-out-of-(N-1)}}$ and $\bar{U}_{\text{K-out-of-(N-1)}}$ are the unavailabilities of (K-1)-out-of-(N-1) and K-out-of-(N-1) structures, respectively.

Since $-\binom{n-1}{k-1}(1-\bar{u})^{k-1}\bar{u}^{n-k}$ is a negative number, $\bar{U}_{\text{(K-1)-out-of-(N-1)}} < \bar{U}_{\text{K-out-of-(N-1)}}$. That is, the availability of the (K-1)-out-of-(N-1) structure increases after switch, when compared with the K-out-of-(N-1) structure.

Typically, if a dangerous failure (DD or DU) occurs in a 2oo3 structure, the logic solver changes the policy from 2oo3 to 1oo2.

(10) `SequentialWork` pattern (Figure 2, MP10) depicts the multiple SIS which work in a sequential order.The failed state of the previous subsystem $i-1$ triggers the successive subsystem i. This one is initially out of work (*subSystemState* == 0). If the trigger action (startDemand) from subsystem $i-1$ is perfect, the subsystem i becomes working (*subSystemState* == 1). If the subsystem i fails (*subSystemState* == 2), it can trigger the working of subsystem $i+1$. Note that if the trigger action is perfect, `SequentialWork` is equivalent to the `PARALLEL` pattern.

C. Coordination Patterns

Coordination Patterns (CP) represent cooperations or synchronizations in a system.

(11) `Repairable unit/Repair crew Coordination` pattern (Figure 3): it models limited repair crews in SIS. The working state of the repair crew (*RepairCrewWork*) is FALSE initially. If the number of busy repair crews (*numberBusyCrew*) is smaller than the total number of repair crews (*totalNumberCrew*) and a repair is required by a repairable unit, the repair is started. Simultaneously, 1 is added to *numberBusyCrew*. Adversely, 1 is decreased to *numberBusyCrew* when a repair is completed.

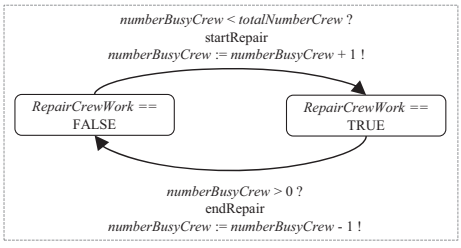


Fig. 3. Modeling pattern (MP11): `Repairable unit/ Repair crew Coordination`.

IV. Proposed Methodology

Figure 4 describes the methodology to model SIS using modeling patterns. This methodology is composed of four steps: classification, pattern-based model, AltaRica 3.0 model, and experimental results. We take typical application (TA) 1-1 in ISO/TR 12489 [2] as an example to illustrate this methodology. As a basic architecture of SIS, TA 1-1 is formed by a pressure sensor, a logic solver, and an isolation valve working in series.

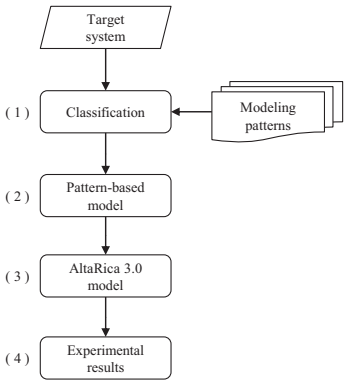


Fig. 4. Pattern-based methodology for reliability assessment of safety instrumented systems.

(1) *Classification*: In this step, we identify units to be modeled and recognize corresponding modeling patterns. The target system is initially decomposed into components and subsystems. We identify modeling patterns that are required to construct these components and subsystems.

Two components are modeled in TA 1-1, where the protected system is shut down during periodic tests and repairs. Thus the activities related to the maintenance/repair are negligible when calculating the system unavailability. Since the system unavailability of TA 1-1 is only generated by DU failures, thus the logical solver (which only has DD failures) has not been considered. The pressure sensor and isolation valve are modeled by the `RevealUndetectedFailure` pattern. These two components work in series, thus SERIES pattern is used as well.

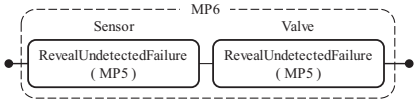


Fig. 5. Pattern-based model of the typical application 1-1 in ISO/TR 12489.

(2) *Pattern-based model*: Based on classification results, a pattern-based model can be obtained. This model is illustrated with the form of schematic diagrams. That is, we use such diagrams to exhibit classification results. The pattern-based model of TA 1-1 is shown in Figure 5. The pattern-based model is prepared for constructing the concrete model with a modeling language.

(3) *AltaRica 3.0 model*: AltaRica 3.0 modeling language is employed to model safety instrumented systems. The pattern-based model can be implemented with AltaRica 3.0.

(4) *Experimental results*: The obtained AltaRica 3.0 model is firstly translated and flattened into a GTS (Guarded Transition Systems) model. Subsequently, experimental results are acquired by analyzing the GTS model with the stochastic simulator [25], [26].

V. CASE STUDY

In order to validate the solidity of the proposed approach, we have modeled all SIS (13 systems with different architectures and assumptions) in ISO/TR 12489 with the proposed methodology and modeling patterns. The results applying proposed approach agree rather well with those from ISO/TR 12489. Because of the limited length of the paper, we consider an Emergency DePressurization (EDP) system (Figure 6) of a hydrocracking unit in ISO/TR 12489 as a case study.

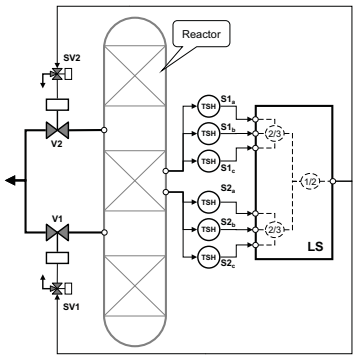


Fig. 6. An emergency depressurization (EDP) system [2].

The EDP system is composed of two groups of temperature sensors (S1a, S1b, and S1c; S2a, S2b, and S2c) organized in 2oo3, one Logic Solver (LS) and two corresponding isolation Valve (V1 and V2) in parallel and piloted by two corresponding Solenoid Valves (SV1 and SV2). This safety system aims to quickly depressurize the reactor when the temperature increases and reaches a predetermined threshold, thus to avoid a runaway of the exothermic chemical reaction.

The assumptions used for the EDP system are:

- DD and DU failures of a given component are independent.
- Constant failure rates are assumed.
- Components are as good as new after repairs.
- Periodic tests are performed when the reactor is stopped.
- Installation is paused during repair of DU failures.
- Installation is shut down during periodic tests and repair of the logic solver.
- Failures that are not covered by periodic tests will not be detected and repaired.
- The 2oo3 logic of a group of sensors is switched to 1oo2 in case of one dangerous detected failure in the group.

In the following, we illustrate how to assess the reliability of the EDP system when applying the pattern-based methodology in Figure 4.

(1) *Classification*: We identify the modeling patterns matching components and subsystems in EDP system. Modeling patterns classification for EDP system is provided in Table I.

TABLE I
MODELING PATTERNS CLASSIFICATION FOR EDP SYSTEM.

Components/Subsystems	Modeling patterns
• S1a, S1b, S1c, S2a, S2b, S2c	MP1, MP2, MP5
• {S1a, S1b, S1c},{S2a, S2b, S2c}	MP8, MP9
• {S1, S2}	MP7
• LS, SV1, V1, SV2, V2	MP2, MP5
• {SV1, V1},{SV2, V2}	MP6

We employ a 2oo3 structure (S1a, S1b, and S1c) as an example to elaborate the results. Since DD and DU failures of a component are assumed to be independent in EDP system, the DD failure of a component (e.g., S1a) can be modeled using `CorrectiveMaintenance` pattern (MP1). Since the uncovered DU failure cannot be repaired, it is constructed with the `NonRepairable` pattern (MP2). The covered DU failure by periodic tests is considered with the `RevealUndetectedFailure` pattern (MP5). The subsystem composed by these three components, {S1a, S1b, S1c}, can be modeled by both `KooN` pattern (MP8) and `SwitchKooN` pattern (MP9). The two groups of 2oo3 structures, {S1, S2}, in the EDP system can be modeled with `PARALLEL` pattern (MP7). Note that S1 and S2 stand for the 2oo3 subsystems. The rest of classification results can be interpreted in the similar way.

(2) *Pattern-based model*: On the basis of results in Table I, we establish the pattern-based model of EDP system, as it is shown in Figure 7. Associated modeling patterns are assigned for each component/subsystem in the diagram. The pattern-based model simplifies the task of constructing the AltaRica 3.0 model.

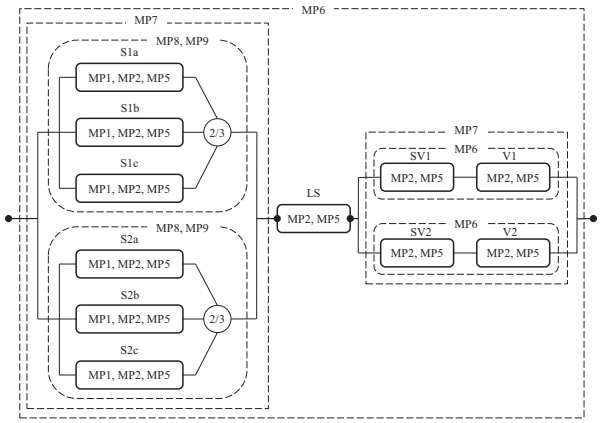


Fig. 7. Pattern-based model of the EDP system.

(3) *AltaRica 3.0 model*: In this step, we translate the pattern-based model of the EDP system into the corresponding AltaRica 3.0 model. Modeling patterns are firstly presented in the AltaRica environment. Subsequently, the AltaRica 3.0 model of the EDP system is constructed with identified modeling patterns.

(4) *Experimental results*: The mission time (length of histories) of this simulation experiment is $131,400$ h (15 years). The number of the Monte Carlo simulations (number of histories) is 10^6. The results comparison can be found in Table II. The AltaRica 3.0 (Stochastic simulator) and ISO/TR 12489 (Fault tree) give almost the same results, where the percentage difference is 1.14%.

TABLE II
EXPERIMENTAL RESULTS OF THE EDP SYSTEM.

Approaches	Average unavailability
Fault tree [2]	3.50E-4
AltaRica 3.0	3.46E-4

VI. CONCLUSION

This paper has presented a pattern-based methodology for reliability assessment of Safety Instrumented Systems (SIS). First, based on a series of SIS provided in ISO/TR 12489, a set of modeling patterns is proposed. Modeling patterns are categorized into behavioral patterns, flow propagation patterns, and coordination patterns. Second, a pattern-based methodology is put forward and illustrated with a simplified SIS. Eventually, the proposed methodology is tested with a complex SIS in ISO/TR 12489. The corresponding AltaRica model has been developed to assess the system reliability. The result obtained from AltaRica model using modeling patterns is in good agreement with that from ISO/TR 12489. It is concluded that the proposed methodology is capable of constructing targeted systems in a modular way.

Raised modeling patterns are based on a limited set of SIS. Even if they are declared to cover most of reliability studies [2], these patterns can be improved with new system behaviors. The current paper is restricted to study SIS and EUC (Equipments Under Control) separately. Further research may consider EUC as an integral part of SIS. New modeling patterns are therefore expected with such integrations.

REFERENCES

[1] IEC, "IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems," 2010.

[2] ISO, "ISO/TR 12489 Petroleum, petrochemical and natural gas industries: Reliability modelling and calculation of safety systems," 2013.

[3] M. Rausand, *Reliability of Safety-Critical Systems: Theory and Applications*. John Wiley & Sons, 2014.

[4] M. Rausand and A. Høyland, *System reliability theory: models, statistical methods, and applications*. John Wiley & Sons, 2004.

[5] Y. Dutuit, A. Rauzy, and J.-P. Signoret, "A snapshot of methods and tools to assess safety integrity levels of high-integrity protection systems," *Proc. Inst. Mech. Eng. Part O-J. Risk Reliab.*, vol. 222, pp. 371–379, 2008.

[6] C. Alexander, *A pattern language: towns, buildings, construction*. Oxford University Press, 1977.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[8] E. Gamma, "Design patterns−ten years later," in *Software pioneers*. Springer, 2002, pp. 688–700.

[9] D. Cook and W. D. Schindel, "Utilizing MBSE patterns to accelerate system verification," *INSIGHT*, vol. 20, no. 1, pp. 32–41, 2017.

[10] B. Hamid and J. Perez, "Supporting pattern-based dependability engineering via model-driven development: Approach, tool-support and empirical validation," *J. Syst. Softw.*, vol. 122, pp. 239–273, 2016.

[11] A. Ruiz, G. Juez, H. Espinoza, J. L. de la Vara, and X. Larrucea, "Reuse of safety certification artefacts across standards and domains: A systematic approach," *Reliab. Eng. Syst. Saf.*, vol. 158, pp. 153–171, 2017.

[12] N. Kajtazovic, C. Preschern, A. Höller, and C. Kreiner, "Towards pattern-based reuse in safety-critical systems," in *EuroPLoP*. , 2014.

[13] T. Prosvirnova, "Altarica 3.0: a model-based approach for safety analyses," Ph.D. dissertation, Ecole Polytechnique, Palaiseau, France, 2014.

[14] T. Prosvirnova, M. Batteux, P.-A. Brameret, A. Cherfi, T. Friedlhuber, J.-M. Roussel, and A. Rauzy, "The AltaRica 3.0 project for model-based safety assessment," in *Proceedings of 4th IFAC Workshop on Dependable Control of Discrete Systems*, 2013, pp. 127–132.

[15] C. Preschern, N. Kajtazovic, A. Höller, and C. Kreiner, "Pattern-based safety development methods: overview and comparison," in *EuroPLoP*. , 2014.

[16] Q. Guo, P. Xu, X. Pei, S. Wong, and D. Yao, "The effect of road network patterns on pedestrian safety: a zone-based bayesian spatial modeling approach," *Accid. Anal. Prev.*, vol. 99, pp. 114–124, 2017.

[17] A. Verma, S. D. Khan, J. Maiti, and O. Krishna, "Identifying patterns of safety related incidents in a steel plant using association rule mining of incident investigation reports," *Saf. Sci.*, vol. 70, pp. 89–98, 2014.

[18] S. Hukerikar and C. Engelmann, "Resilience design patterns - a structured approach to resilience at extreme scale," Oak Ridge National Laboratory, Tennessee, US, Tech. Rep., 2016.

[19] C. Kehren, "Motifs formels d'architectures de systèmes pour la sûreté de fonctionnement," Ph.D. dissertation, Ecole nationale superieure de l'aeronautique et de l'espace, Toulouse, France, 2005.

[20] M. Morel, "Model-based safety approach for early validation of integrated and modular avionics architectures," in *IMBSA, Munich, October, 2014*. LNCS, 2014, pp. 57–69.

[21] A. Rauzy, "Mode automata and their compilation into fault trees," *Reliab. Eng. Syst. Saf.*, vol. 78, pp. 1–12, 2002.

[22] A. Rauzy, "Guarded transition systems: a new states/events formalism for reliability studies," *Proc. Inst. Mech. Eng. Part O-J. Risk Reliab.*, vol. 222, no. 4, pp. 495–505, 2008.

[23] H. Meng, L. Kloul, and A. Rauzy, "Modeling patterns for performance analyses of offshore production systems," in *The Proceedings of the 27th International Ocean and Polar Engineering Conference*. San Francisco: ISOPE, June 2017.

[24] J.-P. Signoret, Y. Dutuit, P.-J. Cacheux, C. Folleau, S. Collas, and P. Thomas, "Make your Petri nets understandable: Reliability block diagrams driven Petri nets," *Reliab. Eng. Syst. Saf.*, vol. 113, pp. 61–75, 2013.

[25] B. Aupetit, M. Batteux, A. Rauzy, and J.-M. Roussel, "Improving performances of the AltaRica 3.0 stochastic simulator," in *ESREL*, Zurich, Switzerland, 2015.

[26] M. Batteux and A. Rauzy, "Stochastic simulation of altarica 3.0 models," in *ESREL*, Amsterdam, Netherlands, 2013.

Appendix D

Acronyms and Abbreviations

BNF Backus-Naur form

BPCS Basic process control system

CCA Cause-consequence analysis

CCF Common cause failures

CM Corrective maintenance

CTMC Continuous-time Markov chains

DAG Directed acyclic graph

DCG Directed cyclic graph

DD Dangerous detected failure

DU Dangerous undetected failure

EDP Emergency depressurization system

ESD Emergency shut down system

ET Event tree

EUC Equipment under control

FMEA Failure mode and effects analysis

FPSO Floating production, storage, and offloading unit

FT Fault tree

GRIF GRaphical Interface for reliability Forecasting (Total)

GSPN Generalized stochastic Petri nets

GTS Guarded transition systems

HIPPS High integrity pressure protection system

IEC International Electrotechnical Commission

ISO International Organization for Standardization

MBSA Model-based safety analysis/assessment

MBSE Model-based system engineering

MC Markov chain

MP Modeling patterns

OREDA Offshore and Onshore Reliability Data

PAP Production assurance program

PBSE Pattern based system engineering

PFD Probability of failure on demand

PFH Probability of failure per hour

PM Preventive maintenance

PN Petri nets

PRA Probabilistic risk assessment

RAMS Reliability, Availability, Maintainability and Safety

RBD Reliability block diagrams

SAP Safety architecture patterns

SIF Safety instrumented function

SIL Safety integrity level

SIS Safety instrumented system

SM2 Second order safety mechanisms

SPN Stochastic Petri nets

Taro Total Asset Review and Optimization software (DNV GL)

MAROS Maintainability, Availability, Reliability, Operability Simulation software (DNV GL)

Bibliography

[1] Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 7(1):39–59.

[2] Alexander, C. (1977). *A pattern language: towns, buildings, construction*. Oxford University Press.

[3] Arnold, A., Point, G., Griffault, A., and Rauzy, A. (2000). The AltaRica formalism for describing concurrent systems. *Fundam. Inform.*, 34:109–124.

[4] Aven, T. (1987). Availability evaluation of oil/gas production and transportation systems. *Reliability Engineering*, 18(1):35–44.

[5] Aven, T. (1989). Availability evaluation of flow networks with varying throughput-demand and deferred repair. *IEEE transactions on reliability*, 38(4):499–505.

[6] Aven, T. and Pedersen, L. M. (2014). On how to understand and present the uncertainties in production assurance analyses, with a case study related to a subsea production system. *Reliability Engineering & System Safety*, 124:165–170.

[7] Barabady, J. and Aven, T. (2008). A methodology for the implementation of production assurance programmes in production plants. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 222:283–290.

[8] Barabady, J., Markeset, T., and Kumar, U. (2010). Review and discussion of production assurance program. *International Journal of Quality & Reliability Management*, 27(6):702–720.

[9] Batteux, M., Prosvirnova, T., and Rauzy, A. (2015). AltaRica 3.0 language specification. Technical report, AltaRica Association.

[10] Batteux, M. and Rauzy, A. (2013). Stochastic simulation of AltaRica 3.0 models. In *Proceedings of the European Safety and Reliability Conference, ESREL*, pages 1093–1100.

[11] Boiteau, M., Dutuit, Y., Rauzy, A., and Signoret, J.-P. (2006). The AltaRica data-flow language in use: modeling of production availability of a multi-state system. *Reliability Engineering & System Safety*, 91(7):747–755.

[12] Borgonovo, E., Marseguerra, M., and Zio, E. (2000). A Monte Carlo methodological approach to plant availability modeling with maintenance, aging and obsolescence. *Reliability Engineering and System Safety*, 67:61–73.

[13] Bouissou, M. (2005). Automated dependability analysis of complex systems with the KB3 workbench: the experience of EDF R&D. In *Proceedings of the International Conference on ENERGY and ENVIRONMENT, CIEM 2005*.

[14] Bouissou, M., Villatte, N., Bouhadana, H., and Bannelier, M. (1991). Knowledge modelling and reliability processing: presentation of the FIGARO language and associated tools. In *Safety, Security, and Reliability of Computer Based Systems : Proceedings of the IFAC/IFIP/EWICS/SRE Symposium*.

[15] Bozzano, M., Cimatti, A., Lisagor, O., Mattarei, C., Mover, S., Roveri, M., and Tonetta, S. (2015). Safety assessment of AltaRica models via symbolic model checking. *Science of Computer Programming*, 98:464–483.

[16] Breipohl, A. M., Lee, F. N., and Chiang, J.-Y. (1994). Stochastic production cost simulation. *Reliability Engineering and System Safety*, 46(1):101–107.

[17] Briš, R. and Kochaníčková, M. (2006). Stochastic Petri net approach to production availability evaluation of special test case. *Safety and Reliability for Managing Risk*, 2:1569–1575.

[18] Brissaud, F., Charpentier, D., Fouladirad, M., Barros, A., and Bérenguer, C. (2008). Safety instrumented system reliability evaluation with influencing factors. In *ESREL 2008*, pages 2003–2011, Valencia, Spain. CRC Press, Taylor & Francis Groups.

[19] Brissaud, F., Charpentier, D., Fouladirad, M., Barros, A., and Bérenguer, C. (2010). Failure rate evaluation with influencing factors. *Journal of Loss Prevention in the Process Industries*, 23(2):187–193.

[20] Buvry, P., Brissaud, F., Declerck, B., and Varela, H. (2015). Comparison of two tools for production availability analyses: MAROS and GRIF/BStok, arxiv e-prints.

[21] Cacheux, P.-J., Collas, S., Dutuit, Y., Folleau, C., Signoret, J.-P., and Thomas, P. (2013). Assessment of the expected number and frequency of failures of periodically tested systems. *Reliability Engineering & System Safety*, 118:61–70.

[22] Chang, K. P., Chang, D., and Zio, E. (2010). Application of Monte Carlo simulation for the estimation of production availability in offshore installations. In *Simulation Methods for Reliability and Availability of Complex Systems*, pages 233–252. Springer.

[23] Cherfi, A. (2015). *Toward an Efficient Generation of ISO 26262 Automotive Safety Analyses*. PhD thesis, Ecole Polytechnique, Palaiseau, France.

[24] Cook, D. and Schindel, W. D. (2017). Utilizing mbse patterns to accelerate system verification. *INSIGHT*, 20(1):32–41.

[25] Dejean, J., Averbuch, D., Maurel, P., Gainville, M., Guet, S., and Johnsen, Ø. (2007). FAMUS I: Risk-based Design of Offshore Oil and Gas Production System–Management of Uncertainties by Integrating Flow Assurance and Reliability Aspects into a Stochastic Petri Nets Model. In *Proceedings of the European Safety and Reliability Conference*.

[26] DNVGL (2017a). MAROS. http://www.dnvgl.com. [accessed 01-September-2017].

[27] DNVGL (2017b). TARO. http://www.dnvgl.com. [accessed 01-September-2017].

[28] Dutuit, Y., Innal, F., Rauzy, A., and Signoret, J.-P. (2008a). Probabilistic assessments in relationship with safety integrity levels by using fault trees. *Reliability Engineering & System Safety*, 93(12):1867–1876.

[29] Dutuit, Y., Rauzy, A., and Signoret, J.-P. (2008b). A snapshot of methods and tools to assess safety integrity levels of high-integrity protection systems. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 222:371–379.

[30] Eisinger, S., Isaksen, S., Johnsen, Ø., and Averbuch, D. (2007). FAMUS II: Risk-based design of offshore oil and gas production system–management of uncertainties by integrating flow assurance and reliability aspects into a RAM model. In *Proceedings of the European Safety and Reliability Conference*.

[31] Elsayed, E. A. (2012). *Reliability engineering*. John Wiley & Sons.

[32] Fink, O. and Zio, E. (2013). Semi-Markov processes with semi-regenerative states for the availability analysis of chemical process plants with storage units. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 227(3):279–289.

[33] Fritzson, P. (2003). *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons.

[34] Gamma, E. (2002). Design patterns−ten years later. In *Software pioneers*, pages 688–700. Springer.

[35] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional.

[36] Gao, X., Barabady, J., and Markeset, T. (2010). An approach for prediction of petroleum production facility performance considering arctic influence factors. *Reliability Engineering & System Safety*, 95(8):837–846.

[37] Gudemann, M. and Ortmeier, F. (2010). A framework for qualitative and quantitative formal model-based safety analysis. In *2010 IEEE 12th International Symposium on High Assurance Systems Engineering*, pages 132–141.

[38] Guo, Q., Xu, P., Pei, X., Wong, S., and Yao, D. (2017). The effect of road network patterns on pedestrian safety: a zone-based bayesian spatial modeling approach. *Accident Analysis & Prevention*, 99:114–124.

[39] Hamid, B. and Perez, J. (2016). Supporting pattern-based dependability engineering via model-driven development: Approach, tool-support and empirical validation. *Journal of Systems and Software*, 122:239–273.

[40] Hauge, S., Hokstad, P., Håbrekke, S., and Lundteigen, M. A. (2016). Common cause failures in safety-instrumented systems: Using field experience from the petroleum industry. *Reliability Engineering & System Safety*, 151:34–45.

[41] Hjorteland, A., Aven, T., and Østebø, R. (2007). Uncertainty treatment in production assurance analyses throughout the various phases of a project. *Reliability Engineering & System Safety*, 92(10):1315–1320.

[42] Hokstad, P. (1988). Assessment of production regularity for subsea oil/gas production systems. *Reliability Engineering and System Safety*, 20:127–146.

[43] Hukerikar, S. and Engelmann, C. (2016). Resilience design patterns - a structured approach to resilience at extreme scale. Technical report, Oak Ridge National Laboratory, Tennessee, US.

[44] IEC (1990). International Electrotechnical Vocabulary, Chapter 191: Dependability and quality of service.

[45] IEC (2002). IEC 62278 Railway applications - Specification and demonstration of reliability, availability, maintainability and safety (RAMS).

[46] IEC (2005). IEC 62061 Safety of machinery - Functional safety of safety-related electrical, electronic and programmable electronic control systems.

[47] IEC (2006a). IEC 61025 Fault tree analysis.

[48] IEC (2006b). IEC 61165 application of markov techniques.

[49] IEC (2007). IEC 62425 Railway applications - Communication, signalling and processing systems - safety related electronic systems for signalling.

[50] IEC (2010a). IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems.

[51] IEC (2010b). IEC 62502 Analysis techniques for dependability - Event tree analysis (ETA).

[52] IEC (2011). IEC 61513 Nuclear power plants - Instrumentation and control important to safety - General requirements for systems.

[53] IEC (2012). IEC 62551 Analysis techniques for dependability - Petri net techniques.

[54] IEC (2015). IEC 62279 Railway applications - Communication, signalling and processing systems - software for railway control and protection systems.

[55] IEC (2016a). IEC 61078 Reliability block diagrams.

[56] IEC (2016b). IEC 61511 Functional safety - Safety instrumented systems for the process industry sector.

[57] Innal, F., Chebila, M., and Dutuit, Y. (2016). Uncertainty handling in safety instrumented systems according to IEC 61508 and new proposal based on coupling Monte Carlo analysis and fuzzy sets. *Journal of Loss Prevention in the Process Industries*, 44:503 – 514.

[58] ISO (2008). ISO 20815, Petroleum, petrochemical and natural gas industries: Production assurance and reliability management.

[59] ISO (2011). ISO 26262 Road vehicles - Functional safety.

[60] ISO (2013). ISO/TR 12489, Petroleum, petrochemical and natural gas industries: Reliability modelling and calculation of safety systems.

[61] ISO (2015). ISO 13849-1 Safety of machinery - Safety-related parts of control systems - Part 1: General principles for design.

[62] Jahanian, H. (2015). Generalizing PFD formulas of IEC 61508 for KooN configurations. *ISA transactions*, 55:168–174.

[63] Jiang, T. and Liu, Y. (2017). Parameter inference for non-repairable multi-state system reliability models by multi-level observation sequences. *Reliability Engineering & System Safety*, 166:3–15.

[64] Jigar, A. A., Liu, Y., and Lundteigen, M. A. (2016). Spurious activation analysis of safety-instrumented systems. *Reliability Engineering & System Safety*, 156:15 – 23.

[65] Jin, H., Lundteigen, M. A., and Rausand, M. (2012). Uncertainty assessment of reliability estimates for safety-instrumented systems. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of risk and reliability*, 226(6):646–655.

[66] Jin, H. and Rausand, M. (2014). Reliability of safety-instrumented systems subject to partial testing and common-cause failures. *Reliability Engineering & System Safety*, 121:146–151.

[67] Kajtazovic, N., Preschern, C., Höller, A., and Kreiner, C. (2014). Towards pattern-based reuse in safety-critical systems. In *Proceedings of the 14th European Conference on Pattern Languages of Programs*. ACM.

[68] Kawauchi, Y. and Rausand, M. (2002). A new approach to production regularity assessment in the oil and chemical industries. *Reliability Engineering & System Safety*, 75(3):379–388.

[69] Kehren, C. (2005). *Motifs formels d'architectures de systèmes pour la sûreté de fonctionnement*. PhD thesis, Ecole nationale superieure de l'aeronautique et de l'espace, Toulouse, France.

[70] Khalil, M., Prieto, A., and Hölzl, F. (2014). A pattern-based approach towards the guided reuse of safety mechanisms in the automotive domain. In Ortmeier, F. and Rauzy, A., editors, *Model-Based Safety and Assessment: 4th International Symposium, IMBSA 2014, Munich, Germany, October 27-29, 2014, Proceedings*, pages 137–151. Springer.

[71] Kim, H. and Kim, P. (2017). Reliability models for a nonrepairable system with heterogeneous components having a phase-type time-to-failure distribution. *Reliability Engineering & System Safety*, 159:37–46.

[72] Kloul, L., Prosvirnova, T., and Rauzy, A. (2013). Modeling systems with mobile components: a comparison between AltaRica and PEPA nets. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 227(6):599–613.

[73] Kumar, M., Verma, A. K., and Srividya, A. (2008). Modeling demand rate and imperfect proof-test and analysis of their effect on system safety. *Reliability Engineering & System Safety*, 93:1720–1729.

[74] Levitin, G., Podofillini, L., and Zio, E. (2003). Generalised importance measures for multi-state elements based on performance level restrictions. *Reliability Engineering & System Safety*, 82(3):287–298.

[75] Levitin, G., Xing, L., and Dai, Y. (2014). Minimum mission cost cold-standby sequencing in non-repairable multi-phase systems. *IEEE Transactions on Reliability*, 63(1):251–258.

[76] Lipaczewski, M., Ortmeier, F., Prosvirnova, T., Rauzy, A., and Struck, S. (2015). Comparison of modeling formalisms for safety analyses: SAML and AltaRica. *Reliability Engineering & System Safety*, 140:191–199.

[77] Liu, Y. and Chen, C.-J. (2017). Dynamic Reliability Assessment for Nonrepairable Multistate Systems by Aggregating Multilevel Imperfect Inspection Data. *IEEE Transactions on Reliability*, 66(2):281–297.

[78] Liu, Y. and Rausand, M. (2013). Reliability effects of test strategies on safety-instrumented systems in different demand modes. *Reliability Engineering & System Safety*, 119:235 – 243.

[79] Liu, Y. and Rausand, M. (2016). Proof-testing strategies induced by dangerous detected failures of safety-instrumented systems. *Reliability Engineering & System Safety*, 145:366–372.

[80] Longhi, A. E. B., Pessoa, A. A., and de Almada Garcia, P. A. (2015). Multiobjective optimization of strategies for operation and testing of low-demand safety instrumented systems using a genetic algorithm and fault trees. *Reliability Engineering & System Safety*, 142:525–538.

[81] Lundteigen, M. A. and Rausand, M. (2007). Common cause failures in safety instrumented systems on oil and gas installations: Implementing defense measures through function testing. *Journal of Loss Prevention in the process industries*, 20(3):218–229.

[82] Lundteigen, M. A. and Rausand, M. (2008). Spurious activation of safety instrumented systems in the oil and gas industry: Basic concepts and formulas. *Reliability Engineering & System Safety*, 93:1208 – 1217.

[83] Mannan, S. (2005). *Lees' Loss prevention in the process industries: Hazard identification, assessment and control*. Elsevier Butterworth-Heinemann.

[84] MathWorks (2017). Simulation and Model-Based Design. http://www.mathworks.com/products/simulink.html. [accessed 01-September-2017].

[85] Meng, H., Kloul, L., and Rauzy, A. (2015). Production availability modelling of FPSO system using stochastic Petri nets. In *Safety and Reliability of Complex Engineered Systems: ESREL 2015*, pages 2271–2279, Zurich, Switzerland.

[86] Meng, H., Kloul, L., and Rauzy, A. (2017a). Modeling patterns for performance analyses of offshore production systems. In *The Proceedings of the 27th International Ocean and Polar Engineering Conference*, pages 1199–1205, San Francisco, USA. International Society of Offshore and Polar Engineers.

[87] Meng, H., Kloul, L., and Rauzy, A. (2017b). A pattern-based methodology for reliability assessment of safety instrumented systems. In *The Proceedings of the 2017 IEEE International Symposium on Systems Engineering (IEEE ISSE 2017)*, Vienna, Austria.

[88] Mhenni, F., Nguyen, N., and Choley, J.-Y. (2016). SafeSysE: A Safety Analysis Integration in Systems Engineering Approach. *IEEE Systems Journal*, pages 1–12.

[89] Morel, M. (2014). Model-based safety approach for early validation of integrated and modular avionics architectures. In Ortmeier, F. and Rauzy, A., editors, *Model-Based Safety and Assessment: 4th International Symposium, IMBSA 2014, Munich, Germany, October 27-29, 2014, Proceedings*, pages 57–69. Springer.

[90] Naseri, M., Baraldi, P., Compare, M., and Zio, E. (2016). Availability assessment of oil and gas processing plants operating under dynamic arctic weather conditions. *Reliability Engineering & System Safety*, 152:66–82.

[91] NORSOK (1998). Z-016 regularity management & reliability technology.

[92] Oliveira, L. F. and Abramovitch, R. N. (2010). Extension of ISA TR84. 00.02 PFD equations to KooN architectures. *Reliability Engineering & System Safety*, 95(7):707–715.

[93] Papadopoulos, Y. and McDermid, J. A. (1999). Hierarchically performed hazard origin and propagation studies. In Felici, M. and Kanoun, K., editors, *Proceedings of 18th International Conference on Computer Safety, Reliability and Security, SAFECOMP'99 Toulouse, France, September 27–29, 1999*, pages 139–152. Springer Berlin Heidelberg.

[94] Papadopoulos, Y., Walker, M., Parker, D., Rüde, E., Hamann, R., Uhlig, A., Grätz, U., and Lien, R. (2011). Engineering failure analysis and design optimisation with HiP-HOPS. *Engineering Failure Analysis*, 18(2):590 – 608.

[95] Point, G. and Rauzy, A. (1999). AltaRica: Constraint automata as a description language. *Journal européen des systèmes automatisés*, 33(8-9):1033–1052.

[96] Preschern, C., Kajtazovic, N., Höller, A., and Kreiner, C. (2014). Pattern-based safety development methods: overview and comparison. In *Proceedings of the 14th European Conference on Pattern Languages of Programs*. ACM.

[97] Preschern, C., Kajtazovic, N., and Kreiner, C. (2013). Building a safety architecture pattern system. In *Proceedings of the 13th European Conference on Pattern Languages of Program*. ACM.

[98] Prosvirnova, T. (2014). *AltaRica 3.0: a Model-Based approach for Safety Analyses*. PhD thesis, Ecole Polytechnique, Palaiseau, France.

[99] Prosvirnova, T., Batteux, M., Brameret, P.-A., Cherfi, A., Friedlhuber, T., Roussel, J.-M., and Rauzy, A. (2013). The AltaRica 3.0 project for model-based safety assessment. In *Proceedings of 4th IFAC Workshop on Dependable Control of Discrete Systems, DCDS*, pages 127–132.

[100] Rahimi, M. and Rausand, M. (2013). Monitoring human and organizational factors influencing common-cause failures of safety-instrumented system during the operational phase. *Reliability Engineering & System Safety*, 120:10–17.

[101] Ramos, A. L., Ferreira, J. V., and Barceló, J. (2012). Model-based systems engineering: An emerging approach for modern systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(1):101–111.

[102] Ramzali, N., Lavasani, M. R. M., and Ghodousi, J. (2015). Safety barriers analysis of offshore drilling system by employing fuzzy event tree analysis. *Safety science*, 78:49–59.

[103] Rausand, M. (2011). *Risk assessment: Theory, methods, and applications*. John Wiley & Sons.

[104] Rausand, M. (2014). *Reliability of Safety-Critical Systems: Theory and Applications*. John Wiley & Sons.

[105] Rausand, M. and Høyland, A. (2004). *System reliability theory: models, statistical methods, and applications*. John Wiley & Sons.

[106] Rauzy, A. (2002). Mode automata and their compilation into fault trees. *Reliability Engineering & System Safety*, 78(1):1–12.

[107] Rauzy, A. (2004). An experimental study on iterative methods to compute transient solutions of large Markov models. *Reliability Engineering & System Safety*, 86:105–115.

[108] Rauzy, A. (2008). Guarded transition systems: a new states/events formalism for reliability studies. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 222(4):495–505.

[109] Schönbeck, M., Rausand, M., and Rouvroye, J. (2010). Human and organisational factors in the operational phase of safety instrumented systems: A new approach. *Safety Science*, 48:310–318.

[110] Sharvia, S. and Papadopoulos, Y. (2015). Integrating model checking with HiP-HOPS in model-based safety analysis. *Reliability Engineering & System Safety*, 135:64–80.

[111] Signoret, J.-P. (2000). SAFERELNET–Production Availability Test Case Version 1, Document of TOTAL–DGEP. Technical report, TDO/EXP/SRF 04-013.

[112] Signoret, J.-P. (2010). Production availability. In Guedes Soares, C., editor, *Safety and Reliability of Industrial Products, Systems and Structures*. CRC Press.

[113] Signoret, J.-P. (2012). Petri nets modelling and calculations for RAM and SIL analyses. In Soares, C. and Santos, T., editors, *Maritime Technology and Engineering*. CRC Press.

[114] Signoret, J.-P., Dutuit, Y., Cacheux, P.-J., Folleau, C., Collas, S., and Thomas, P. (2013). Make your petri nets understandable: Reliability block diagrams driven petri nets. *Reliability Engineering & System Safety*, 113:61–75.

[115] Sklet, S. (2006). Safety barriers: Definition, classification, and performance. *Journal of loss prevention in the process industries*, 19(5):494–506.

[116] Stamatelatos, M., Vesely, W., Dugan, J., Fragola, J., Minarick, J., and Railsback, J. (2002). Fault tree handbook with aerospace applications. Technical report, NASA Washington, DC.

[117] Stroustrup, B. (2009). *Programming: principles and practice using C++*. Pearson Education.

[118] Sun, T. (2017). Production Availability Analysis- Implications on Modelling due to Subsea Conditions. Master's thesis, Norwegian University of Science and Technology.

[119] Taivalsaari, A., Noble, J., and Moore, I. (1999). *Prototype-Based Programming: Concepts, Languages, and Applications*. Springer.

[120] Torres-Echeverria, A., Martorell, S., and Thompson, H. (2009). Modelling and optimization of proof testing policies for safety instrumented systems. *Reliability Engineering & System Safety*, 94:838–854.

[121] Torres-Echeverria, A., Martorell, S., and Thompson, H. (2011). Modeling safety instrumented systems with MooN voting architectures addressing system reconfiguration for testing. *Reliability Engineering & System Safety*, 96:545–563.

[122] TOTAL (2017). GRIF. http://grif-workshop.com. [accessed 01-September-2017].

[123] Verma, A., Khan, S. D., Maiti, J., and Krishna, O. (2014). Identifying patterns of safety related incidents in a steel plant using association rule mining of incident investigation reports. *Safety science*, 70:89–98.

[124] Vesely, W. E., Goldberg, F. F., Roberts, N. H., and Haasl, D. F. (1981). Fault tree handbook. Technical report, U.S. Nuclear Regulatory Commission.

[125] Vesteraas, A. H. (2008). Comparison of methods and software tools for availability assessment of production systems. Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway.

[126] Zhang, H., Innal, F., Dufour, F., and Dutuit, Y. (2014). Piecewise deterministic Markov processes based approach applied to an offshore oil production system. *Reliability Engineering & System Safety*, 126:126–134.

[127] Zio, E. (2009). Reliability engineering: Old problems and new challenges. *Reliability Engineering and System Safety*, 94(2):125–141.

[128] Zio, E. (2013). *The Monte Carlo Simulation Method for System Reliability and Risk Analysis*. Springer Series in Reliability Engineering. Springer London, London, England.

[129] Zio, E., Baraldi, P., and Patelli, E. (2006). Assessment of the availability of an offshore installation by Monte Carlo simulation. *International Journal of Pressure Vessels and Piping*, 83(4):312–320.

[130] Zio, E., Marella, M., and Podofillini, L. (2007). A Monte Carlo simulation approach to the availability assessment of multi-state systems with operational dependencies. *Reliability Engineering and System Safety*, 92:871–882.

université PARIS-SACLAY | ÉCOLE DOCTORALE INTERFACES Approches Interdisciplinaires : Fondements, Applications et Innovation

Titre: Modélisation des patterns d'analyse des performances des systèmes de production et de sûreté de fonctionnement dans l'industrie des procédés

Mots clés: Pattern, Performance, Fiabilité, Disponibilité, AltaRica

Résumé: Les systèmes de production et de sûreté de fonctionnement sont d'une importance majeure dans l'industrie des procédés. Leurs performances impactent directement les intérêts de l'industrie. Ces systèmes ont des comportements similaires. Ces comportements peuvent être conceptualisés dans des modèles via des patterns de modélisation. La réutilisation de ces patterns permet de rendre le processus de modélisation à la fois simplifiée et plus efficace.

Dans cette thèse, nous proposons un ensemble varié de patterns de modélisation. Ils sont classés en fonction de leur usage, ce qui reflète le fonctionnement d'un pattern de modélisation. Les patterns sont présentés sous forme d'un catalogue. Sur la base de l'étude de nombreux systèmes de production et de sécurité, vingt-quatre (24) patterns de modélisation sont introduits. Chaque pattern est représenté par un ensemble d'éléments structurés. Nous proposons une méthodologie basée sur les patterns pour l'analyse des performances des systèmes de production et de sûreté de fonctionnement.

Pour tester la pertinence des patterns de modélisation suggérés, nous avons mené des études expérimentales sur un ensemble de systèmes de production et de sûreté. Tous les systèmes de validation sont extraits de la littérature. Ces systèmes traitent la majorité des difficultés de modélisation détectées auparavant. Une comparaison est effectuée entre les résultats obtenus en utilisant la modélisation basée sur les patterns et ceux rapportés dans la littérature.

Title: Modeling Patterns for Performance Analysis of Production and Safety Systems in Process Industry

Keywords: Pattern, Performance, Reliability, Availability, AltaRica

Abstract: Production and safety systems are crucial in the process industry. Their performances affect significantly the industry interests. These systems have common behaviors. Such behaviors can be captured in models via modeling patterns. By reusing modeling patterns, the modeling process can be simplified and made more efficient.

In this thesis, we propose a versatile set of modeling patterns. They are classified according to their purpose, which reflects what a modeling pattern works for. Modeling patterns are exhibited as a catalog. Based on reviewing numerous production and safety systems, twenty-four (24) modeling patterns are introduced. Each pattern is illustrated with a set of structured items. We propose a pattern-based methodology for performance analysis of production and safety systems.

To test the applicability of proposed modeling patterns, we conducted experimental studies on a set of production and safety systems. All systems are extracted from the literature. These systems are declared to cover most of modeling difficulties. Comparisons are performed between the results obtained using modeling patterns and those reported in the literature.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France