# Applications of Dependable Computing Concepts to National Infrastructure Systems

Thesis by

Roberta Velykienė

In Partial Fulfillment of the Requirements
for the Degree of
Master of Philosophy


School of Computing Science,
Newcastle University,
Newcastle upon Tyne, UK

October, 2013

*Andriui.*

*To the best husband...*

## Acknowledgements

The greatest thanks are to my supervisor Cliff Jones for his gentle and patient guidance, and—especially—for the personal support during difficult times. He was able to see me through and help me when I needed it the most—thank you.

Thanks to all the nice people I got a chance to know and work with at Newcastle University.

Also, I am grateful to the ITRC project for providing a context to this research and to EPSRC for supporting the studies.

Finally, I am thankful to the examiners for their time, comments and advice.

# Abstract

Modern infrastructures such as energy, transport or information and communication technologies are complex and highly interdependent systems. Modern society and economy place growing expectations and reliance on them. Failures in infrastructure systems can affect large parts of the everyday life and inflict significant losses.

This thesis is concerned with reasoning about the dependability of national infrastructures. It takes a holistic view of national infrastructure systems and treats them as an interconnected system-of-systems. The thesis focuses on describing and analysing failure in such systems-of-systems, especially how it propagates between systems via infrastructure interdependencies.

The aim is to improve the ability to reason about interdependent infrastructure systems by reviewing notions and techniques from dependable computing and assessing how they can be applied to national infrastructure systems analysis. The thesis proposes a framework to describe and reason about complex infrastructure systems. The framework employs fault-error-failure concepts; emphasises the roles of assumptions, boundaries and structure in systems description; uses top-down system view supplemented with formal techniques. The framework is extended to include the planning process and humans as systems within the same analysis umbrella.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Infrastructures such as energy, transport or information and communication technologies (ICT) have a crucial role in the everyday function of modern society. They are called critical national infrastructures and have been identified as the backbone of a modern economy [Tre11]. These systems have gradually grown from separate services to integrated and highly interdependent *system-of-systems*.

Infrastructures are built with some redundancy and are quite dependable in normal conditions. However, the current ageing infrastructure has to face challenges from growing demand and changing climate conditions, which uncover new vulnerabilities. Recent events such as floods in Northumberland, UK and other natural and man-caused disasters had significant negative impact due to interdependencies between infrastructure systems. This suggests that re-evaluation of current infrastructure is needed addressing dependability issues as well as providing opportunities for research in improving infrastructure efficiency by sharing resources and exploiting ICT potential, changing human behaviour, etc.

Current approaches to addressing issues, investment and planning of infrastructure systems must be cross-sectoral and include the various links between different infrastructure systems. Such approaches are being put in motion in government, academia and industry. The UK government plans announced in 2011 feature cross-sector investment in infrastructures and include over 500 projects and programmes worth over £250 billion [Tre11]. Although part of this is expected to be privately funded, the government aims to set out a high-level strategy for future infrastructure development.

The long-term strategies for UK future infrastructures will need to plan upgrades to the ageing UK national infrastructure as well as developing completely new infrastructure systems to satisfy needs and improve efficiency in the future for society

and the economy [Cou12, PM1]. Unfortunately, development of long-term strategies for modern interdependent infrastructures faces challenges in lack of mature approaches to design and analyse such systems. The major difficulties arise from the complexity of these infrastructure systems as well as from rather limited experience in cross-sectoral thinking and research. This exposes the need for research into designing, analysing and evaluating complex interdependent infrastructures in a holistic way, which has been attempted by numerous research projects in the last decade.

This thesis is linked with the research programme of the UK Infrastructure Transitions Research Consortium (ITRC)[1]. ITRC was established to study long term infrastructure planning issues. The ITRC research programme aims to inform analysis, planning and design of national infrastructure, with a focus on interaction between infrastructures. The expected research outcomes span theory, models and practical decision support tools to enable strategic analysis and planning of national infrastructure systems in changing economic, social and natural environments that might be faced in the future. This thesis aims to contribute to such goals by exploring techniques for high-level description and analysis of future infrastructure systems.

Increasing dependence on infrastructure has resulted in increasing demand for dependability. Significant failures that affect dependability of national infrastructure arise due to interdependencies between individual infrastructure systems and can have a major impact, affecting different services throughout interconnected systems. This thesis is concerned with infrastructure interdependencies and failure propagation between them.

Some interdependencies of infrastructures are inherent (e.g. ICT requires electricity to operate and visa versa). Moreover, further interdependencies in modern infrastructures bring opportunities to achieve major savings and efficiency.[2] The interdependencies, however, come with a price: they add vulnerabilities and thus affect the dependability of infrastructures. Cascading failures occurring through links and connections of interdependent infrastructures can cause large costs and negative effects on a national scale.[3]

[1] `http://www.itrc.org.uk`

[2] E.g. reuse of the Channel Tunnel to lay electrical inter-connector to Europe would save £130-180m vs. a new line across the sea bed [Tre11].

[3] For example, a 2003 electric power blackout in the United States and Canada affected water supply, transportation and communication sectors with estimated costs of $4-$10 billion [For04].

Interdependence analysis is one of the main points of cross-sectoral infrastructure research. It is usually approached with complex network modelling and simulation to identify weak links and vulnerable sections of the infrastructure network. The exercises, however, are often limited in scope by availability of data or size of models. Furthermore, such an approach favours a bottom-up view and may fail to provide the full picture. Section 5.2 provides a further overview of the common approaches and issues.

This thesis investigates how to understand and mitigate the vulnerabilities introduced by system interdependencies and thus achieve better dependability in future infrastructures. The aim of the thesis, however, is not to improve existing or creating new simulation techniques, but to provide an alternative approach to describe, model and analyse infrastructure interdependencies. The approach builds upon the ideas in dependable computing research, which can be adapted for infrastructure systems.

Parallels can be easily established between systems-of-systems analysed in computing science research and infrastructure systems. This suggests that the established concepts, techniques and practices from dependable computing can be adapted to describe and analyse national infrastructures. Various software and hardware computer systems are used in critical environments, such as aeroplane engine controls, power plant, life support systems, etc. They are often complex systems with high interconnectivity between their components. However, the critical environment they are used in requires such systems to be designed with high assurance and precision to avoid life-threatening failures. Dependability research in computing science has been concerned with these issues for the last several decades and has delivered theory and practice for developing such systems, ranging from theoretical notions and taxonomies of failures to rigorous development using formal methods and formal verification.

This thesis explores how the theory and practice of dependable computing could be applied to national infrastructure systems, in particular to the design and analysis process that concerns with the infrastructure interdependencies. The research is guided by three hypotheses, which are presented below.

The main hypothesis H1 proposes to adapt concepts from dependable computing to describing and reasoning about infrastructures.

**Hypothesis H1** *Established concepts from dependable computing can help us understand the interdependencies between infrastructure systems and this understand-*

*ing can be used to increase the dependability of infrastructures.*

Concepts from dependable computing theory, in particular notions notions such as fault, error and failure, would bring precision to describing failure propagation between infrastructure systems. Furthermore, clear and precise descriptions provide a foundation for the design of complex infrastructure systems and reasoning about their dependability. Chapter 2 re-examines the core dependability definitions in the context of national infrastructure systems.

Under the umbrella of the H1 hypothesis this thesis explores several approaches to increasing the dependability of infrastructure systems. A top-down view on infrastructure system description and development provides a structured way to tackle complexity and introduce details when necessary for reasoning about system dependability. This thesis raises the hypothesis that these techniques can be applied to good effect to national infrastructure systems:

**Hypothesis H2** *By analysing infrastructure systems from the top-down view, one can gain a better understanding of interdependencies and failure propagation.*

Such an approach benefits from precise identification of assumptions and even development of a formal specification to establish the context for the dependability arguments. Chapter 3 brings these and other techniques from computing science, with the aim to develop a framework to describe, understand and reason about infrastructure systems, their interdependencies and failure propagation among others.

Moreover, this research investigates the case when the originating system failure lies outside the conventional national infrastructures. System faults can be inherited from the planning and design process, etc. Thesis proposes to expand the scope of the analysis framework and include the planning process in the analysis as a separate system. This is covered under the following hypothesis of "systems-generating-systems" and explored to extent in Chapter 4:

**Hypothesis H3** *By explicitly studying the concept of a planning system that gives rise to future (or changes existing) infrastructure systems, we can identify and reduce latent faults or errors in future infrastructures that could engender failure in those systems.*

These hypotheses set out the scene for the research presented in this thesis. The main thesis chapters propose a framework to address these hypotheses. Various

aspects of the framework are detailed, in particular how the dependable computing concepts would be adapted for infrastructure systems. Furthermore, basic examples and a high-level case study aims to present applications of the said ideas. These attempt to illustrate the initial steps of how a full size application is to be performed. Note, however, that a full evaluation and a comparison study with current infrastructure analysis techniques is outside the scope of this thesis. Such an evaluation project is a major undertaking that needs to cover the full development of a new or analysis of an existing infrastructure system.

## 1.1 Contributions

The application of dependable computing ideas provides a novel approach to infrastructure systems description and analysis. The focus on high-level description and reasoning is a departure from activities used by current analysis methods, such as data collection, low-level modelling and simulation.

The thesis adapts dependable computing techniques for infrastructure systems and provides hints, examples and a case study on how they could be applied and benefit the analysis. Furthermore, adaptation of some concepts reveals parts of infrastructure analysis that are overlooked, such as importance of assumptions, relationship of cost to achieved dependability, etc.

The proposed framework is generalised in application to various entities related to infrastructure systems. Under the concept "everything is a system", the thesis extends the application of ideas beyond conventional infrastructure systems. This includes human operators, which can be treated as a *human system* to model failures caused by humans. Furthermore, by considering planning systems within the same framework, a different dimension of failure propagation is explored: system faults caused by failures in system design or development.

During the earlier part of the author's MPhil studies, a technical report on ICT infrastructure constraints on evolving physical infrastructure [VJ11] was produced. It contributed to the development of high-level strategies within the ITRC project as part of the fast-track analysis of current and prospective national infrastructures [HHHN12].

## 1.2 Thesis outline

The remaining chapters are organised as follows. Basic dependability notions from computing science are adapted to infrastructure systems in Chapter 2. Chapter 3 argues about the importance of assumptions to system description as well as proposes a top-down view of infrastructure systems supplemented with (formal) specification. Chapter 4 extends the framework scope by including planning systems as possible causes for system faults. An overview of current infrastructure analysis approaches and related work is provided in Chapter 5. Finally, a simplified case study is attempted in Chapter 6 to illustrate how the main ideas would be applied within a single reasoning exercise, followed by conclusions and future work directions in Chapter 7.

# Chapter 2

# Dependability key concepts

One of the central concepts when talking about system dependability is the notion of system failure. This chapter sets out the basic terminology and notions of system failure in the field of dependable computing and related areas. The aim here is to take the underlying ideas and key concepts of dependable computing and adapt them in the context of national infrastructure systems, as stated in the H1 hypothesis. These concepts help with understanding and reasoning about failure and dependability of infrastructure systems. They form the core notions within the overall reasoning framework proposed in this thesis.

This chapter explores several of the main areas of system dependability: describing and reasoning about failure; the errors and faults that cause failure; considering human aspects in system dependability; weighing system dependability against the cost to achieve it. The aim is not to present some new method of improving system dependability, but to introduce concepts and ideas to describe and reason about systems, their relationships and dependability. For example, describing system boundaries or identifying faults, errors, failures and links of causality does not improve system dependability by itself, but gives a tool to recognise and improve upon weak or unaccounted facets of a system or a system-of-system configuration.

The aim of this thesis is to show how to take these concepts, which are being applied in dependable computing, and adapt them to the context of infrastructure systems. Some of the current methods used in infrastructure systems are reviewed in Chapter 5. Chapters 3 and 4 use the notions introduced here in the wider view of developing or describing a whole system, or even including the development process itself.

## 2.1 On systems (of systems)

The notion of *system*, and in particular, *system-of-systems* can be used to describe entities of different complexity and relationships. This thesis adopts an abstract definition of a system, as formulated in [ALRL04]: *a system is an entity that interacts with other entities*. Such definition is not concerned with physical properties of the system or behaviour restrictions—the important feature of a system is its relationship with other systems, e.g. services it provides and consumes, different other interdependencies, etc. The abstract definition of the system allows applying the ideas presented in this thesis to "systems" in different domains of infrastructure, e.g. physical infrastructure systems, human systems or even treating the process of *infrastructure planning* as a system.

A collection of systems that interact together can give rise to some emergent behaviour or service. Such collection can thus also be considered a new system, which consists of other systems. To emphasise the importance of component systems, the name "system-of-systems" is used within this thesis. Note that in [ALRL04] these concepts are defined as a system and its *component* sub-systems. In this thesis, "system-of-systems" and "system with sub-components" will be used interchangeably, but system-of-systems is preferred to emphasise the modularity and internal relationships between the sub-systems. Furthermore, the ideas presented in this thesis can be applied recursively and independently to each component system.

Note that such abstract notion of "system-of-systems" is somewhat different and more permissive than the established definition. The common definition of system-of-systems requires independence, emergent behaviour and geographic distribution of its component systems [Mai98]. Most of these requirements are satisfied by systems within national infrastructure. For example, energy, transport, ICT, water and waste infrastructure systems are independent in their operation and management, they have different evolutionary developments and emergent behaviours. The geographic distribution appears at different levels of abstraction, e.g. power plants are distributed within the energy system-of-systems, but the energy system *as a whole* is not geographically distributed in regards to the other top-level systems, e.g. ICT. Therefore, while infrastructure systems analysed in this thesis can satisfy the established definition of "system-of-systems" at certain levels of abstraction, in general these requirements are not of importance to the ideas proposed in this thesis. For this reason, the "system-of-systems" describes a collection of arbitrary

logical constructs that allow us to modularise a complex structure to accommodate reasoning and description.

## 2.2 System dependability

The fundamental concepts in the field of dependable computing are set out by Avižienis et al. [ALRL04]. This thesis adopts the proposed terminology and notations, especially the crucial tripartite distinction of a system's "problem" as *fault*, *error* and *failure*. These concepts are used throughout the entire discussion in this thesis about all systems, including computing, infrastructure and others.

Dependability is a very broad concept that encompasses various properties related to the quality of a system's service over a period of time. The notion addresses availability, reliability, safety, integrity and maintainability of a system. This thesis does not address these specific facets of dependability and instead aims to link dependability to different kinds of failure. For example, instead of talking separately about how some events would affect a system's availability or safety, these are instead generalised and included under general notions of *failure* and *dependability*.

Dependability defines a system's ability to deliver its intended[1] service to the user. Note that the user could be another system that relies on the delivery of service by the first system. Because of the above-mentioned generalisation, this thesis employs the broader definition linking dependability with failures: *dependability is the system's ability to avoid service failures that are more frequent and more severe than acceptable* [ALRL04]. Thus the important link here is between some failure and how it affects the system dependability in general.

When talking about system failure, it is beneficial to discuss its causes and context. The notions of *errors*, *faults* and even fault-error-failure chains as causes are introduced later. However, first it is essential to identify the context in which the failure is discussed, i.e. the system under consideration. Reasoning about failure and its causes can only be meaningful if it is grounded in a clear identification of the system under discussion. Modern infrastructure is provided by a system-of-systems but, if one wants to analyse, for example, the failure to provide electricity to an area, it is crucial to distinguish the electricity generation and distribution systems and

[1]The term *intended service* is used instead of the *correct* one, because *correctness* requires unambiguous documentation. Ideally, a specification is used to define the correct system behaviour (see Section 3.5). However, common practice is that a system's correct state or service are not documented or the documentation is not complete and precise. Therefore, generally, a definition of a system's intended service is used [ALRL04].

furthermore the system of fuel provision (to power stations), etc. Disagreements about the cause of a failure often have their origins in the fact that the discussants are considering different systems or different views of the system. The identification of the system boundaries and the scope of discussion (i.e. systems included in the reasoning) can clarify the context. The discussion on system boundaries is continued in Sections 2.3 and 3.3.

The precision about identifying the discussed system should be kept when talking about failure and its cause as well. The basic terminology and notions of system failure in the field of dependable computing were established by Avižienis and others [ALRL04]. The actual words chosen are less important than the separation of three concepts—but given their wide use, it was chosen to stay with the words "fault, error and failure". Without much elaboration, these concepts can be understood as following:

**Failure** A service failure is a deviation of a system's service from a correct one. (Note that some kind of justification is needed to judge a service incorrect.)

**Error** An error is a system state that deviates from one needed for correct operation; an error may thus lead to a failure. (Note that an error may not lead to a failure and a failure may be a result of more than one error.)

**Fault** A fault is simply a cause of an error. In many cases faults are the result of external failures in other systems (e.g. damage to the system). Faults may also be created during development and thus be part of the system until fixed, but may never actually be activated (lead to error). An exhaustive fault taxonomy is given in [ALRL04].

These notions comprise a chain that describes the failure life-cycle:

$$\cdots \rightarrow fault \rightarrow error \rightarrow failure \rightarrow \cdots$$

When describing a failure, the error and fault leading to it cannot be excluded. The following sections elaborate on these definitions and on how their chains link together in the event of failure propagation.

### 2.2.1 On failures

The notion of failure may seem simple and intuitive enough that it may prevent one from elaborating on its description: e.g. *system stopped working*, *system produced*

*an incorrect result*, etc. However, one cannot talk about failure without agreeing on what is actually the intended service of a system. For example, consider a power plant shut down for a scheduled maintenance—it fails to provide its service (generating electricity) and the dependent systems could treat this as a failure in providing electricity. However, this is not a failure, because maintenance stops are part of a correct service description—the plant design requires maintenance stops.

This shows that a failure is a *judgement* of the system made by humans or another system. The fact that the power plant is not producing electricity can be treated differently: as a failure by the electricity user who is not aware of maintenance stops; or as normal activity, as intended by system design. Therefore, in general, the judgement on whether a failure has occurred may differ. Furthermore, the judging system itself may be incorrect! For example, a faulty sensor system may misreport a failure of the monitored infrastructure system.

To put these ideas in a general framework, a failure could be described in the context of a system-of-systems. A *failure* of a system within this framework is an event that occurs when the system's service deviates from the intended one. The identification of a failure is *judged* by its user, which could be another system [ALRL04]. The judgemental system can be an automated one (e.g. a control system), a human being or human system, etc. [Ran00]. A simple approach would be to describe the system and its users and relations as a system-of-systems. This provides a good context to talk about the failure. Each related system could act as an independent judge, thus yielding different and subjective views of the failure. This view allows for a non-uniform identification of failure: different systems could have different expectations about the system in question. Furthermore, such judgemental systems may misreport or themselves fail in the judgement of another judging system. This subjectivity in observation and judgement means that failure itself is not an absolute notion [Ran00, Jon03], hence the need for a precise description of the failure and systems in question.

To avoid arbitrary judgements, Jones [Jon03] suggests using formal specification to describe a system—the failure then is a *deviation from the said specification* (see also further discussion on specifications and top-down development in Chapter 3). Note, however, that the specification may also be faulty and describe the system function inadequately [ALRL04]. Further ideas about this are explored in Chapter 4.

### 2.2.2 On errors

When analysing a failure, one needs to try to identify the incorrect system state (the *error*) leading to this failure. For example, consider a waterway used for ship transport. In some event, e.g. a dry season, the water level may become too low for the big ships. This erroneous state of the transport system may lead to a big ship running aground—a system failure. Note, however, that the error state may not lead to an actual failure—it may be *latent*. If no big ships use the waterway during the duration of low water, the error state will not manifest a failure.

The identification of an error state is necessary for determining applicable measures of handling the said error. A taxonomy of errors and handling measures in dependable computing is given in [ALRL04]. Most of them can be adapted for infrastructure systems as well. For example, the low water error could be *compensated* for by disallowing big ships from using the waterway and providing an alternative route to destination (redundancy). Another solution could be to try *rolling the error state forward*—recovering the system to a new correct state, e.g. lowering the water usage in surrounding areas, redirecting water from other sources or adjusting water level using sluices. The *rollback* recovery would be less frequent in infrastructure systems than it is in computing. The nature of digital data allows a computing system to be easily rolled back to the last good backup or to revert a transaction. For infrastructure systems, however, the physical components and events can rarely be *undone* and instead require fixes and adjustments—forward recovery.

The same failure can be caused by different error states. Therefore it is important to identify all possible error states, because there can be different strategies for handling them and thus preventing failure. Continuing the earlier example, the failure of a ship running aground can be the result of an accumulation of some obstacles on the bottom of the waterway. The error state (waterway becoming obstructed) requires different recovery strategies than earlier, e.g. clearing the bottom of the waterway.

Identifying and describing the different error states allows devising appropriate strategies of error recovery. Note that this is a different activity from dealing with *failures*: failure is a consequence of the error state and dealing with it means addressing the faults and errors it activates during failure propagation (Section 2.3).

### 2.2.3 On faults

Faults are the causes of error states in the system. Similar to the relationship between error and failure, there can be different faults causing the same error state. Faults represent the problems or weaknesses in the system or external events causing the error state, etc. Addressing the faults may eliminate certain failures altogether.

The notion of *fault* encompasses all possible causes of errors and then failures, which have been subject to attempts at classification in different ways [ALRL04, Kop11]. Adapting and reusing such classifications for infrastructure systems could provide an exhaustive list of things to identify and address when reasoning about dependability of such systems.

Avižienis et al. [ALRL04] recognise three major partially overlapping classes: development faults (occurring during system design, construction or deployment), physical faults (includes all possible faults that affect hardware) and interaction faults (all external faults). The thesis will use different examples of faults throughout, but focuses most on *development* faults, which should be addressed before the system is operational.

Faults that cause error states in the system are *active* ones. However, faults can be *dormant* in the absence of events that cause the error [ALRL04].

In the context of real-time embedded systems (which infrastructure systems can be considered to be), Kopetz [Kop11] categorises faults in two dimensions: according to their *space* and *time*. Fault space can be internal and external to the system (component) in question. Internal faults are either physical (e.g. break in a wire), or design faults, either in software or hardware. External fault are physical disturbances (e.g. a flood causing problems in the power supply) or provision of incorrect input data.

The spatial aspect of system faults can be tricky to reason about and identify appropriate relationships. These can encompass failure in one system affecting another one due to spatial proximity (e.g. explosion in power plant damages nearby transport link) or multiple nearby systems can be affected by the same external fault (water from nearby river flooding adjacent power plants). Kopetz [Kop11] recognises that embedded systems are normally designed in a way that spatial faults are contained within a single system, thus limiting the scope of such faults. For example, a flood in a nearby river would only affect one power plant if by design there is not another one built right next to it.

From a time perspective, faults can be *transient* or *permanent*. Note that design

faults are always permanent: a software bug cannot appear temporarily. A transient fault is one that appears for a short interval and does not require a repair action after its activity. An example would be the wearing out of electronic hardware—corrosion damages the system, but has not developed to a state to fail the hardware permanently. A preventive maintenance action should be performed to avoid a permanent fault of the system [Kop11].

A permanent fault requires a repair action to remove. A permanent external fault would be a lasting lack of service by a dependent system (e.g. a power supply). Physical problems in hardware or software are categorised as permanent internal faults.

The role of faults is especially important in failure propagation. When linking systems together, a failure in one can activate a fault in another, thus causing the failure propagation (see next section), and addressing the faults improves dependability of the system and the overall system-of-systems. Note that achieving a nearly fault-free system is an activity requiring possibly great cost and effort. The issue of system dependability vs. its cost is explored further in Section 2.5.

## 2.3 Failure propagation

In a system-of-systems scenario, a failure in one system may manifest as a fault in another (dependent) system, which could cause an error and lead to a failure of that system. Such failure propagation can be described in connected fault-error-failure chains. An abstract example follows (similar examples in [Mas06] and [ALRL04]).

Consider two dependent systems $A$ and $B$, where service $S_B$ depends on a correct service $S_A$ from system $A$. System $A$ may have faults, which are dormant during normal operation. Some external or internal event may activate a dormant fault $T_A$, leading to an erroneous state $E_A$. At the system boundary, this error $E_A$ manifests as incorrect system service—failure $F_A$. Since system $B$ depends on a correct service $S_A$, the failure $F_A$ acts as an external fault $T_B$. Note that in an alternative case, failure $F_A$ may in turn activate a dormant fault $T'_B$ in system $B$. Following the fault-error-failure chain, this fault $T_B$ can produce error $E_B$, which may yield incorrect service $S_B$—failure $F_B$. Figure 2.1 illustrates both cases of this failure propagation scenario.

A failure describes a consequence of a fault-error-failure chain. The incorrect service is the result, so to prevent such a failure, one needs to identify the error

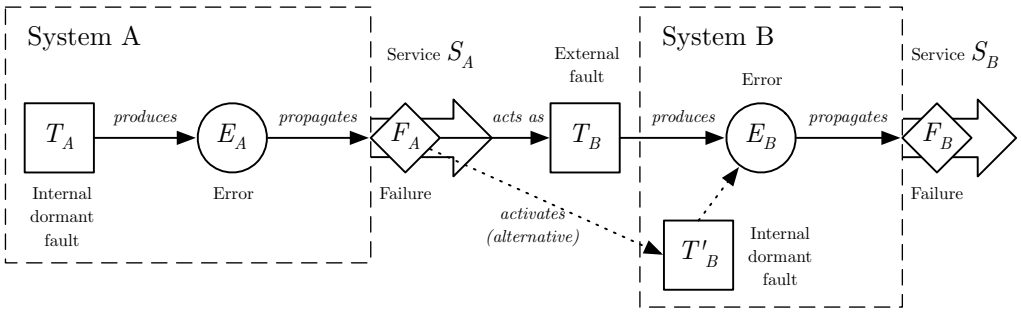


Figure 2.1: Failure propagation scenario.

leading to it and then the fault origin. In a system-of-systems, where failure propagation can occur, it is important to recognise how the failure propagates: what faults are activated/affected at each system boundary. By having a clear understanding which system each step in the failure propagation must be attributed to one can trace the context of original fault that led to the eventual failure [Jon03]. It is moreover essential to remember that whether a failure has occurred is in fact a judgement made in another system.

It is suitable to describe systems at the level of their boundaries. This can help clearly identify the jump from failure to fault. The failure is thus described in the context of its system. By agreeing on this context, one can start investigating measures to prevent this failure from propagating into the other system. Note that this raises a question of which system to handle the propagation on: if a fault is being activated in the dependent system, it could be fixed to prevent the failure from activating this fault; or if the failure itself becomes an external fault, then it must be prevented, e.g. by correcting the error state or eliminating the fault causing it.

Note that for complex system-of-systems, the boundaries are rarely exhaustive and fully described. More commonly, a system boundary may be represented by different views on the system. Full discussion on system boundaries and structure and how a complex system-of-systems is linked together is available in Sections 3.3 and 3.4.

## 2.4 Human role

The human element can have a large impact on system dependability. The proportion of faults caused by human interaction can be significant[2]. This thesis supports a view that good system design can help reduce human interaction errors by providing an intuitive and convenient environment for operators. Human operators are often attributed with the failure because their activities are the last steps in the failure chain and they are easier targets to blame than addressing the underlying issues [Rea00, RV06]. Unfortunately, changing human behaviour is rarely an option, yet designing appropriate support systems and operator environments is in the power of the system developers.

Human failures are inevitable—even on familiar tasks, humans are prone to slips and lapses. Some statistics show that 60% of human errors are on (familiar) "skill-based" automatic tasks, while difficult ones constitute the remaining 40%: 30% on "rule-based" reasoning tasks, 10% on "knowledge-based tasks that require novel reasoning from first principles" [Rea90].

The inevitability of human fault hints that systems dependability analysis must consider that humans will fail. Reason [Rea97] states that "human error is a consequence not a fault" and by understanding the context that led to error one could try to limit its recurrence. It is important to include all relevant contributing factors, e.g. supervision, training, procedures and equipment into analysis in order to find underlying reasons for human error. Thus in failure analysis, focus must be directed to latent conditions and situational contributions to the error, instead of personal ones.

Furthermore, judging from the proportions above, system design could include machine support for the repetitive, automatic tasks. When designing a system, the human operator interaction could be included in the dependability analysis, especially for these automatic tasks. This would allow for design of fault-tolerance measures for the human tasks, e.g. computer-based assistants to train or car drivers (recognition of incoming obstacle, a missed sign, etc).

The difficult tasks (knowledge-based or requiring complex interaction) are much harder to include in dependability analysis or to design fault-tolerance for. However, they constitute a much smaller part of human errors. Some of the human interaction accidents come from failure in following instructions. However, human systems can

[2]Different studies report 50-80% of accidents being attributed to operator errors and other human factor causes [Per84, RV06].

also use that to the advantage of overall system dependability. The ability to judge the situation and their actions independently and to react outside the set rules to avoid failure has a number of benefits [Jon05]. There are examples illustrating operators ignoring procedure to avoid accidents, as well as ones describing how following prescribed rules resulted in a failure [Lev11].

As mentioned in earlier sections, this thesis proposes to incorporate humans as systems in the overall dependability modelling and analysis. This provides a generic framework and allows reusing dependability ideas for human interaction modelling. Avižienis et al. [ALRL04] classify human interaction problems as external faults to the system. In the proposed system-of-systems approach, the human system is just another component system and thus falls into the standard failure propagation framework: human interaction could be an external fault and should be handled in the human system; or could activate a dormant fault in the dependent system and should have fault-tolerance designed for it. Furthermore, by expanding the scope of dependability analysis, one could include failure propagation between human systems, as well as adding other factors, e.g. *training* or *management* systems. The configuration of the overall infrastructure system would have assumptions on human activities to ensure that both humans and infrastructure are properly deployed [Jon05].

## 2.5 Dependability vs. cost

The view taken in this thesis is that systems that are fault-free in absolute sense, are impossible to achieve within limited resources of a real-world scenario. This leads to acceptance that faults in systems are inevitable and must be included in reasoning and analysis about systems.

A high level of dependability can still be achieved by implementing certain fault-tolerance measures (e.g. adding redundancies, resource buffers, etc.). Development of highly dependable systems, however, can increase costs enormously, take vast amounts of time to complete and even exceed allocated and available resources. Regardless of all such efforts in improving fault-tolerance, systems may still fail due to unforeseen faults or unexpected conditions. Factors such as limited current knowledge to anticipate future faults and other uncertainties, as well as a system's complexity outgrowing the scope of understanding can prevent us from achieving desired dependability. As a result one cannot talk about the absolute dependability

of system, but instead must reason about the *trust* in the system, which can be described by *acceptable dependability* [ALRL04].

The notion of acceptable dependability should be considered both when designing (modelling) a complex system $A$ itself, and when including the other systems $D$s that the designed system would depend on. So one cannot assign a specific dependability to the designed system, but instead should define the acceptable dependability to account for the uncertainties mentioned above. The same should be done for the systems that provide a service to the system in question. The dependabilities of these needed component systems should be judged and included into the system analysis at a certain acceptable level.

Modelling the dependency systems $D$s with acceptable dependability is problematic because of the additional uncertainty. The designed system $A$ itself is better understood and thus its acceptable dependability can be evaluated more precisely. Evaluation of acceptable dependability for $D$s systems requires information about them. The sound way of arguing about the dependability of these component systems is to refer to their specifications. However, one may not be available for the system in question, or the specification may be inadequate. Alternatively, one could try to avoid the component systems and their uncertain dependabilities altogether. A solution would be to create a self-sufficient system to ensure that the designed system—the whole system-of-systems—meets the required dependability level. For example, one could design a self-sufficient telecommunications node that generates all electricity required to power itself. This case would see reduction (or even elimination) of the dependence on the energy grid. The cost of a such system would increase significantly, however in some high dependability systems such cost may be justified. The cost, however, is a deciding factor in many industries. The common practice is to use off-the-shelf or existing systems, which are cheaper to integrate and use, rather than creating custom self-sufficient solutions, but they must be included with acceptable dependability.

The cost of achieving high dependability should not be evaluated on its own. When considering failures in systems with acceptable dependability, one has to talk about their probabilities and thus make decisions about the system's dependability using statistical methods. Therefore the cost of high dependability should be weighed against the possible cost of failure and the probability of its occurrence. High dependability of a system is certainly justified in some systems, where failure would incur high costs: lives of people (e.g. in safety-critical systems), direct damage

to the system, lack of important service, expensive recovery procedures or damaged reputation of the organisation. In non life-threatening scenarios, however, the cost of redundancy in a system (e.g. one of a "spare" power station) may significantly exceed the costs of just letting it fail occasionally. For example, one should consider that users may accept an occasional electricity shortage, e.g. in remote areas with low population, in order to receive the electricity service at a lower cost. In the end, one should always consider the trade-offs between dependability and cost to achieve it.

The full cost of failure should also include considerations of failure propagation occurring, which can increase the cost greatly, and considerations on the cost of redundancy and system recovery. An in-depth study in evaluating the monetary cost of a data centre downtime has been undertaken in [Eme11]. The authors associate the vulnerabilities of the data centre and failures of dependency systems such as power, cooling or monitoring with the costs of data centre downtime. They argue that while added redundancy incurs additional costs (they still need to repair original equipment failures), the always-available backup prevents failure propagation leading to disrupted data centre availability and thus substantial indirect and opportunity costs to the organisation.

Another aspect worth noting here is the need for investment prioritisation as well as the time dimension when such an investment should be made. The need for prioritised investment of national infrastructure systems and solving cost/benefit optimisation problems appeared in [Tre11] as well. A further discussion is given in Section 4 when investigating a system's design/planning stage as another system in a complex system-of-systems. The investment into a system's dependability should target underlying issues of failures in complex systems. Analysis of failure propagation between infrastructure systems could help find the real cause of a problem. Then it should be fixed instead of dealing with the consequences of propagated failure (e.g. adding redundancies when the problem should actually be fixed in the originating system). An interesting view appears when considering failures propagating from human systems. This thesis emphasises that in most of the cases, human error is a consequence, not the cause of failure (see Section 2.4). Therefore, the investment should address the underlying problem: fixing the working environment, training or management.

When investigating complex infrastructure system as a whole (as system-of systems), one could find alternative solutions to some of the problems, such as high

failure occurrences. For example, consider high frequency of road accidents in certain area. The usual direction would be to improve the traffic controls or the road network, but alternatively, one could invest in railways or ICT to improve home working, thus easing the pressure on the road traffic systems.

To summarise, one should not talk about system dependability without considering one's willingness to pay for it.

## 2.6 Application to infrastructure systems: failure propagation study

The parallel between dependability modelling and analysis in computing systems and infrastructure systems can be easily established. The complexity of both systems, the role of humans and the interaction of different component systems raise similar problems, especially when concerned with how such systems fail or how to prevent these failures. This supports the hypothesis in this thesis that concepts from dependability analysis in computing can be reused beneficially in infrastructure systems.

This chapter has introduced the basic notions of fault, error and failure and their descriptions in the computing literature. The examples aimed to show how they would appear in the context of infrastructure systems (see the "*ship running aground*" example in Section 2.2). Furthermore, this chapter touched on the issue of failure propagation, recognising that failures in one system can activate (or act as) faults in a dependent system, thus potentially leading to another failure.

This section analyses a more detailed example of failure propagation. It uses the fault/error/failure notions to describe an accident involving several interdependent infrastructure systems. The study describes a blackout of a major telecom node in Rome, Italy, which happened in January 2004 [CMS+07]. The telecommunications node was flooded and a blackout occurred causing problems and delays in different infrastructures including the electrical distribution network.

A report describing the accident is available in [CMS+07]. In short, a pipe burst in the air-conditioning system of a telecommunications node and flooded the node devices. To fix the pipe, the air-conditioning was turned off by the technicians, which caused overheating of the whole node. Furthermore, redundant power systems failed due to the flooding, taking the telecommunications node offline. The node failure resulted in overload of the whole telecommunications network. Also, it shut down

all communications links between control stations in the power grid.

This example is decomposed into fault-error-failure chains happening in different systems. Each such chain is described using a table that outlines every step below. Furthermore, the failure propagation overview is provided in Figures 2.1–2.4: first, the *cooling* and *power* subsystems are illustrated separately and then the full propagation is shown in Figure 2.4.

The report describes a pipe burst as the start of the problems leading to communications blackout. Table 2.1 tries to identify the reasons for the pipe burst and flooding. The identified system is chosen to be *Plumbing*, because flooding is a failure of the water transport service. Note that depending on the level of abstraction, one could choose to have the whole *Air-conditioning plant* as the target system. However, then the definition of intended service for the air-conditioning plant would need to include a clause that "*air conditioning does not damage surrounding environment*" or something similar so that the flooding failure describes a deviation from that service.

Table 2.1: Fault-error-failure for *plumbing* system.

| System: | *Plumbing subsystem of air-conditioning plant* |
|---|---|
| Fault: | *Corrosion of pipes (transient internal fault).* |
| Error: | *Pipe wall is too weak.* |
| Failure: | *Pipe bursts and floods the telecommunications node.* |
| Failure: | *Plumbing fails to provide water to the air-conditioning system.* |

The identification of the fault could raise questions about how to address it. "*Corrosion of pipes*" is a transient fault and maintenance activities can prevent it from becoming permanent. Thus if the analysis included further systems, e.g. maintenance of the node, the source of the failure propagation can be traced there.

The failure of the plumbing subsystem starts the failure propagation described in this scenario. Figure 2.2 follows the propagation within the whole cooling system. The figure provides an overview of the main failures and faults in the illustrated parts of the chain. The errors are excluded from the figure for brevity. The graphical notation used is similar to the one in Figure 2.1. The next steps of failure propagation are described below.

The main problem identified by the report is that in order to repair the pipe burst, technicians had to shut down the air-conditioning system. Table 2.2 describes the issue leading to the air conditioning failure.

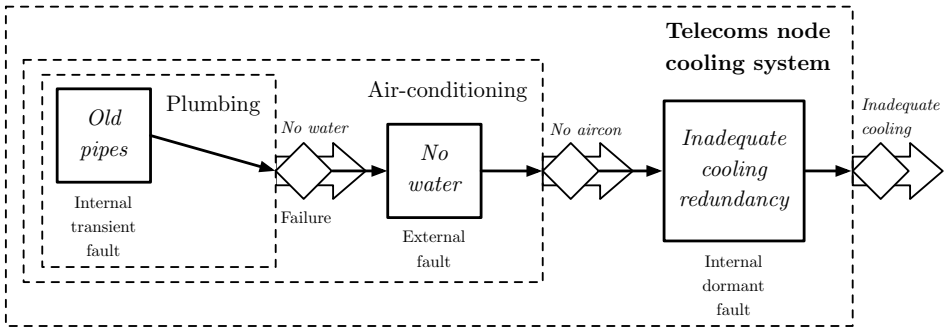


Figure 2.2: Failure propagation within *cooling* system.

Table 2.2: Fault-error-failure for *air-conditioning* system.

| System: | *Air-conditioning plant* |
|---|---|
| Fault: | *Water is not supplied (external fault from plumbing subsystem).* |
| Error: | *Shortage of water in the system—system is turned off.* |
| Failure: | *Air conditioning is turned off for too long.* |

The need to shut down the air-conditioning for repairs should have been planned into the system design. To avoid a single point of failure, the overall cooling system had to have alternative cooling solutions. Table 2.3 identifies this as a development fault in the planning system—lack of redundancy in case the air-conditioning system is shut down.

Table 2.3: Fault-error-failure for *cooling* system.

| System: | *Cooling system (only air-conditioning plant in the telecommunications node)* |
|---|---|
| Fault: | *Lack of redundancy in case air-conditioning system is shut down (development/planning fault).* |
| Error: | *All cooling systems are not operational.* |
| Failure: | *Temperature is allowed to rise too high (cooling is not adequate).* |

Table 2.4: Fault-error-failure for *telecoms node* system.

| System: | *Telecommunication node* |
|---|---|
| Fault: | *Temperature is too high (external fault—failure propagation from cooling system).* |
| Error: | *Node devices are overheated.* |
| Failure: | *Node devices stop working.* |

The flooding in the telecommunications node took out the main power supply as well as prevented startup of the diesel generator, which was designed as a redundant power source. A second redundancy was provided by a battery, which was exhausted quickly. Figure 2.3 illustrates the main points of the failure propagation within the telecommunications node power supply. The detailed descriptions of fault/error/failure for each system are provided in Tables 2.5-2.7.

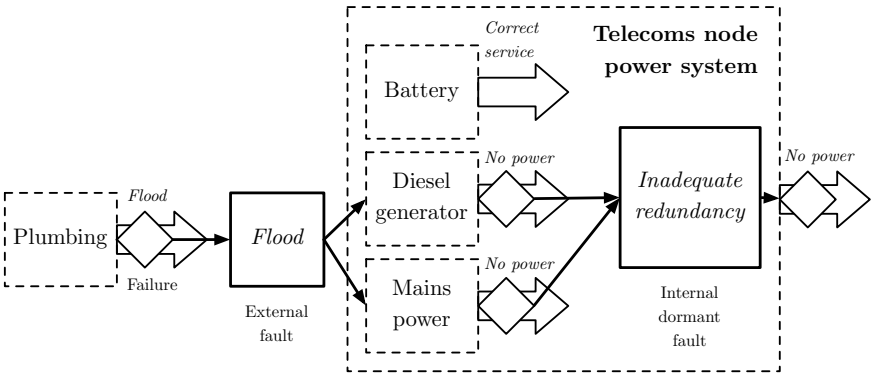


Figure 2.3: Failure propagation within *power* system.

Table 2.5: Fault-error-failure for *mains power* system.

| System: | *Main power supply from the power grid* |
|---|---|
| Fault: | *Supply cabling is flooded (external fault that is the failure of plumbing system).* |
| Error: | *Power supply system short-circuits.* |
| Failure: | *No power is supplied.* |

Table 2.6: Fault-error-failure for *diesel generator* system.

| System: | *Backup diesel generator* |
|---|---|
| Fault: | *Generator is flooded (external fault that is the failure of plumbing system).* |
| Error: | *Water in the generator.* |
| Failure: | *Generator fails to start.* |

Table 2.7 talks about the overall power system, which was inadequately designed or configured to allow for double redundancy to fail.

Failure of the telecommunications node affected dependent systems. Figure 2.4 illustrates the main points of the overall failure propagation within the scenario.

Table 2.7: Fault-error-failure for *all power* system.

| System: | *Whole power system (grid, diesel generator, battery)* |
|---|---|
| Fault: | *Inadequate dependability of power system configuration.* |
| Error: | *All power systems are not operational (power grid out of service, generator failed to start, battery exhausted).* |
| Failure: | *No power is supplied.* |

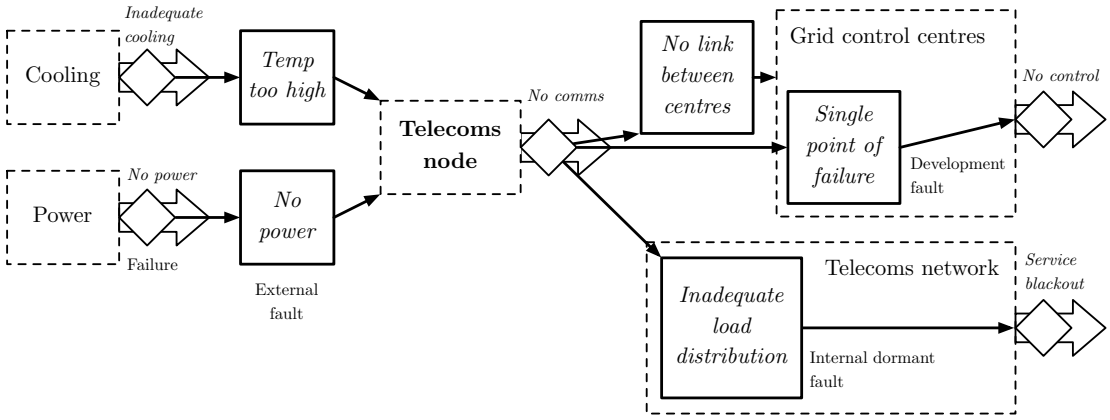


Figure 2.4: Failure propagation within the scenario.

The outage of the telecommunications node had effects on the overall telecommunications system, which resulted in disruptions and lack of availability of fixed and mobile telecommunications networks (Table 2.8).

Table 2.8: Fault-error-failure for *telecoms network* system.

| System: | *Telecommunications network* |
|---|---|
| Fault: | *Telecommunications load distribution problems (fault activated by telecommunications node failure).* |
| Error: | *Other telecommunications nodes in the network are overloaded.* |
| Failure: | *Telecommunications service blackout or disruption.* |

One of the consequences of the telecommunications node outage was the complete unavailability of two redundant communication links between power grid control centres. One of these centres is unmanned and controlled remotely from the other one. The lack of a communications service affected the power grid substations linked to the disconnected control centre—they had no control commands (Table 2.9).

The fact that a single point of failure was overlooked when deploying the supposedly redundant communication links between SCADA control centres should be

Table 2.9: Fault-error-failure for *grid control* system.

| System: | *Power grid control system (SCADA)* |
|---|---|
| Fault: | *Telecommunications service between two control centres is not available (external fault—telecommunications node failure).* |
| Error: | *No communication between control centres.* |
| Failure: | *Unmanned control centre and its managed substations are not controlled.* |

treated as a development fault (Table 2.10).

Table 2.10: Fault-error-failure for *grid control* system.

| System: | *Power grid control system (SCADA)* |
|---|---|
| Fault: | *Single point of failure (node outage) for redundant communication lines (development fault).* |
| Error: | *No communication between control centres.* |
| Failure: | *Unmanned control centre and its managed substations are not controlled.* |

This example illustrates failure propagation between interdependent infrastructure systems. Some of the faults and errors are speculative given the available literature, however they aim to show the general idea and not strive for precision regarding this specific accident. The identification of faults causing the failure propagation allowed description of development faults that need to be addressed to avoid similar accidents in the system.

...

This chapter is important to infrastructure interdependency analysis because infrastructure systems are often strongly coupled (energy depending on ICT for control, and ICT depending on energy for power, etc). By identifying how the fault-error-failure chain spans such systems, one can start reasoning about and designing against the failures.

The next chapter continues the topic by focusing on description, modelling and construction of such complex systems. The interdependencies between systems can be described with improved precision by defining system boundaries and assumptions. Furthermore, the system can be constructed with explicit fault tolerance (or records on its vulnerabilities) by describing it abstractly and then introducing details as needed, but ensuring that abstract requirements hold.

# Chapter 3

# Top-down approach and formal methods

The top-down approach to systems development, especially supplemented with formal methods techniques, facilitates abstraction of key concepts in a system, emphasises the role of system's a structure in its dependability and allows for a focus on important dependability properties during the design and analysis processes. This is especially important when considering complex systems-of-systems, as it allows talking about key dependability properties and relations without getting lost in the low-level details.

This chapter focuses on such systems-of-systems dependability. These are systems which can be considered systems-of-systems recursively. At any depth of decomposition of such a system, it can consist of further component systems-of-systems or be considered as atomic. The determining factor is whether the contents of the system are important or at least interesting to the analysis at the chosen level of abstraction. For example, when considering a human system (e.g. some organisation), one may be interested in it as a whole, in relations between its departments (next level), or even roles of specialists or specific workers (a further level). However, the level of a single person could be considered atomic, as the decomposition into internal organs is no longer relevant to the dependability analysis.[1] Furthermore, this thesis does not restrict analysis to one kind of system and considers complex systems of different natures, e.g. human systems, technological, civil engineered systems and so on. The main objective here is the same: understanding (and designing against) failure propagation from one system to another one.

[1]Unless, for example, one would need to ensure that an airplane pilot's heart monitor device does not interfere with plane controls.

The top-down perspective enables one to see an uncluttered big picture of the system in question. One could reason at the abstract level by omitting irrelevant components and restricting the details of components to only the parts that are required at the specific level of abstraction. With a limited number of concepts to manipulate, one can benefit from further techniques, such as formal methods, to construct a rigorous argument about the system.

Formal top-down development is an active research area for development and analysis of complex or high-integrity software and other computer systems [WLBF09, in particular Sect. 7.1]. The H2 hypothesis raised in this thesis suggests that these techniques can be applied to good effect to national infrastructure systems. The top-down view can provide a clearer description of the systems and hence their interdependencies and failure propagation.

This chapter investigates several avenues of taking a top-down view to (infrastructure) systems dependability analysis. Note that the thesis does not aim to be exhaustive and provide a fully-developed methodology of applying top-down development to infrastructure systems. Instead, the aim is to investigate and suggest directions in this area that could lead to improved dependability and understanding of complex infrastructure systems.

This chapter starts with a discussion of the importance of assumptions, system boundaries and the system structure to its dependability. Then it argues that formal methods and formal specification allow defining and reasoning about a system's assumptions, boundaries and its behaviour in a rigorous, mathematically-based manner. Finally, a proposal on how one would go about applying such techniques to a specific infrastructure system is provided.

## 3.1 Analysis and design of infrastructure systems

The concepts and techniques presented in this thesis are concerned with both the *analysis* and *design* of future infrastructures. While the proposed approach is generic and thus applicable in both of these areas, the activities in analysis and design differ slightly.

The aim of infrastructure *analysis*, as described in this thesis is, in particular, to identify flaws of the current existing infrastructure systems and how they affect other systems (see Section 5.2 for an overview of other existing approaches to analysing failures in complex infrastructure systems). Evaluation and assurance of existing

infrastructure dependability is achieved during analysis activities: clearly identifying the concepts and their lifecycles within an existing system, i.e. faults, errors and failures; identifying system boundaries and compositional structure; describing (preferably formally) the system services and relationships with other systems; identifying and recording external and internal assumptions about the system. By collecting these descriptions within some (formal) framework would allow constructing a specification of the existing system. The application of a top-down approach to such descriptions would help tackle the overall complexity. Further techniques can then allow verifying the specification to identify flaws within the overall system, e.g. whether the low-level details of the system actually correspond to the system description at the higher level. A rigorous description by itself would help identify gaps in the system relationships.

The *design* of infrastructure system can involve either improving existing infrastructures, planning new extensions—up to designing full infrastructures from scratch. All these activities would benefit highly from employing the top-down development approach presented in this thesis. When designing a *new* or improved infrastructure system, the design activities would follow a top-down approach: first it would be identified what is expected from the system, its functionality as well as required dependability. Then it could be refined to an actual system that would be built by introducing implementation details step-by-step. The process must ensure that the original specification and dependability assurances are preserved. During the process, the activities would be similar to those of analysis: from precise definitions to the overall verification. The existing infrastructure considerations would be included in the specification and reasoning, e.g. as given constraints.

The analysis and design of infrastructure systems are closely linked together. Analysis identifies flaws within existing infrastructure systems and the next natural step is to design upgrades to address them. The design process cannot ignore the importance of analysis of the existing systems—the results would shape the new system. Therefore the overall planning of future infrastructure systems would be a synthesis of analysis and design techniques which cover existing and new systems.

## 3.2 Assumptions

Assumptions are an inevitable part of any system design and are critical in reasoning about system dependability. This thesis recognises assumptions as a key concept

and explores their role in system design. The fact that system dependability is reliant on the validity of its underlying assumptions is particularly emphasised.

Assumptions are a way of dealing with uncertainties or lack of knowledge—not only in system design, but in day-to-day activities and situations. As a routine example, take travelling using public transport. When planning a journey, one frequently assumes that even though the schedule may not be kept to precisely, the general availability, approximate frequency and other conditions will hold (e.g. a bus will arrive within a 15 minute interval). The reliability of "arriving at the destination on time" holds as long as the assumptions are valid (e.g. the bus does not get into a traffic accident).

In general, assumptions are a certain subset of different future scenarios, which one considers most likely. By selecting this subset of scenarios, the context for a decision is established and one can reason about decision within this context (e.g. by assuming that there will be a bus every 15 minutes).

Assumptions allow us to focus on the important parts of a problem and not explicitly model (include in the decision) real world components. Consider the model for the decision on when to wake up to catch the bus. For an exhaustive model, one would need to include all possible scenarios of the bus trip, including all of its attributes (e.g. there is no public transport strike, the driver is feeling well, there is no car blocking the road, etc.). By making an assumption that there will be a bus within 15 minutes, one describes only the dependency and not the full model of the bus behaviour.

Assumptions allow the shaping of the model and the design process of the system. One must take care, however, to avoid assuming a model of a system that is too simple and does not reflect reality (whether the model captures the relevant properties of the system). The model should be simple, but not too simple—paraphrasing C. A. R. Hoare[2] or A. Einstein[3].

Making erroneous assumptions may result in a fault in the system, which may lead to a system failure. The choice of assumptions should be justified and informed. Because of limited knowledge and skills, the assumptions may still be unreliable. It is argued that it is essential to write down all assumptions explicitly. This way the system is built on explicit assumptions and one may trace decisions down to the

[2] "There are two ways of constructing a software design: One way is to make it so simple that there are *obviously* no deficiencies, and the other way is to make it so complicated that there are no *obvious* deficiencies."—C. A. R. Hoare [Hoa81].

[3] "Everything should be made as simple as possible, but no simpler."—attributed to A. Einstein.

assumptions. Written assumptions can be subjected to review, external certification or other tools in order to check their justification, validity, correctness and applicability. In this way faulty assumptions be found and addressed in a manner similar to other system faults. Only by writing the assumptions down can one subject them to the rigour of the whole system analysis.

A major danger is that designers make some assumptions without even realising (and thus recording) them. The field of safety cases (analysis) specifically tries to detect unwarranted assumptions. Safety cases use structured arguments and evidence to argue about safety and (via extension [DK04]) dependability of systems. Kelly and Weaver [KW04] emphasise the crucial role of both argument and evidence in establishing the safety case: argument without supporting evidence is unfounded; evidence without argument is unexplained. The argument and evidence are defined within a particular context, which can be established as a set of assumptions. However, without defining the context, arguments and evidence cannot be established.

Great care must be taken in incorporating assumptions into analysis. They must be written down completely and unambiguously. Formal methods provide concepts and tools to establish assumptions with mathematical rigour, structure and layer them, as well as find contradictions and inconsistencies. They are discussed in more detail in Section 2.2.

The importance of explicitly identifying all assumptions can be observed in day-to-day activities as well. No two people are ever talking about exactly the same thing, because each bases his argument on his own assumptions. System design is highly influenced by designers' assumptions about how other people think and act, as well as by familiarity with the designed system. So when the system presents the user with a choice, it is often not clear what assumptions and consequences are associated with each option.

An illustration of a failure due to implicit assumption can be found in an airplane crash at Chicago Midway airport on the 8th of December, 2005 [Nat07]. The airplane crashed by overrunning a snow-contaminated runway. The runway length required to land safely in these conditions was calculated by an on-board computer, which worked on data provided by the pilots. The crew input the weather conditions and received a confirmation from the computer that the airplane would be able to land and completely stop with 560 feet of runway remaining. However, the computer was programmed to assume that the engine thrust reversers (a form of brakes) will be deployed immediately upon touchdown, which they were not—it happened

18 seconds later. This delay resulted in the airplane overrunning the runway and crashing.

The implicit assumption made by the program led to the pilots receiving an indication of predicted successful landing. The pilots were not aware of the assumption and thus could not adjust accordingly.

Assumptions can provide a trade-off between system dependability and its cost: one could assume that the system will operate in a safe environment and thus design a simple system without being concerned, for example, that a hacker attacks it. Based on such assumption, one may design a system that is more elegant, faster and cheaper. However, the system dependability would be directly linked to the trust in the assumptions it is designed on (system is dependable as long as the assumptions hold).

When defining assumption for a robust system, one could choose to disregard probabilities of the assumption holding (event happening) and consider every scenario to have a binary outcome: `0` and `1` , where `1` is used to indicate that there is some likelihood of scenario occuring, and `0` when it is impossible. This produces assumptions that if a failure can occur, it will. Such a "worst-case" approach requires reasoning about and implementing adequate measures for fault tolerance during the design process. For example, if a power plant is being designed that could be deployed near a river, one may want to reason about it being flooded. The likelihood of that happening may be very small (say 0.01%). Still, a robust model would require it to be treated as a hazard and included in the model—"*power plant can be flooded*"—with appropriate fault tolerance. Alternatively, one could choose an assumption that "*power plant cannot be flooded*". This simplifies fault tolerance, however this explicit recording of a design decision will need to be justified during deployment: ensure that flooding is not just *unlikely*, but actually impossible. Such treatment of probabilistic events leads to a robust system, however it also means that one has to use the same level of redundancy (prevention) even when the real likelihood is very small.

Note that to capture the actual system and real world environment in the model, one may consider defining assumptions and system properties with probabilities. However, one of the main problems of using probabilistic description is that in real life scenarios such definitions would need to be based on existing data: manufacturer information, empirical data of system in operation, etc. Such data is rarely available or accurate and frequently requires lengthy evaluation of the system to collect the

required data.

In general, the fault assumptions identified at the beginning of system design should shape the subsequent design activity [Ran00]. For example, assumptions regarding the type, probability and magnitude of system hazards shape system design in such a way that certain fault tolerance measures (strategies) need to be employed that tolerate the specified faults [Kop06]. The fault types and fault tolerance are discussed further in Section 2.2.

Assumptions are not limited to the initial stages of the design process. The underlying assumptions need to be re-examined when the system environment changes. This is particularly true for infrastructure systems, where the dynamic nature of infrastructure and its environment (natural, social, technological and political) should be addressed in making assumptions about such systems. The infrastructure systems are long-term projects and after certain periods of time the initial requirements can no longer be met because the assumptions do not hold anymore.

One of the tasks that the ITRC research project[4] investigates is development of decision procedures for future infrastructures given different (and changing) assumptions about the environment. The basic dependability (including balancing capacity and demand) must be re-evaluated in the context of new assumptions about changing social and economical (technological) environments, new hazards (and of increasing magnitude) and others. One particular area of ITRC concern is hazards arising from climate events—mostly focusing on flooding.

This section emphasised the role of assumptions in fundamental reasoning about system dependability. Assumptions create a context, hence should be included in reasoning. Moreover, they must be clear, unambiguous and justified. Most importantly, however, one should be explicit about any assumptions upon which the model or any reasoning depend.

## 3.3 On boundaries

When taking a top-down view of a complex system-of-systems, its inner workings can be omitted from the analysis. This way each system would be described on its boundary—by specifying its behaviour as interaction with its environment. Note that environment can be considered to be another system that may be comprised of further component systems.

[4] http://www.itrc.org.uk/

One system's interaction with another happens via its interface determined on the system boundary. Systems may have different interfaces to interact with various other systems. Moreover, there can be multiple interfaces between any two given systems. For example, consider a bank and its customers as two distinct systems. The banking system provides a variety of interaction capabilities for customers: internet banking, telephony, or face-to-face customer service in a branch. These could be considered as various interfaces of the banking system.

When modelling or reasoning about a system interaction scenario, one needs to specify the scope and boundaries of the systems in question. A system boundary comprises a collection of interfaces relevant to the reasoning activity—it is a viewpoint on the system. In theory one could try to incorporate all interfaces into a single boundary, however in practice and especially for complex systems it is probably infeasible and unnecessary (irrelevant in most cases). To control the scope included in the reasoning, irrelevant details are omitted. Therefore, when talking about one system interacting with another, it is not necessary to know about its other interactions and interfaces. For example, consider a car system: its driver interactions may include usage of the cruise control system, but do not need to include the engine repair activities.

System boundaries are not always clearly identified. Different perspectives (points of view) would define different boundaries of the same system, which may overlap [JR06]. Therefore it is important to determine the boundaries when reasoning about systems carefully. This is emphasised in the thesis. Absence of such agreement could render a discussion between two experts pointless, since they would have different system boundaries (and thus possibly different behaviours) in mind.

A system interface incorporates a set of assumptions on the environment and other systems, as described in the previous section (§3.2), which would also normally be written down at the level of the system boundary. If expressed formally, these assumptions and specifications would become the contract of a system's interface.

A system-of-systems is built from systems (components/entities) that interact with each other. The boundaries of the whole system and its components could be considered in a compositional manner. The component systems have their own interfaces, which describe their interactions with the environment and other systems. The behaviour of the whole system is then normally discussed by fixing boundaries of its constituent components and considering how the components are linked together via their internal interfaces [GIJ+02]. This way, the internal interfaces would not be

considered as part of the big system's boundary. However, the way the components are linked via these interfaces dictates the emergent behaviour of the whole system. Note that internal linking of component systems during the composition raises the key issue of such systems-of-systems: mismatches between interfaces [GIJ+02].

The interfaces between systems (and between the system-of-systems and its environment) serve as the main route for failure propagation. One can consider that interdependencies between systems constitute their interfaces. This thesis investigates failure propagation through system interfaces, as well as failures producing faults in other systems.

In conclusion, system boundaries depend on the specific viewpoint taken—boundaries are not universally determined for particular systems. They are an abstraction to frame one's reasoning. In any fruitful discussion it is very important to agree upon them and avoid shifting without explicit notification and agreement. The next section concerns the structure of different systems interacting with each other and particularly emphasises the role of system structure in failure propagation, i.e. how it is directly linked to system dependability.

## 3.4 On structure

A top-down approach allows one to step back and perceive the high-level structure of a system-of-systems. The previous section discussed the boundaries of systems. This section builds on this by talking about the structure of component systems that make up a larger system, linking together via these boundaries. The aim here is to understand the role of structure to system dependability and how structure could be used to restrict or prevent failure propagation. This view of system structure's role in dependability was proposed in [JR06].

The component system boundaries, their functions and interactions describe how each system fits in the structure of the whole system-of-systems. This can be applied recursively to describe structures of the component systems themselves. Jones and Randell [JR06] argue that it is the structure of the system that enables components to interact and thus determines the behaviour of the whole system-of-systems. For instance, disassembling a bicycle would take away its structure and yield a number of separate components without the bicycle-like behaviour.

When assembling a system, different structures can be chosen to achieve the same system behaviour. However, these decisions could have a strong effect on the

dependability of the system in question. The dependability of the whole system cannot be composed solely by analysing its component systems' dependabilities, but emerges from how these components are linked together, i.e. what structure has been chosen. As a trivial example, take a brick wall that is constructed out of a number of single bricks. If bricks are stacked one exactly on top of the other, a failure in a lower brick (e.g. removal of the bottom one) would bring down the whole column. Instead, if the bricks are laid *bonded* (e.g. using a Stretcher bond to cross the bricks) to increase stability, removing one of the bricks would not bring down the whole column.

A particular case where the structure of a complex system-of-system matters is in ensuring the absence of a *single point of failure*. This is a situation when a failure of a single component can stop the entire system from working. An ironic case would be when other components have high redundancy, but a failure of this single point can still bring the whole system down. Take, for example, a database server that uses multiple backup servers and replication to ensure availability if the primary server fails. However, all these machines are serviced by a single router to be accessed from the Internet. Even with multiple redundancy, a failure in the router (or, for example, of the Internet provider) would completely disable the database service provision.

Failure propagation happens over the structure of some systems. A failure in one component can disrupt its service and thus become faults in other components that depend on it (are linked to it). Section 2.2 saw a discussion that a fault-free system does not exist, and a fault in some component may create an error, which in turn can become a failure propagating to other components. The "Swiss Cheese" model of failure propagation [Rea97] suggests looking at systems as layers where failure propagates when latent faults and active failures align given the circumstances. Such cascading failure points to a weakness that needs to be identified.

The role of structure in the dependability of such systems is in restricting such errors from causing a failure. A structure governs component system interactions with each other by enabling interactions between some components while forbidding them between others (i.e. it creates a modular structure). This limits the failure propagation paths in the system. Then, on these paths, redundancies or other fault-tolerance measures can be established to stop the failure from propagating (providing means of *error confinement* [JR06]). One example of such error confinement is the physical structuring of watertight bulkheads used to create com-

partments in ships [JR06]. If one compartment is breached and filled with water, the rest of the structure (and ship's ability to float) is not affected—the error is contained. Similarly one can see the construction of a bridge to be error-confining. A bridge is built with a high level of redundancy, so that if one part fails the load is distributed through the structure and thus a single component is not able to cause the collapse of the entire bridge. Structuring in such way allows management of the failure propagation—establishing fault-tolerance. Note that large or very complex systems require additional care with their structure. Modularisation and fault-tolerant structures can be infeasible or introduce additional risks. There is a real danger that modularisation and component isolation may result in a faulty structure.

The structure of a system might be difficult to determine. This thesis considers systems of various natures, such as human, technological or civil-engineered. In some cases, the system structure may be clearer, e.g. when describing computer systems hardware or civil engineered systems. However, the view becomes less clear when talking about human systems (organisations) [JR06]. The "boundaries" of a single human specialist "system" could be described by to whom he reports (how information flows), what are his activities, etc. However, the documented responsibilities may not reflect reality, where the activities of a specialist can shift arbitrarily (e.g. covering for a colleague). This may be different across various human systems, e.g. military systems are obligated (and thus more likely) to follow rules and chain of command, while university (academic) systems are more ad-hoc.

Different deciding factors can figure in designing the structure of a system. Cost can be a limiting factor in real-life situations and is discussed further in Section 2.5.

In conclusion, it is important to include the structure concept in a system dependability analysis not only because structure determines system behaviour, but because it may help identify the single point of failure, weak links or patterns for failure propagation in the system.

## 3.5 Formal methods and formal specification

Formal methods are mathematically-based techniques to describe and reason about systems, in particular software and hardware computer systems. Formal methods aim to increase assurance in the system and achieve system correctness—mathematical techniques with tool support can allow identification of mistakes

and inconsistencies even before the system is tested. Furthermore, one can benefit from analysis and verification with mathematical rigour at any part of system's life-cycle: requirements engineering, specification, architecture, design, implementation, testing, maintenance and evolution [WLBF09].

Formal methods have mostly been used by academia in computer science domains. Historically, industry has found most use in development of critical systems, where very high assurance is required. An overview of the most recent state, challenges and benefits of formal methods' application in industry[5] is given in [WLBF09]. The survey highlights good effects of formal techniques on time, cost and quality of systems development: estimated three times reduction in required time and five times in cost as well as quality increase in 98% of cases (no reported decreases). The improvements were reported in detection of faults, improvements in design, increased confidence in correctness as well as improved understanding of the system being developed.

The increased quality and confidence can require high level of manual effort and appropriate skills and training to apply formal techniques and tools. The survey acknowledges, however, that increased levels of automation and computing power over the years will help with adoption and use of formal techniques in more mainstream projects [WLBF09]. Hinchey and Bowen [HB95] provide another survey on 15 different applications of formal methods in industry, detailing the experiences from telecommunications, nuclear power, railway, aerospace and other industries. A quite recent account of a large-scale formal methods application is available from the DEPLOY research project [RT13]. This research project aimed to introduce the Event-B [Abr10] formal method to industrial organisations in different domains: automotive, railway, space as well as business information sectors. The results agree with the survey in [WLBF09] that applications of formal methods is practical with significant improvements in quality assurance and development processes [RT13].

As mentioned above, formal methods encompass various techniques and can be applied in all parts of a system's life-cycle. Several possible aspects of use:

- Writing formal models and specifications of the system—various mathematical notations and concepts are available to allow precise and unambiguous descriptions.

[5]The survey covers mostly computer science industry applications of formal methods.

- Formalising system requirements, assumptions and overall description—use similar techniques to give precise meanings when discussing the system.
- Specifying properties and queries about the system based on its mathematical description.
- Formal rules for how to transform and refine the abstract model or specification to a concrete one—ensure that assumptions made at the abstract level still hold for the implementation.
- Verification using theorem provers, model checkers and other tools.
- Checking the specifications or descriptions for consistency—ensure that requirements are not contradictory, ensure that the described system is viable, etc.

For the purposes of this thesis, *formal methods* should be taken to mean formal specification and verified development of software and other systems. This section aims to introduce some aspects of this approach and hint at how one could benefit from using them in describing and reasoning about infrastructure systems dependability. One important benefit of employing formal techniques is an increased precision in defining dependability notions: fault, error and failure. Jones [Jon03] suggests that by having a formal system specification, one can reason about the correct service of the system and thus what would be considered a failure, error or fault in that system. Specification is a technical contract between system developers and clients to establish a common understanding of system behaviour. By having a specification, a subjectivity about system failure is removed: a system service or state deviates from the correct one when it is inconsistent with system specification. Dependability is defined with respect to a specified behaviour (e.g. a system's specification) [GIJ+02].

When employing a top-down approach to system development, one should consider and start with an abstract view and corresponding abstract specification. Formal methods are especially beneficial in preparing abstract formal specifications. They can provide a precise description of what the system has to do and postpone discussing of how it is to be achieved [WLBF09]. With abstract specification, system developers have a clear framework to work within but are free to choose the actual implementation details, as long as the abstract requirements hold.

Formal specification and modelling is the most widespread use of formal methods—[WLBF09, Sect. 3] reports this technique to be used by the majority of surveyed applications of formal methods. The specifications are first and foremost used during system design phase to record and clarify the requirements, to inform system design as well as identify and remove faults at the early stages. The specification can also be verified to ensure, for example, that intended safety properties indeed hold for the designed system. Subsequently, the specification can be used in the development and testing phases, e.g. the specification can be refined to a concrete implementation, some program code can be generated directly from the specification, the specification can be used to generate test cases, etc. These are some of the possible uses of formal specification—[WLBF09] provides a wider overview of important applications.

Formal methods and languages such as VDM [Jon90], Z [WD96] and others provide mathematical notations and concepts to write formal specifications. With a focus on abstraction, they provide description tools to describe system *contracts* and avoid unnecessary implementation details. This way, the system and its behaviour are described at boundaries. For example, system operations (transitions, user actions, etc) can be described using *pre-* and *post-conditions*: indicating only what is needed for an operation to succeed and what will hold afterwards without saying how it will be done. This allows describing interactions between systems in a system-of-systems as interface contracts. This can also be extended to interactions between a system and its operational environment—assumptions about the behaviour of *the world* can be recorded formally [HJJ03, JHJ07].

This approach to system specification has several benefits. First, the interactions between systems in question are clear and well-defined: all requirements, services and assumptions about each system are written down as the system interface (see also further discussion on assumptions in Section 3.2). Furthermore, interface contracts allow the internal system of constituent components (can also be systems) to change, as long as the contractual interface specification is respected [PBFR12, GIJ+02]. This also applies to system dependencies—alternative implementations can easily be swapped if they have the same interface contracts.

Explicitly specifying assumptions on the environment is important (see Section 3.2). Jones et al. [JHJ07] use *rely conditions* to specify physical world requirements for correct function of a software system. This allows avoiding the modelling the explicit of real world components. Rely-guarantee reasoning [Jon81] addresses

the concurrent nature of real-world interactions and allows specifying assumptions that are required to hold during some system operation or transition (*rely conditions*), as well as what system properties can be guaranteed during the said operation (*guarantee conditions*). This is an extension of the classical *pre-* and *post-conditions*, which are only concerned with *before* and *after* the operation, not *during*. In addition to specifying the assumptions about the physical world, Jones et al. [JHJ07] present a systematic way of deriving the system specification from these assumptions. This is a good illustration of the top-down approach, where at first only the very abstract interaction with the world is known and the further details are derived and introduced during the system development. Note that the real world is often perceived via sensors and therefore assumptions involved should consider issues with sensor service, e.g. data validity [MC10].

This thesis does not attempt to apply full formal development to infrastructure systems but instead aims to indicate how this could be done. For that reason, detailed introduction of formal specification and development concepts and constructs is avoided. The reader should refer to appropriate literature of the available formal methods (e.g. Vienna Development Method (VDM) [Jon90]) for more details. A brief introduction of the main concepts mentioned in this thesis follows.

Formal methods and languages like VDM [Jon90] or Z [WD96] frequently describe the system and its behaviour as a model of system *state* and *operations*. System state is used to collect a system's structure, main components and properties. *Invariants* are logical expressions about the system state and specify what is always true about the system. Note that one could have several system states, e.g. the *correct* one and various *error* states.

Systems are rarely static: they can have various transitions, user operations and other events. These can be modelled as *operations*, describing how the system *state* changes and its various inputs or outputs. To avoid specifying how the operation is actually implemented, *pre-* and *post-conditions* describe what is expected before system operation and is established after. Furthermore, rely-guarantee reasoning [Jon81] can be used to talk about what is expected and established *during* the operation.

Correctness of system development can be verified using further techniques, such as *data reification* [Jon90]. They are used to establish a formal relationship between the abstract specification and the concrete one. More concrete specifications can introduce additional concepts, provide details to operations and link all the way to

the actual implementation. Reification (also called refinement) rules require that concrete specification still adheres to the requirements and properties indicated in the abstract specification. Furthermore, this gives a formal basis to dependability notions, because it offers a precise notion of what it means for a system to satisfy a specification. The link between system $B$ and its specification $A$ is established by showing that for each operation described in the specification $A$, there is a corresponding system $B$ operation that respects the specification in $A$ [Jon03].

This thesis advocates at least a light use of formal methods in infrastructure systems, namely formal-like specifications and refinement ideas to describe the various systems-of-systems. The verification and associated tools can increase assurance further, but would require special skills. Formal specification, however, is worth considering as a starting point given the complexity of the systems, different viewpoints and the discussion in this thesis how important it is to agree upon and be precise about the analysed infrastructure systems. Note that while this thesis provides only a brief overview of what formal specifications are about, significant examples of formal specification use are available in the literature. For example, the *Tokeneer* ID station [CB08] describes an entry system to a secure enclave and its formal specification has been released as a demonstrator project of secure software engineering. The case study of the *Mondex* smart card system [SCW00, JW08] is also worth mentioning. Verification of its formal specification was set as an academic challenge and was done (including redoing the specification in many cases) using different formal methods and approaches.

## 3.6 Application to infrastructure systems

This chapter has investigated benefits and techniques of applying a top-down development approach to complex and critical computer and other systems. The hypothesis of this thesis proposes that taking such a development and analysis approach with national infrastructure systems would be advantageous in their design, analysis and reasoning about their dependability. This section now aims to demonstrate how such an approach could be applied to infrastructure systems: how it can be used to construct system-spanning arguments without getting lost in irrelevant details, provides hints on how one would build up a formal model of connected infrastructure systems, and offers some comparison with bottom-up approaches.

Historically, national infrastructure planning has been a sector-centred effort

that focused on addressing issues of one specific infrastructure sector. There were different methods tailored to reason about and deal with the specific sectors: energy, water, transport, waste, ICT and others (see Chapter 5). Following this, the current infrastructure sectors were built independently—lacking in analysis and insight of inter-sector dependencies. Modern infrastructures, however, exhibit high interconnectivity between each sector and therefore should be analysed in a holistic way. This requires arguments about national infrastructure as a whole (e.g. about its functionality or dependability) to span multiple sectors and include components from various infrastructure systems. To avoid taking every irrelevant detail into consideration, one needs to specify the level of abstraction to the reasoning. By taking a top-down approach, an abstract argument is first constructed, which is then refined to include more detail where necessary. The remainder of this section provides some views on constructing a model and reasoning about it for some abstract infrastructure system.

Note that applying top-down development to systems involving real-world phenomena requires additional caution (compared to applications to mathematical domains). Jackson [Jac82] argues even more broadly, that lack of competence in the early top-down development stages of *any* system can result in significant errors at low-levels of design. Top-down development requires taking significant decisions at very early stages. An erroneous decision at the abstract level would ripple through the refinement process and could result in a concrete design of a system that does not have any implementation options (or has significant obstacles and is not viable for implementation). Jackson argues that top-down development is more of a system "description" than actual "development", with the actual development already largely completed invisibly in the designer's head [Jac82]. Furthermore, abstract modelling provides more freedom than may be allowed by the available real-world implementations. Naive design decisions at the top level may describe a "perfect" system in an elegant manner, but would fail finding actual implementation that satisfies the conditions required by such design. For example, one could design a safe distance between nuclear power plants that would exceed the area that is intended to build them. Finally, caution must be adopted when describing real world objects using mathematical constructs to avoid oversimplifying them. Jackson [Jac00b] notes that formalising reality is always an approximation and is thus prone to errors arising from that. An example illustrating such a case would be assuming a switch having just the on/off states, while it can actually be on/off/"something in

between".[6] To avoid such issues in applying top-down development, the assumptions about the real world should be identified and considered at appropriate stages of development. Furthermore, expertise within the domain as well as in top-down modelling is needed. Still, as with any development process, mistakes may occur and rework of affected parts would be needed.

### 3.6.1 Role of existing infrastructures

Top-down development starts by taking an abstract view of a system. The level of abstraction is chosen to be as abstract as possible whilst still being able to discuss properties of interest. The aim is to identify and specify the required functionality and important properties about the system in question before writing the details of concrete realisation. For this reason, existing components (e.g. existing software in computer systems) are ignored at the abstract level. The idea is that the abstract specification of a system can be refined to various different concrete implementations. So at the abstract level, one should clearly describe what is expected of the system, what are its properties, etc., without being concerned, for example, with lower level physical constraints.

For national infrastructure, this raises the question of existing infrastructure systems. They do not allow one to start designing and reasoning from scratch. To take the top-down approach, one should ignore them at the abstract level and focus on the abstract functionality and dependability properties, ensuring a *correct* abstract specification. Then, during refinement of the abstract model towards a concrete implementation, the existing infrastructure could be shown to satisfy the conditions of the abstract model or specification, or adapted to match the new requirements. A planned reuse of existing infrastructure systems would involve including the specifications of said systems into the reasoning of the overall infrastructure design and showing that the abstract specification is adhered to, thus preserving the original correctness of the specification.

The idealistic approach of top-down development may raise concerns when dealing with existing systems. However, there is evidence from computing that such a thought experiment can actually be a practical approach. Conventional wisdom would say that some large software systems (often having grown quickly and without detailed planning) are so valuable that any thought of tackling them with a top-

[6]The three-state device in question is a valve closure mechanism involving a solenoid and a spring. The example comes from the Three Mile Island accident description [Fer92] via [Jac00b].

down approach must be dismissed. IBM's "CICS" (Customer Information Control System) was just such a result of minimally controlled evolution [HK91]. However, around 1980, the need to make major changes to the functionality of CICS meant that the mass of poorly documented code had to be brought under better intellectual control. The decision was made to undertake a formal specification. This was a large undertaking and was only made possible by a novel collaboration between IBM and an extremely strong group from Oxford university (with Rod Burstall and Cliff Jones as consultants). The result was a much cleaner structure and a Queen's Award for Technological Achievement to Oxford university [HK91]. The CICS example illustrates that very complex existing systems can be incorporated in the design of a new system using the top-down approach.

In spite of the huge investment in such software systems, it is certainly not claimed that this experience suggests that physical infrastructure systems would be scrapped and rebuilt in accordance with a plan that was devised top-down from a clean sheet of paper. The costs of the existing UK infrastructure for power, transport, etc are greater than software costs by orders of magnitude. What a top-down view can offer, however, is a fresh viewpoint and a goal towards which evolutionary changes can be directed.

Taking the idealistic "world planning" situation for the infrastructure systems, the top-down development would start at the abstract level by recognising national infrastructure as a single connected system. The shape of the model and direction of reasoning would be driven by which properties and requirements are interesting and important to the system analysis.

### 3.6.2 Purpose of reasoning and model

When starting a top-down development of a system, one needs to identify the purpose of the model being developed. This will then shape the level of abstraction and the included concepts within the abstract model. Some could purposes be the following: domain modelling of a system, describing and reasoning about some non-functional property (e.g. dependability) within the system, or defining functional properties (e.g. how the system is supposed to work)—writing down the specification of a system.

Domain modelling is used to improve understanding of a system by identifying its component systems and revealing its structure, how components are linked together, etc. When reasoning about national infrastructure systems, a domain model of in-

frastructure can be used to reveal, identify and formally specify interdependencies between systems. For example, using formal methods, one can specify boundaries of different systems with mathematical rigour, and then link them together, ensuring that all interfaces are matched. Then these interface matches become interdependency specifications.

When reasoning about dependability of a system, it can be described as certain properties holding over the model. For example, the failure propagation property could be formulated as "*a failure in one system does not activate a fault (all systems have faults) in an other system*". This property can be described formally over the whole infrastructure (spanning different infrastructure systems). Normally such a property would be composed of properties over different types of faults/errors/failures. Note that to reason about a system holding such a property, one must justify it for all systems defined in the model. Then the refinement of the model has to be done in such way that the property would hold for the more concrete model (see below for more details on abstraction). However reasoning about subsystems that are not defined in the model at certain abstraction levels is not allowed.

A functional specification is used to describe how a system being designed implements its function. In systems design, it captures the requirements of a system, which must be satisfied by the concrete implementations. The formal specification would define transformation from input to output. Furthermore, it can be checked that the requirements are consistent. Note that different levels of abstraction could feature different details of the functionality requirements. The case study in Chapter 6 offers further hints on how a specification could be constructed for infrastructure systems.

These different goals lead to a better understanding of different facets of the system. For a complete analysis, one could benefit from a combination of all of them by exploring them at different stages of analysis. Domain modelling tends to utilise higher-level abstractions, while functional modelling frequently spans lower-level details of the designed system.

### 3.6.3 Abstraction

An abstract model of a system allows us to focus on just the components relevant to the specific analysis, thus avoiding the need to describe the system in detail at the beginning. This allows description of a certain facet of a complex system as a

simpler but still representative abstract system in order to improve understanding of the original one. Note that abstraction can also be utilised to provide different "points of view" on the same system, not just a single "abstract" view.

The initial level of abstraction should reflect the taken viewpoint or the problem being modelled. As a trivial example, consider a model of electricity being supplied to a house. At the very abstract level, one may only want to describe the property that electricity is being generated and supplied to the house. So the highest level of abstraction would only involve entities of *electricity generator* and the *house*, with a supply link between them. This is enough to capture the notion of electricity being provided and does not include irrelevant details. When examining dependability properties of such a modelled system, one is quite restricted in how detailed they can be. An interface of the generator at this level would be limited to "*generator provides electricity*". Then, only one failure could be specified: "*generator stops working*"—the *OFF* state. This is because the model does not include any details about the generator or how the electricity is delivered. However, the arguments constructed are well founded, since the model includes everything being talked about.

During the course of system design or analysis, this high-level view would be refined to a more concrete system model including additional high-level concepts, components that support further analysis—down to concrete implementation details. For example, another level of abstraction for the above model could introduce the concept of *cables*. Cables deliver electricity from the generators to the house and link these components. At this level, the abstract *generators* are replaced by linked *cables* and more concrete *generators*, e.g. represented by entity *generator2*. This allows one to refine the interfaces, system properties and reasoning about causes of failure. Additional components introduce further faults: now the system can fail when either the more concrete generator fails, or when the cables fail delivering the generated electricity. This way the properties of interest are enriched with further detail, but the model still contains only the relevant details.

When the analysis includes several properties of interest, a good way to structure the development process is to start with the most abstract properties (ones that require the least detail). Then further properties are introduced during refinement (see below), thus ensuring that the previous properties still hold.

The example above is very abstract and only aims to illustrate how one could start thinking about infrastructure systems from the very top. However, depending on the purpose of the model, one could just as well start at a more detailed level.

If, for example, a property being reasoned about requires quite detailed concepts, it should be included from the start without artificially creating levels of abstraction that will not actually be used. So in the above example, if the property of interest is concerned with the cables, that second level should be the one to start with.

### 3.6.4 Assumptions

Assumptions are used for circumscribing components in the model that will not be modelled explicitly. Following the rule that reasoning can only talk about things that are in the model, such components and their properties need to be part of the model. By writing an assumption (e.g. using rely-guarantee rules), one can describe how the modelled part will interact with that external component, but avoid modelling it in this way.

To illustrate, a simple example about electricity generators being cooled with some water system is given. The electricity system is the one modelled originally. Since it can overheat, it needs to be written down that for correct operation, electricity generators are cooled correctly to prevent failure from overheating. The cooling is not modelled explicitly here and is therefore written down as an assumption "*generators are cooled correctly.*"

In the event of the model being extended with water infrastructure, it could be chosen that water will be used for cooling the generator. When the water and electricity system models are combined into one, the assumption becomes an internal interface between the generator and the water-cooling system. Depending on the abstraction level, that could be left as "*water system is enough to cool the generator.*" Alternatively, one can choose to model in more detail, e.g. "*the temperature of the generator is always less than 50°C.*" However, in this case, temperature and other related variables must be modelled in the system. It would become important to know how to measure and retrieve the temperature of the generator, how it changes, what affects it, what are acceptable intervals of measurement, etc. Further details about the water-cooling system would also be necessary: how the water system cools the generator, what are the limits of its power, how it is measured, etc. These details do not need to be modelled explicitly, though, since they could be written down as further assumptions—the cooling system is modelled, but not in its entirety, it must respect the given assumptions.

### 3.6.5 Structure

In complex systems, structure dictates the behaviour of the whole system by linking its components in a certain manner (see Section 3.4). When constructing an abstract model of a system for analysis, structure would also be defined in an abstract way. It would link abstract entities in the model instead of replicating a real-life concrete configuration. Consider the earlier example of a *house* being supplied with electricity by *generators* via *cables*. All these are abstract entities, thus they could have an arbitrary structure, e.g. a house may have several electricity supply cables, which would in turn deliver electricity from different power generators. The model thus would specify the general state of a *house* having multiple *cables* (thus covering all possible variants), which in turn are linked to multiple *generators*, which can be shared between the *cables*.

In this model, one could define properties to analyse, e.g. failure propagation. By employing various reasoning techniques, general configurations that would satisfy the property can be identified. For example, one may discover a situation where at least two cables connecting at least two shared generators to the single house are required to avoid some specific failure propagation. When constructing a specific instance of the system, with a specific set of houses, cables, generators and links between them, it would be necessary to verify that they satisfy these discovered conditions (that refinement is correct). Then the failure propagation properties discussed at the abstract level would hold.

### 3.6.6 Refinement

The idea of top-down analysis is that one starts with an abstract (high-level) model and then step by step transforms it into a concrete (low-level) model. A set of rules on how one could perform such a transformation is called a *refinement* (or *reification*) [Jon90]. Refinement rules ensure that the properties specified at the abstract level hold for the concrete one as well. This allows structuring the specification in a way that allows reasoning about it at an appropriate level.

During refinement, properties can be replaced by more concrete versions of themselves as the model is enriched with more details. The following example illustrates such a mapping using the model of the electricity system being cooled described above. To prove the refinement correct, first the mapping between the concrete and abstract states must be verified. Such mapping is called the *retrieve function*

and it defines how find an abstract state that is represented by the given a concrete state. The retrieve function must satisfy two properties to establish a correct mapping: *totality* and *adequacy* [Jon90]. A *total* retrieve function ensures that all concrete states have some abstract state they represent. The *adequacy* shows that all abstract states are accounted for, i.e. there is at least one concrete state for every abstract one. With the retrieve function defined, the system operation refinement can be checked: an operation of a system performed at the more concrete level must correspond with the respective operation at an abstract level. Refer to [Jon90] for further details on data and operation refinement.

Consider an electricity-generating system that needs to be cooled. At the abstract level this would be defined as an invariant "*system is cooled enough*". All operations at the abstract level that manipulate the system state, such as "*produce a unit of electricity*" preserve this invariant, thus applying such an operation on a valid state produces another valid state.

At the more concrete level, the property "*cooled enough*" can be specified as an invariant "*system temperature is less than 50°C*". A retrieve mapping between the abstract and concrete properties would link that "*enough*" is equivalent to "*less than 50°C*". Showing the totality and adequacy of this mapping is trivial: there are two of each abstract and concrete states, i.e. "*enough*" and "*not enough*" at the abstract state, and "*less than 50°C*" as well as "*more or equal than 50°C*" at the concrete state, where the property values map respectively in both directions).

The refinement of operations would include the "*produce*" operation becoming, say, "*generate 1 kW of electricity*". This would include details on how much heat this generation produces and how the cooling system works to compensate the increase in temperature.

To prove the refinement correct and thus to show that the abstract property of "*cooled enough*" holds at the lower level of abstraction, one needs to analyse how the operation affects the concrete state in relation to the corresponding abstract operation. The operation "*generate*" is performed on a concrete valid state where the temperature is less than 50 °C. Now there could be two cases that can happen during reasoning. Say that the operation produces a state where the temperature is 40 °C. When mapped back to abstract state, this maps to "*cooled enough*", which is a valid state produced by the abstract operation as well, thus the refinement is correct. Alternatively, say the temperature becomes 60 °C. Mapped to the abstract state, it will yield "*not cooled enough*". From this it follows that the concrete operation does

not reflect the abstract one (and the concrete system does not correspond correctly to the abstract one). Therefore the refinement is invalid and properties defined at the abstract level may not hold for the concrete one.

# Chapter 4

# Systems generating systems

Leading to this chapter, the thesis explored dependability and failure propagation in systems that are deployed or built upon each other. It covered conventional systems: the actual systems being built, e.g. infrastructure systems or computer systems from which the concepts are borrowed. This chapter proposes to expand these concepts to include the planning (design) process as another system in the whole analysis. Such systems, which can be thought of as *systems generating systems* or *systems changing systems* can also produce failures that give rise to faults in the actual infrastructure systems. Therefore they can also be treated with similar rigour, and benefit from dependability analysis and reasoning techniques in the manner of conventional systems.

In a complex system of systems, the component systems interact with and depend on each other through certain interfaces between these systems. This interaction can give rise to failure propagation: when a failure in one system manifests a fault in another one. The fault then can trigger an error state that can result in a system failure. When performing analysis of the failure, one can track this chain of failure propagation back to the source, for example, to eliminate the original fault or add adequate fault tolerance measures.

In conventional systems, the origin of this chain may be some latent fault already existing in the system. This chapter proposes to look further beyond the boundary of this system where such faults could actually be created by failures in system design, development, maintenance, etc. An obvious example from computer science is a bug in a computer program. The bug is a fault which can be triggered, for example, by certain inputs, to put the program into an error state (that produces a failure, that can in turn produce erroneous input to a dependent program, thus propagating the failure). The bug is actually caused by a failure of the programmer (or designer,

architect) developing the system. Looking even further, one can explore the faults in the development (design) process that lead to the programmer making the bug (failure). This suggests that analysis of the system development process may lead to understanding and reducing faults in the resulting system.

When taking the top-down approach, an abstract view of the system is first developed and then refined to a concrete realisation. Thus for such an abstract approach, the "creation" of the system is its planning and design process. This chapter explores the H3 hypothesis, which proposes inclusion of the planning process of a system that comprises the eventual infrastructure in the overall dependability analysis. The idea is generalised to "systems-generating-systems", i.e. any systems that give rise to future (or change existing) infrastructure systems.

The idea is to extend the system concept to capture the planning, development and maintenance processes as just another system, thus leading to the position that "everything is a system." The source of failures can then be traced into these "generating" systems. The analysis of infrastructure systems would then include systems outside the conventional infrastructure. Take, for example, a train crash whose immediate cause appears to be the driver crossing a red signal (failure in transport infrastructure). Depending on the scope of the "systems" under consideration, this failure could be treated as being caused by poor signal position (failure in planning system) or by inadequate driver training (failure in human systems).

By treating the "generating" systems as just another system, one can try extending considerations about failure propagation, fault tolerance and interdependency analysis of planning or development stages to be similar to ones in physical infrastructure systems. The belief is that a planning process included as a system in infrastructure system analysis might help raise different questions about system dependability. Hopefully, that would give a better perspective to understand and analyse interdependent national infrastructure systems.

## 4.1 Planning as a system

To describe the planning or design process as a system, one needs to identify its functionality, boundaries and how it interacts with other linked systems. Note that the relationship between the generating system and the generated one (actual system that will comprise infrastructure) is logical. This interdependency cannot be described as information or resource flow between generating and generated systems,

because the planning system changes the generated system itself. So the actual generated system is an output (and input) of the planning system.

This leads to the interface of the planning system. Section 3.3 argued that the whole system can be described at its boundary, which consists of function description, assumptions, requirements and other information, providing different views of the system. From its function perspective, a planning system interface consists of requirements for the plan (design), environment description as well as the existing infrastructure system data as the input and then the completed plan, design or the new infrastructure itself as the output.

Depending on the level of abstraction, such planning systems can actually be decomposed into constituent systems. Note that an abstract infrastructure system would correspond to the abstract planning system. For example, at a very abstract level, an infrastructure system could be represented as an abstract model or specification. This model or specification is then the output of the corresponding planning system. The planning system could be described at any level of abstraction, but introducing unnecessary details for planning abstract infrastructure would not bring much benefit. For example, if the infrastructure system model is only an abstraction at this point, detailed descriptions of construction or deployment systems as part of the planning system could be considered somewhat excessive. The high abstraction level of the infrastructure system suggests that in the planning system analysis there is no need to introduce low level details as well.

If top-down approach is employed for infrastructure analysis, further steps introduce additional details into the infrastructure, down to realisation details. To follow this, the concept of its planning system should be refined to match the level of detail. This may require splitting it into sub-systems, e.g. design, construction, deployment, but yield the constructed infrastructure as the output of the whole planning system-of-systems.

In failure propagation analysis of a scenario involving a complex system-of-systems, the fault-error-failure chain could span both the physical and the planning systems. Within the physical system-of-systems, development faults are part of the component systems and the chain happens when a failure in one system *activates* a fault in another one. For "systems generating systems," a failure in the generating system *creates* a fault in the generated one. The act of fault tolerance in planning would result in elimination of the said fault, not prevention of its activation.

This difference of relationship between the planning and the generated system

introduces the separation between these systems. Therefore the dependability analysis would include considerations about both the generating and generated systems, however they would be analysed separately.

The planning system with its connected systems can be analysed as a system-of-systems itself. The most common example is a planning system comprised of different organisations responsible for different parts of infrastructure. This thesis argues for a holistic view of infrastructure system-of-systems: the analysis must encompass all infrastructure systems. The same approach must then be taken when analysing their planning systems: analysis must take into account all participating organisations, institutions and governmental bodies that plan, design and develop the infrastructure systems.

Note that parallels can be easily identified between the structures of the generating and generated systems. Organisational structures often reflect the structure of their designed system. The direction of "which mirrors which" is actually an interesting sociological observation. At the high level, it is the system that influences organisational structure, e.g. each infrastructure sector is often governed by a corresponding organisation or institution. At lower levels of system structure, however, one could assume that Conway's law would hold: "the structure of the system mirrors the structure of the organisation that designed it" [HG99].

In most cases, different organisations are responsible for different interdependent systems. Within an organisation, separate interdependent components are frequently developed by different management teams. This can lead to a level of separation, incompatibility and mismatched interfaces between the designed components where organisational links are faulty. A famous example of incorrect assumptions between organisations led to the Mars climate orbiter being lost in space. The separate software components were developed by two different companies: NASA and Lockheed. Miscommunication on measurement units used (imperial vs metric) led to incompatible components and failure in the probe [NAS13].

## 4.2 Addressing failures of planning systems

The planning and development process of an infrastructure system can consist of multiple subsystems and therefore have failure propagation as well. The mismatched interfaces between planning systems as simple as in the NASA example above can create major faults in the resulting generated system. Faults are created by failures

that propagate outside of the planning system and manifest in generated results (model, specification or the final created infrastructure system). To avoid failures affecting planning system services, they should be identified and addressed using appropriate fault tolerance measures.

In computer science, various process models have been developed to facilitate good interaction and collaboration between organisational units to lead to a correct computer system. They offer frameworks to ensure the quality of development output that incorporate testing, review and other similar phases. These can be considered fault tolerance measures to catch and address failures of system designers or developers. Such process models from computer science and other domains could help answer questions and solve issues identified during analysis of the planning process as a system.

For example, one may try to avoid problems with mismatched interfaces by having a central body to regulate the construction of each subsystem. If examined in an abstract configuration, however, this could yield a single point of failure in the central body. Thus if a failure happens here, it may affect all planning subsystems and therefore produce a system-wide failure. An alternative could be to avoid the single point of failure by having more independent design teams (planning systems) or review of the central body activities. Expert review is one possible fault tolerance measure for planning systems. Following the proposed approach, it should also be treated as a system and described at its boundary: the expert review process provides the review service and would catch existing failures but, say, only with assumptions that the experts have adequate expertise and skills, the review is performed in a formal manner, etc.

The planning system can be considered as one of the judging systems for failures in infrastructure systems. The information about infrastructure systems can be collected via various means (sensors): users or operators, automated (computer) systems analysing failures, etc. They would identify failures and their context and report to the design or development systems for improvement. So the operators can have a dual role of being part of an infrastructure system itself as well as being part of the planning system (identifying, judging and reporting failures, or even changing the system via workarounds!).

This chapter argues that planning systems can be subjected to the same framework of description and reasoning. They exhibit properties and concepts parallel to ones in conventional systems: boundaries, human failures, failure propagation,

fault tolerance measures, etc. The ideas can be extended to as many meta-levels as required: the planning system could have another one that generates it (planning of planning). Consider a government devising a plan to create an infrastructure system and setting up the process of how the infrastructure will be developed. Note that failures at this level of planning would likely have a major impact to the final infrastructure system through failure propagation (faults would be created in the infrastructure planning and construction process, which would affect the created infrastructure system when activated). For this reason, adequate expertise is vital for people working at the "planning of planning" systems.

The analysis of a planning system as another system could help identify origins of faults in infrastructure systems and design fault tolerance measures for system design and development processes. This chapter provides several small examples related to infrastructure system design. Some more insights in to how the framework could be used to describe the planning system are available in the case study (see Section 6.3).

# Chapter 5

# Problem background and related work

The economy and modern society strongly rely on provision and availability of infrastructure services. The interconnected nature of infrastructures raises challenges when failures occur, since a failure in one infrastructure can propagate to other infrastructure systems and disrupt their services as well. Thus it is important to understand and analyse these interdependencies and the dependability of infrastructure as a whole.

This thesis has proposed ideas on adapting existing dependability research in computing science. The main focus has been on top-down development and formal description and analysis of infrastructures, their dependability and relationships. Note that the background and references for these ideas have been mentioned in the previous chapters, where they have been presented. This chapter aims to give a brief overview of related work and alternative techniques to description and analysis of national infrastructure dependability.

The issues around infrastructure dependability have been examined by numerous researchers and research projects. Historically, the focus had been to tackle the complexity of a specific infrastructure sector or system and its failure and dependability analysis. However, recent research addresses the interconnectedness of infrastructure systems and takes a holistic approach to infrastructure analysis, treating national infrastructures as "systems-of-systems". The main focus is on identifying and analysing interdependencies, especially ones that can cause failure propagation.

This chapter provides some notions to describe interdependencies of national infrastructure and then introduces some common approaches, methods and tech-

niques that researchers employ to tackle interdependent infrastructure systems and failure propagation analysis.

## 5.1 Describing interdependencies

The holistic approach to infrastructure analysis has been advocated for modern infrastructures [RPK01, Lit03, Rin04, PF07], because it accounts for the fact that current infrastructure systems are too interconnected to be analysed in isolation. The classic approach of analysing each infrastructure by itself must be supplemented by analysis of interdependencies between these infrastructure systems.

Some interdependencies are straightforward (e.g. energy powering other infrastructures), but some are less understood. Interdependencies are not necessarily physical connections—they can be identified and characterised according to different criteria. A general definition of (inter)dependency talks about the state of one infrastructure *somehow* influencing or correlating to the state of another one. Interdependency is "linkage or connection between two infrastructures, through which the state of each infrastructure influences or is correlated to the state of the other" [RPK01]. Furthemore, Rinaldi et al. [RPK01] propose a taxonomy for how infrastructure interdependencies could be described according to different criteria and categorised them into four main classes:

- **Physical** interdependencies describe physical links with material flow between inputs and outputs of systems in different infrastructures.
- **Cyber** interdependencies are concerned with information flow: the state of an infrastructure depends on information (e.g. control events) transmitted through the ICT infrastructure.
- **Geographical** (also *spatial* [Zim04]) interdependencies describe the importance of physical proximity: a nearby system can be affected by a failure in another infrastructure.
- **Logical** interdependencies encompass all other types of connections. These include relationships caused by political, social and other human decisions or actions.

Moreover, Rinaldi et al. [RPK01] suggest that full description of interdependency requires a multi-dimensional analysis. Interdependency categorisation should

consider the states and characteristics of infrastructure systems, the environment as well as properties of the link itself (e.g. coupling, etc.) or the type of failure being carried by the interdependency.

An alternative taxonomy for interdependencies has been used in [LMW04]. Lee et al. classify interdependencies by how they link related systems structurally, not by what the link represents (note that these categories can be used for both physical and other interdependencies):

- **Input**: provision of one infrastructure service requires input from one or more services of other infrastructure.
- **Mutual dependence**: bi-directional dependencies between all infrastructures in a collection of infrastructures.
- **Co-located**: any physical components of infrastructure systems are located in the same geographical area (arbitrary region).
- **Shared (`AND`)**: two or more services share some physical components or activities of an (another) infrastructure system.
- **Exclusive-or (`XOR`)**: a specialised shared interdependency, which only allows service for one infrastructure system at a time.

The identification and description of interdependencies can also be done using different approaches. Eusgeld et al. [EHK08] state that the interdependency identification process can be either qualitative or quantitative. The qualitative one requires expert insight: expert interviews, round-table discussions or workshops to identify and characterise interdependencies. However, the approach is also vulnerable to failure in discovering hidden interdependencies. The quantitative approach is based on computer simulation techniques and requires data about the infrastructures to construct and run the simulations.

The framework and ideas proposed in this thesis lean towards the qualitative approach. Top-down development would require expert decisions about the model and relationships between systems but without the need for low-level empirical data at the beginning. Furthermore, the suggested formal specification and mathematical reasoning would improve assurance in the qualitative method.

The taxonomies above can be used to describe various different relationships between infrastructure systems, but can result in too many possible relationships to

consider in analysis. This hinders dependability analysis due to the sheer number of possible connections in infrastructure systems: "everything is connected to everything" [Mas06]. Depending on system properties, only some of them would actually affect the said properties across the infrastructure system.

Regarding the dependability of infrastructure systems, Masera [Mas06] proposes a dependability-orientated approach to describe interdependencies. He narrows the view taken by Rinaldi et al. [RPK01] by relating interdependencies to the dependability of systems. To analyse dependability, one must analyse failures within these connections, thus only interdependencies carrying failure should be taken into account. This allows us to define interdependencies as connections between failure mechanisms of two infrastructure systems, through which *dependability* of each system influences or is correlated to the dependability of the other [Mas06].

Masera [Mas06] incorporates the chain of fault, error and failure (see also Section 2.2) into his definition of interdependencies: they are connections that can cause a failure propagation, i.e. where a failure in one system may be related to negative effects in another. Thus in general, interdependencies are a threat to the dependability of infrastructures.

This view of infrastructure dependability and interdependencies is similar to the one used in this thesis. As presented in Chapter 3, the systems are linked together via their interfaces. Interfaces feature description and assumptions of intended service and the linkage must ensure these assumptions are satisfied, otherwise failure may occur. Therefore, each link can carry failure from one system to another (see also Section 2.3).

Note that the main questions that this thesis, and numerous other research projects, try to address exist because the interdependencies are an inevitable part of modern infrastructures. One cannot improve dependability by removing interdependencies altogether, because the benefits of having high connectivity are significant. This is best illustrated by extensive current and planned linkage of physical infrastructure with information technology systems. ICT already affects the design, construction, control and other aspects of infrastructure systems with many potential future applications such as monitoring or management systems, distributed and remote control devices, complex live data and information systems. All this promises improved reliability and efficiency at reduced cost [Lit02].

Identifying and characterising interdependencies requires expert insight, and the taxonomies change with different approaches. For example, some approaches distin-

guish only between *geographical* and *functional* interdependencies; others categorise interdependencies differently, e.g. control (SCADA—Supervisory Control and Data Acquisition—systems) relationships can be treated either as *cyber* or *physical*. Note that due to high complexity of infrastructure systems, modelling and simulation normally avoid considering all types of interdependencies and instead focus on narrower views [EHK08].

Physical and geographical interdependencies are best understood and usually included in the analysis. The examples and case study in this thesis also focus on physical relationships the most in order to communicate the ideas using a familiar domain. However, the framework is general enough to allow describing different kinds of interdependencies. For example, assumptions can be used to specify requirements on geographical interdependencies as well as various other relationships (see Section 3.2). The assumptions would contribute to the overall dependability analysis and would need to be verified during deployment for the reasoning to hold. The ITRC project also initially focuses on these types, however cyber interdependencies are becoming more important due to the increase of computer-based control and communications in infrastructures.

The different taxonomies mentioned in this section provide a language to talk about different interdependencies. Furthermore, when designing a system, they provide guidelines of things to consider when describing infrastructure system boundaries.

## 5.2 Analysis of interdependent infrastructures

The current research techniques on analysing interdependent critical infrastructures are most concerned with analysing existing infrastructure systems for risks of failure. A number of modelling and simulation techniques are available in the literature (see surveys in [PDHP06, RD06, EHK08, DPM06, KZ11]). This section provides a brief overview of the most popular ones as well as some associated concepts.

The methodologies mostly work within the framework of risk (or vulnerability) analysis of critical infrastructure systems. They aim to provide techniques for modelling, assessment and management of risk [HSC+08]. The definition of *risk* talks about failure probability and the extent of its consequences. Risk assessment aims to address the main questions of *what can go wrong* in the complex system, *what is the likelihood* of that happening and *what are the consequences* of failure to the system

and its users [HSC+08]. Note that modelling and simulation techniques require such concepts to be quantifiable: e.g. define probabilities of failure happening, establish the extent and consequences of failure as monetary or otherwise calculable value, etc. Furthermore, risk deals with concepts that are hard to quantify: e.g. the failure can have impact on related social, economic and technological systems, it almost always implies future events and numerous hazard and outcome scenarios [Hel07].

*Vulnerability analysis* is a narrower approach that focuses on the infrastructure system itself. *Vulnerability* is a property of the system and is often considered to be a fault that can trigger a failure when exposed to a hazard. Vulnerabilities are quantified by the frequency of failure when exposed to a hazard as well as by the extent of damages caused by the failure [KZ11]. Vulnerability assessment of interdependent infrastructures aims to identify and quantify vulnerabilities in a system. In addition, it can also be used to identify most frequent failures, devise prevention or repair strategies [EHK08]. The concept of vulnerability is easier to deal with than that of risk, because it is considered a property of the current system and is more concerned with how the system can be affected instead of investigating possibilities of future hazards happening (cf. risk) [Hel07].

Risk management follows vulnerability (or risk) assessment and aims to provide measures to reduce vulnerabilities, improve resilience and address failure consequences. This analysis also needs to consider trade-offs of taken measures and how current decisions may impact future options for the infrastructure systems [HSC+08].

The risk and vulnerability framework provides questions to be raised about infrastructure system dependability and guides how to evaluate the results. The methods described further in this section operate within this framework and provide tools and techniques to identify vulnerabilities and evaluate their effects. The framework is comparable to ideas proposed in this thesis. Parallels between vulnerabilities and a fault-error-failure chain can be easily established. The full chain, especially if defined formally, gives a more precise description of failure propagation and thus clearer identification of vulnerabilities (see Sections 2.2 and 2.3). Vulnerability and risk assessment can be viewed as a bottom-up approach—it concerns evaluating existing infrastructure and establishing properties about it. This thesis advocates the use of a top-down approach, which raises these questions early in the system design and development process, thus prompting developers to address the vulnerabilities (see Chapter 3).

The infrastructures and their interdependencies are often modelled as complex

networks. Infrastructure components are regarded as nodes in the network, and their relationships captured as edges. Therefore interdependencies are edges which join nodes from two different infrastructure networks [HDO11]. Complex network theory approaches can be used to identify vulnerabilities. For example, metrics are proposed that describe and quantify the system's quality of service. The network can be modelled as purely topological or with weights on edges to represent the strength of the connection [KZ11]. A common approach then is the "what if" analysis, where nodes or edges are selected (e.g. randomly) and their state is modified. The altered system is re-evaluated under various scenarios using simulation and the outcomes are examined to identify vulnerabilities [DPM06]. To address vulnerabilities, these models can include fault tolerance measures, e.g. buffering [SW07]. Furthermore, various different metrics are available to describe system vulnerabilities in terms of the interdependencies: e.g. connectivity loss (calculates ratio of available connection paths before and after failure) [HDO11], temporal scale of interdependency (considers durations of infrastructure service outages) [MIA10, CBB+11], etc.

Another interdependency analysis model is used to examine ripple effects of disruptions to outputs of component systems. The Leontief Input-Output model is a holistic framework to estimate economic impacts and sector interdependencies [HSC+08]. It is used to capture interconnectedness among different economical sectors via the commodity or information flows and forecast how the operability of interconnected systems is affected when one system output decreases by a certain amount. Such an approach is suitable to model interdependencies of infrastructure systems and has been adapted to critical infrastructure systems [HJ01, JH04].

The agent-based modelling framework allows looking into interdependencies and vulnerabilities of infrastructures from the level of system components [RPK01]. Each component is modelled as an agent: an autonomous system with specified behaviour and interactions with other agents. Such an approach allows simulating interactions between systems as well as with the physical world [KZ11]. Moreover, agents can be modelled with the ability to learn about the environment and formulate unique decision rules [EHK08]. Agent-based modelling permits simulation of an infrastructure system from simple behaviour of low level components that exhibit emergent behaviour in cooperation. The model describes each agent with a set of rules, which comprise the following basic characteristics [EHK08]:

- **Location** describes agent's physical space, e.g. coordinates or region.

- **Capabilities** describe agent's interaction with the environment, e.g. how it reacts to changes, shares knowledge and adapts to the changes.

- **History** is the agent's memory of previous experiences, e.g. stress or ageing.

Agent based modelling uses bottom-up design strategy to describe the behaviour of the agents [EHK08]. The emergent behaviour that arises from rules designed for each agent and their interaction removes the need for system-level description. The high-level models of the overall system (e.g. description of the behaviour of the whole infrastructure) are no longer needed, because the high-level behaviour follows from low-level interactions. This model provides a natural approach to distributed systems [RD06, EHK08].

One of the main disadvantages of simulation models for critical infrastructures is the lack of low-level data availability. Information about infrastructure component systems is usually proprietary and considered very sensitive by the infrastructure stakeholders [KZ11]. Empirical data (e.g. probabilities or other statistical data) about events causing "interesting" failures or failure propagations is often lacking, because such events can be quite rare. Researchers either try to scavenge for such data from public documentation (e.g. [RIT+08]) or use intuition and common sense to create "appropriate" data.

Furthermore, tackling complexity of infrastructure systems via simulation is a challenge in itself. The common approaches are still usually limited in interdependency types, number of infrastructures and their metrics that are included in the scope of analysis (e.g. [HDO11]). Also, the system complexity makes it infeasible to achieve good levels of assurance and correctness through the use of simulation [Cia04].

# Chapter 6

# Case study

This chapter explores a way of applying dependable computing theory, particularly top-down development, for describing and creating dependable system-of-systems involving national infrastructure. This case study takes the task of describing and modelling an abstract hospital example to illustrate the ideas presented in the thesis. The focus is on relationships between systems, e.g. dependencies between hospital and infrastructure systems. The selected hospital system could be generalised to any other system, even infrastructure systems themselves.

The case study does not aim to produce a full model of an infrastructure system-of-systems.[1] The steps developing the hospital example give hints and directions on the suggested approach. They aim to illustrate some of the concepts and benefits of the ideas presented in this thesis:

- High-level description of infrastructure systems.
- Top-down development of infrastructure-based systems: refining abstract descriptions to introduce relevant details.
- Recording system properties and assumptions—then ensuring they hold during the development.
- Matching interfaces between dependent systems.
- Designing fault tolerance measures for non-reliable systems to increase system dependability: e.g. redundancy or buffering.
- Planning processes and human interactions described and analysed as systems.

[1] The task of modelling a full infrastructure system is outside the scope of an MPhil thesis and should actually be attempted with resources of a whole organisation.

- Describing and modelling failure.

All of the techniques presented in this case study can be generalised and employed in describing other infrastructure systems and similar concepts.

## 6.1 Approach

The case study illustrates the development of and reasoning about a system which depends on infrastructure services. The system and its dependencies constitute a system-of-systems. It is described in a top-down manner, starting with an abstract view and introducing further details in the following levels. The case study aims to illustrate how the dependencies, relationships and failure propagation can be described in an abstract way, leading to a clear definition.

The aim of the case study is to take a system with related infrastructure systems. Note that the example system can be substituted with different systems in other scenarios: the approach will be transferable to other developments. For this reason, the case study tries to be more abstract and general than perhaps required for a specific system development. The example system could also be another infrastructure system, i.e. the case study is just a point of view to a complex system-of-systems with the example system (the hospital) as a reference point here.

These models presented in the case study are informal and thus the reasoning is also informal. It serves the purpose of illustrating the process and main ideas, but does not aim to be exhaustive or formal. For this reason, a formal modelling language is not used (e.g. VDM or Z) and instead an ad-hoc format is employed to present the reasoning.

The approach taken in constructing a model is to focus on definitions of various *states* of systems. A state description records important features of the system, which can comprise *properties* of the system itself as well as *assumptions* on some external dependencies. The majority of the case study is concerned with the *correct* states, i.e. describing systems and their behaviours (as values of the specified properties) that provide the correct service. During the modelling, the goal is to find the circumstances leading to an always-correct state of the reference system. Such an approach ensures detailed specification of system functions that cannot cause failures. The system faults are found and handled during abstract modelling and development rather than during deployment. Nevertheless, modelling error states

and failures leading to them can be important to system description. Section 6.2.7 explores how they can be incorporated within this framework.

Note that the term "correct service" is used throughout the case study with assumption that such modelling would be accompanied with a (formal) specification to justify the "correctness". Otherwise, the wording "correct" should be taken to mean "intended service" (see also Section 2.2).

Following a top-down approach, the system model is presented at several layers of abstraction. Each level is introduced by refining a more abstract description. Note that the order of refinement as presented in the case study is somewhat arbitrary. Certain steps of refinement are independent and can be done in parallel or skipped altogether. The case study is organised in a way that more concrete layers of abstraction follow the abstract ones, though.

The presentation below uses a box below for the description of a system *state*. It describes the different properties that would hold for a correctly working system (failure modelling would involve the cases when the properties do not hold). Furthermore, it aims to give descriptions of assumptions about relationships (dependencies) with other systems. These properties and assumptions constitute a system interface (boundary).

**System**

| | |
|---|---|
| Systems: | *List of subsystems within this system-of-systems.* |
| Property: | *System property, e.g. a description of some system feature. (The value in parenthesis describes system invariant: whether the property must always hold for the described state:* `true` *or* `false`*.)* |
| Property: | *There can be multiple properties—each defined on its own.* |
| Assumption: | *Assumptions on external conditions required for correct state.* |
| Needs: | *Dependencies on other systems.* |

The fields *systems* and *needs* are used to list related systems within the model. The *systems* field is used for abstract decomposition, i.e. to write down the main components/participants of the system described by the box. Thus the *systems*

field talks either about sub-system decomposition or about a collection of related systems that comprise a system-of-systems.

The list of *properties* can describe what the system in question does, how it behaves and what are the conditions and parameters for the system operation. In this case study, all *properties* are boolean-valued: the property may hold for a specific state of the system or not (values `true` and `false`, respectively). Furthermore, the majority of the case study describes the *correct* state of the described system. The properties are phrased in the way so that they are `true` for the correct state. This is expressed as a system invariant, i.e. by writing down the value that will always hold for the state. Some *failure states* (where certain properties become `false`) are sketched and explored in Section 6.2.7. The *properties* can talk about individual component/participant systems in the context of the whole system, span multiple components or describe to the main system itself.

The *assumptions* are used to write down dependency considerations. They are concerned with the properties of systems that the described one depends on. These systems (dependencies) are listed in the *needs* field. The assumptions thus describe the system *environment*, normally listing the conditions required for the correct operation of the system. The assumptions contribute to the system interface and must be checked when the interfaces are matched within a larger system-of-systems.

The following case study uses these boxes to describe different levels of abstraction of the modelled system. A single system can be described in incremental detail using several boxes: each box would represent a different level of abstraction. Each level is a refinement of the previous, more abstract, level of the system. The concrete description can introduce additional concepts, properties or related systems; or revise the existing ones with increased detail and accuracy.

## 6.2 Infrastructure analysis: hospital case study

The case study takes a *hospital* as the reference system. It has been selected because of the various dependencies on infrastructure systems: energy, ICT, water, waste, human systems, etc. Note that any other system (even an infrastructure system) could be substituted in the hospital's place. The assumptions and dependencies would differ, but the general approach and ideas would be very similar.

### 6.2.1 Abstract hospital and electricity infrastructure

The case study focuses on relationships between systems in order to describe their interfaces and possible failure propagations between them. To avoid unnecessary details at the very abstract level, it is limited to two systems: the hospital system and electricity infrastructure system, which it depends on.

The hospital and electricity systems comprise the basic system-of-systems of this case study—so called "the world". The world represents the *top* level system-of-systems and records all top systems considered in the model. Figure 6.1 sketches the systems introduced at this level. When new top-level systems are introduced later during the refinement, they would need to be recorded in a refined version of the world as well.

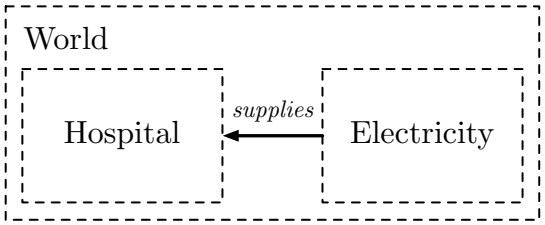


Figure 6.1: *World* system-of-systems.

**World**

Systems: *Hospital, Electricity*

Property: Hospital *is supplied with* Electricity.

The abstract descriptions of hospital and electricity systems follow: they aim to record the requirements and services of each system but do it at the appropriate level of abstraction. In fact, at the very abstract level, only certain features of interest are recorded, thus allowing to "slice" the model into manageable parts.

When describing system boundaries, they should only be described in terms of concepts already introduced in the model. For example, the property "*hospital is operational*" cannot reference details about what "*operational*" means. This property records the intended requirement for the system but hides a lot of details behind the statement. This is a way of insulating against internal details not needed at this level (e.g. equipment, building, etc.). See Section 6.2.5 for details on hospital

system refinement. In a full analysis, this would include various external assumptions, e.g. "*absence of floods*", but these would come in later—this is an example of abstraction.

**Hospital**

| | |
|---|---|
| Property: | *Hospital treats patients.* (`true`) |
| Property: | *Hospital is operational.* (`true`) (H-Oper) |
| Assumption: | *Hospital is supplied by electricity sufficiently and reliably.* |
| Needs: | *Electricity* |

**Electricity**

| | |
|---|---|
| Property: | *Provides adequate electricity.* (`true`) |
| Property: | *Electricity system operates reliably.* (`true`) |

All properties presented here are two-valued in general, i.e. "*hospital is operational*" can be either `true` or `false`. As mentioned before, the majority of the case study is concerned with the *correct* state of the described systems. A correct state of the hospital requires it to be operational, thus the invariant includes the requirement that property H-Oper holds, i.e. is `true`. The error state, where hospital is not operational (invariant for H-Oper is `false`), would be used in failure description. Section 6.2.7 elaborates more on modelling failures.

When collecting the systems within the "world" (or within any other composite system), one needs to justify that the world configuration is correct. The most important parts are reasoning about matching interfaces between the systems and identifying whether all assumptions and requirements are addressed and satisfied. Assumptions that have not been addressed become assumptions of the whole system—the "closed world". They must be noted and verified to match the deployment environment (see Section 6.3).

In the case of *World*, the interfaces between *Hospital* and *Electricity* do match at this level of abstraction: *Electricity* provides "*reliable*" service to *Hospital*.

Note that "*reliably*" is not defined here due to the high level of abstraction: it

hides a number of properties and requirements of actual systems. The example thus records that there exists a reliability-based link between the hospital and electricity systems. The actual properties contributing to a link's "*reliability*" would then be identified at further levels of detail.

### 6.2.2 Unreliable electricity

This level adds further details to the electricity system defined abstractly in *Electricity*. The abstract system had an assumption that electricity is provided "*reliably*". The case study assumes that reliability of a single real world electricity system cannot reach the level required by themselves. Therefore, when more concrete levels of description that lead towards implementation (deployment with real-world systems) are introduced, it is important to model the real-world features and thus the inherent unreliability of said power systems.

The next level of description must refine the abstract one. Figure 6.2 sketches the refinement of the abstract *Electricity* system with a more detailed *ElectricityRW* that involves real-world (RW in the name) unreliable power sources. The

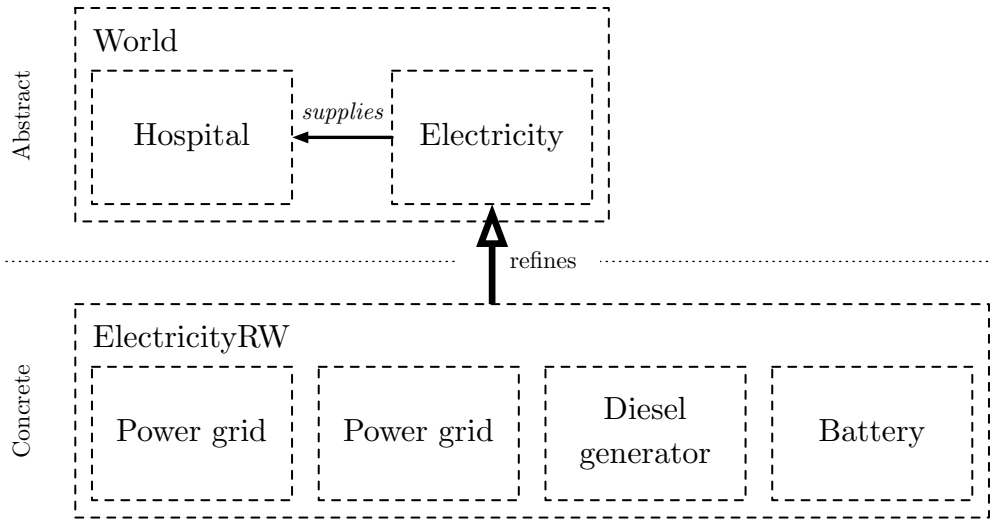


Figure 6.2: Refinement of *Electricity* system with real-world components.

full description and justification of this level of abstraction follow. Note that this refinement is only concerned with the electricity system and the *Hospital* system is not affected. By satisfying the abstract properties on *Electricity*, the assumptions on *Hospital* ensure the system interfaces are unchanged and match.

When the non-reliable electricity source is included in the detailed model, the model needs to satisfy corresponding assumptions at the higher level of abstraction:

that "Electricity *supply is reliable.*" Otherwise the refinement is not correct and the abstract property about matching interfaces between hospital and electricity no longer holds. This can lead to a failure propagating from unreliable electricity to the hospital via the dependency.

The top-down approach forces the designer to reason about and justify more concrete levels of the model. If the refinement is correct, the detailed model must explore all cases and address possible faults of the detailed system interface. By showing that refined electricity is indeed "*reliable*", the designer ensures that the original property of matching interfaces holds.

With real-world power systems being not reliable enough to satisfy the assumption, further measures are needed to achieve the "*reliably*" status. Note that this example just demonstrates a general idea, so the measures chosen are for illustration purposes. Suitable means to achieve the necessary reliability and thus satisfy the assumption is *redundancy*. The example selects two independent power grid sources, thus improving redundancy, plus another redundant system (diesel generator) of a different kind. To supplement that, one can also design *buffering* redundancy, e.g. battery power source for electricity or some kind of water storage for water infrastructures.

The *ElectricityRW* box outlines a way to describe a configuration of energy dependency to achieve the necessary reliability. Some details and independent definitions of each system are omitted for the sake of conciseness.

**ElectricityRW ("real world")**

| | |
|---|---|
| Systems: | *Two Power grid supplies, Local diesel generator, Battery* |
| Property: | *Power grid supplies are independent.* (`true`) |
| Property: | *If one power grid is not operational, generator is on.* (`true`) |
| Property: | *If grid and generator are not working, battery is used for energy.* (`true`) |
| Property: | *Battery supplies max* $\mathrm{E}_{\mathrm{battery}}$ *amount of electricity.* |
| Property: | *Generator produces max* $\mathrm{E}_{\mathrm{diesel}}$ *amount of electricity.* |
| Property: | *[Properties on grid, generator and battery to achieve necessary reliability in this configuration]* |

The task of showing that the *ElectricityRW* configuration satisfies the "*supplies reliably*" property in *Electricity* requires further reasoning. The overall argument relies on the properties of each component system, in particular on their reliability. For example, since *ElectricityRW* does not talk about any measures of reliability yet, just about the configuration, the descriptions of component systems could be abstract as well. For example, the following description of *Power grid* indicates an adequate but non-reliable system without specifying further details.

**Power grid**

Property: *Electricity supply is not reliable.*
Property: *Electricity supply is adequate.*

Subsequent refinements of *ElectricityRW* could introduce some measure of reliability and would need matching refinements of each power system. Then mathematical formulas can be derived to describe the intended requirements, which would be validated against each system description. All the derived (refined) requirements would be eventually validated during deployment to ensure that the actual selected systems implement the model—the overall correctness would hold then.

The question of failure propagation here is addressed by top-down development. By showing a correct refinement, one can ensure that no failure propagation will happen as long as the derived properties on *ElectricityRW* hold. The question of failure modes is addressed during development, not during deployment or actual use.

These two levels of abstraction demonstrate a single step of refinement in the process of describing and developing an infrastructure model using a top-down approach. The first state described the intentions of the model and recorded high-level assumptions. The next state introduced further real-world details and required reasoning to show that the chosen configuration satisfies the requirements defined in the abstract state. The steps should continue in a similar manner until a satisfiable level of detail is achieved.

For completeness, similar models and reasoning should be defined for other infrastructures as well. A hospital depends on energy, water, ICT and other infrastructures. The way of describing and modelling them is similar to the electricity case shown above and thus will not be written down explicitly here. Note, however,

that all the additional details refine the original abstract property "*hospital is operational*". Furthermore, consider that additional justification will be needed when addressing multiple interdependencies between infrastructure systems, e.g. electricity depends on ICT infrastructure for control services. However, as was mentioned at the beginning of the case study, this example can easily be adapted with different reference points, e.g. *Hospital* could be substituted by *Electricity* and the analysis would be about its relationships with other infrastructures.

### 6.2.3 Human systems

The generality of a top-down approach allows expanding its application to the analysis and development of other types of systems as well. Following the idea that "everything is a system", similar modelling can be used to describe human systems. Humans contribute to a large number of failures in systems and thus should be considered during development and analysis. Note that the effect of humans is twofold: they can make mistakes because of tiredness, and they do not always follow rules, etc; however they may react outside the designed rules for the benefit of the overall system, e.g. take improvised but appropriate and correct decisions (see Section 2.4).

Consider the hospital example in this case study. The hospital staff treats the patients and operates the hospital equipment. Failures in executing their tasks as required would affect the outcome of the hospital's *service*: "*Hospital treats patients.*" This step of refinement tries to record the dependency of the *Staff* as a human system. Figure 6.3 provides a sketch of this refinement step. Note that this refinement step can be considered as parallel to the previous one (electricity refinement). The order of taking these steps is not important, as the assumptions on *Hospital* insulate against refinements in each system. Therefore in the abstract step (Figure 6.3), either *Electricity* or *ElectricityRW* can be used.

At the abstract level, *Hospital* box defined dependency on electricity and can similarly be extended for dependencies on other infrastructure systems. The refinement step in this section introduces *Staff* as another system with assumptions in the hospital. To avoid too much detail on infrastructures here, the boxes below provide the human aspect of the hospital model.

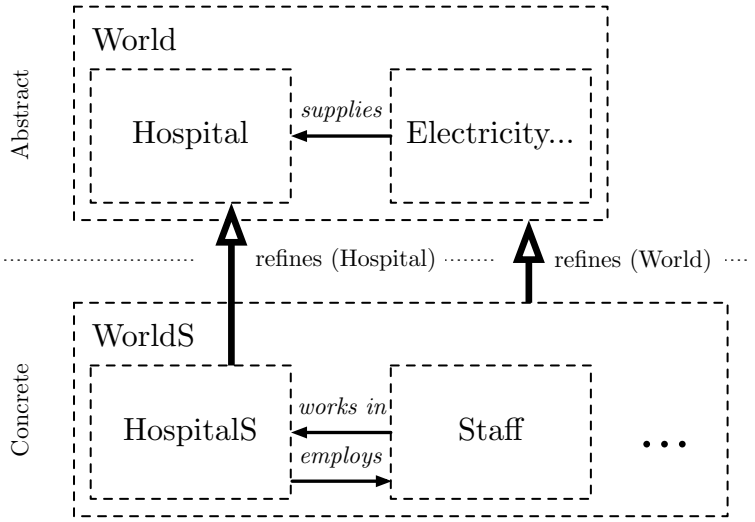


Figure 6.3: Refinement of *Hospital* system to include *Staff*.

**HospitalS ("with staff")**

| | |
|---|---|
| Property: | *Hospital treats patients.* |
| Property: | *[Properties about hospital being operational are unchanged.]* |
| Assumption: | *[Assumptions on electricity and other infrastructure systems.]* |
| Assumption: | Hospital *is provided with staff performing their duties correctly.* |
| Needs: | *Staff* |

**Staff**

| | |
|---|---|
| Property: | *Staff performs their duties correctly.* (`true`) |
| Property: | *Staff is trained adequately.* (`true`) |
| Assumption: | *Staff working environment matches their training appropriately.* |
| Needs: | *Workplace, e.g. Hospital.* |

The hospital in the model is described as a black box, i.e. the case study records the assumptions needed to fulfil hospital service correctly, but does not give details on how the actual patient treatment is done. The *Staff* dependency is therefore

recorded in a very similar manner to infrastructure systems: "*staff performs correctly.*"

This level of abstraction assumes perfect employees: if they are trained appropriately and work in an environment matching this training, they do not fail: "*staff performs correctly.*" Therefore, in the "world" of this level (*WorldS* in Figure 6.3), interfaces between *HospitalS* and *Staff* match correctly. Note that the assumption "*environment matches training*" is not addressed at this level, because the model does not yet talk about the hospital environment. However, when it is introduced at more detailed levels, the dependency in the other direction—staff depends on hospital to provide adequate working environment—will need justification.

Furthermore, depending on the scope of the system model, the "*staff is trained*" property could be explicitly modelled as well. In a similar manner, one could treat *Training* as another system, making "*staff is trained*" an assumption on the *Training* system. A failure in training could propagate into staff service and eventually into the hospital service. By extending the scope to address such assumptions, one could be addressing faults, not the consequences, e.g. staff is not trained enough, thus it is making mistakes and patient treatment goes wrong.

### 6.2.4 Human systems detailed

Following the top-down approach, human systems (*Staff*) and corresponding *Hospital* assumptions should be refined to an adequate level of detail. This case study will not do that explicitly but instead provides some hints and considerations for the process. The addition of further detail levels to human systems will have to satisfy the abstract properties, in this case, "*staff performs correctly.*"

When the model introduces specialist roles, one will have to face the questions: does the specialist role *always* perform its duty correctly; how critical is this role for the *correctness* of staff service?

There are various situations that may arise and prevent the designer from ensuring that assumptions are satisfied. For example, if the hospital has only a single specialist with a specific critical role, he becomes a single point of failure. If affected by illness, leaving the job or some other absence, the system as a whole can no longer "*perform duties correctly*". The situation should be approached with a failure mode mindset, e.g. the system designer must investigate possible redundancy or other measures: how quickly can an adequate replacement specialist be found, hired or trained; maybe it is more suitable to have another specialist (redundancy) from the

start? Furthermore, employees are rarely perfect and may fail accidentally. This would appear if the *Staff* system is decomposed into smaller, more detailed subsystems (even a single person could be treated as a system!). Again, to show correct refinement, one must ensure that such failures are caught and handled somehow, e.g. install a review process to the hospital staff operation.

Furthermore, to reiterate a point mentioned earlier in the case study, the approach is general enough to substitute any other system for the hospital. The staff considerations would be similar, for example, if the reference system was an electricity infrastructure.

### 6.2.5 Hospital decomposed

The earlier steps in this case study illustrated top-down development directions to describe infrastructure systems and human systems. To match the increasing level of detail, the hospital system itself would also need to be refined. One could do that by "pulling out" systems and keeping the hospital as a black box with assumptions necessary for correct service (as was illustrated by the *Staff* system). Further details could also be introduced by identifying the inner components and relationships of the hospital system. For example, to describe dependencies on national infrastructure systems in more detail, the model could introduce *Equipment*, *Building* and *Information* as component systems of *Hospital*. These sub-systems require external services, such as ICT or Energy, thus the dependencies are also introduced. The refinement here is done in two steps, which are sketched in Figure 6.4.

Along with the new sub-systems, their main properties are also specified. These properties have to refine the main property in *Hospital*: "*Hospital is operational.*"

**HospitalC ("components")**

| | |
|---|---|
| Systems: | *Equipment, Building, Information* |
| Assumption: | *[Assumptions on staff and infrastructure systems hold.]* |
| Property: | *Hospital* Equipment *is operational.* (`true`) (Hc-EqOper) |
| Property: | Building *is usable.* (`true`) (Hc-Usable) |
| Property: | *Patient medical* Information *is available and up-to-date.* (`true`) (Hc-Info) |

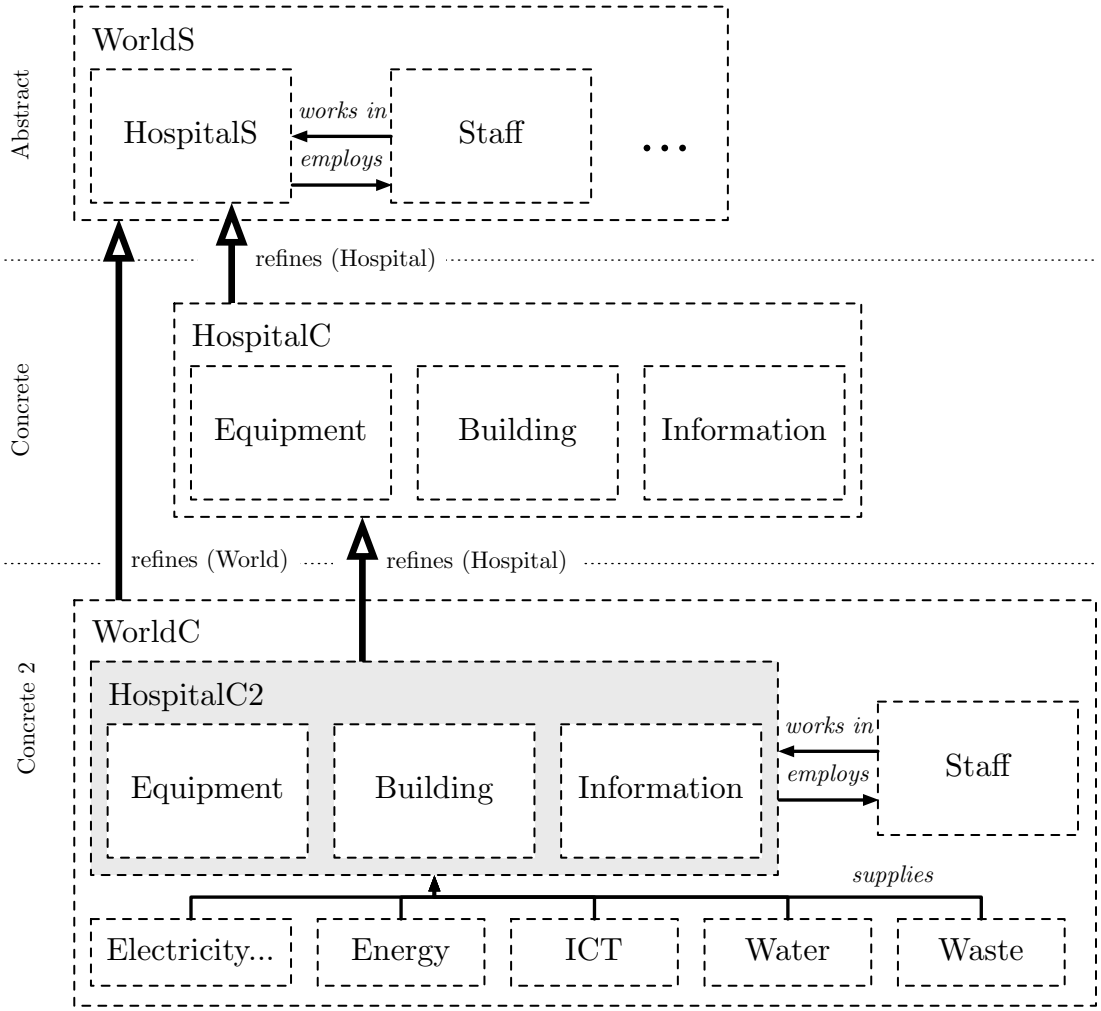


Figure 6.4: Two step refinement of *Hospital* system to identify its components and further dependencies.

The properties about *Equipment*, *Building* and *Information* refine the property "*Hospital is operational*" in *Hospital*. To justify the refinement, it is important to construct a *retrieve function* between the abstract and the concrete descriptions of the refined system. The next step is to show that this function is *total* and *adequate*. By proving these properties about the retrieve function, a *correct* data refinement is ensured (see Section 3.6.6 for more details). This case study, as explained at the beginning, does not aim for any formalisation in the model, but a quick sketch of refinement justification for this particular step is provided below.

For this example of refinement, consider just one property in the abstract *Hospital*, namely "Hospital is operational". It will be referred using its identifier H-Oper from here on. In the concrete *HospitalC* description, it is replaced by properties Hc-EqOper, Hc-Usable and Hc-Info. A retrieve function defines how the concrete descrip-

tion can be mapped back to the abstract one, i.e. function *retr-HospitalC* describes what abstract *Hospital* description corresponds to the given concrete *HospitalC*:

*retr-HospitalC: HospitalC* $\rightarrow$ *Hospital*

This retrieve function needs to account for all properties in *HospitalC* and given some (e.g. either correct or error) state of *HospitalC*, construct a corresponding (again, either correct or error) state of *Hospital*. In this case study, the properties of interest are mapped with the following rule: "*All properties about* Equipment*,* Building *and* Information *must hold (be* `true`*) for the* Hospital *to be operational, i.e. for the abstract property to be* `true`." This can be represented as a conjunction of the mentioned properties, e.g. the retrieve function *retr-HospitalC* is defined as:

Given *HospitalC(*Hc-EqOper*,* Hc-Usable*,* Hc-Info*, ...)*
results in *Hospital(*H-Oper*, ...)*
where H-Oper = Hc-EqOper and Hc-Usable and Hc-Info, ...

Showing that the data refinement is correct requires proving *totality* and *adequacy* of the retrieve function. The totality requires the retrieve function to be applicable for any concrete state *HospitalC*. For the properties of interest, showing totality is trivial, since the value for H-Oper can always be calculated from any property values in *HospitalC*. For adequacy, it is necessary to show that there exists a concrete state *HospitalC* for every abstract *Hospital*, i.e. that all abstract states are accounted for by the refinement. In general (disregarding the invariant), the abstract *Hospital* can have two states regarding property H-Oper, i.e. either it is `true` or `false`. Proving adequacy means finding concrete states for each, which is trivial. For example, the correct state where H-Oper=`true` is represented by *HospitalC* where all properties are also `true`. The `false` state is represented by any *HospitalC* state where at least one of the properties is `false`. Note that since *Hospital* and *HospitalC* actually only denote the *correct* states (their invariants require properties to always be `true`), showing adequacy and totality is even easier, since the properties can have only a single value. Because the retrieve function is both total and adequate, the refinement is correct.

Further details in *HospitalC2* refine the assumptions about dependencies on national infrastructures by linking them to the introduced subsystems. This adds precision to the hospital interface.

**HospitalC2 ("further component details")**

| | |
|---|---|
| Systems: | *Equipment, Building, Information* |
| Assumption: | Equipment *is supplied by electricity adequately and reliably.* |
| Property: | Equipment *is in working order.* |
| Assumption: | Information *is provided via ICT reliably.* |
| Property: | Information *is available and up-to-date in ICT databases.* |
| Assumption: | Building *is supplied by electricity adequately and reliably.* |
| Assumption: | Building *is supplied by heating adequately and reliably.* |
| Assumption: | Building *is supplied by water adequately and reliably.* |
| Assumption: | Building *is supplied by waste removal adequately and reliably.* |
| Property: | Building *is in working order.* |
| Assumption: | *[Assumptions on* Staff *hold.]* |
| Needs: | *Electricity, Energy, ICT, Water, Waste, Staff* |

The new properties and assumptions refine corresponding properties and assumptions in *HospitalC*. Refer to Figure 6.4 for the overview picture of both refinement steps presented in this section.

### 6.2.6 Hospital functions

The modelling steps above focus on relationships between systems and how to introduce details in the model while preserving the abstract properties. Often the designer also needs to record the actual function of the system: what it is doing, how does it affect the state of the system, etc. The state definition provides a context for function definition: it describes assumptions and properties that always hold for a *correct* system state.

This section provides some hints on how the hospital's function could be described. Figure 6.5 sketches the refinement that specifies additional objects and properties in *Hospital* as well as defines a function on the refined state.

To illustrate the specification of a function, a simpler example is chosen that requires introducing new objects to the model: *Patients* and *Budget*. They do not have obvious relationships with external systems but are necessary for the description of the function.

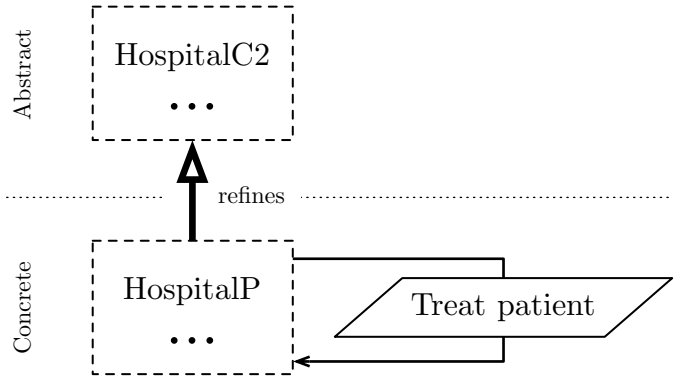


Figure 6.5: Refinement of *Hospital* system to define *Treat patient* function.

**HospitalP ("with patients")**

| | |
|---|---|
| Objects: | *Patients, Budget* |
| Property: | *There is a certain number of patients in the hospital.* |
| Property: | *Hospital can accommodate* $\text{Patients}_{\max}$ *number of patients.* |
| Property: | *A fixed budget* Budget *is available for patient treatment.* |
| Property: | *[Previous definitions on subsystems and their properties hold.]* |

A system function is normally defined over the correct state of the system. In the case study, it is described using a similar box to the state, but extended with several more property types. The *preconditions* and *postconditions* define what is needed for the function to performed, and what is the end result of the function having been completed, respectively. Note that the case of a precondition not holding means that the function cannot be performed, but does not mean that the system is erroneous: for example, one cannot "*pour water*" into a full *bucket* (precondition "*there is space for water*" does not hold), however the *bucket* being full is not an error state.

***Treat patient*** on **HospitalP**

| | |
|---|---|
| Property: | *The patient receives diagnosis and treatment for his illness.* |
| → Pre: | *Patient is ill.* |
| → Pre: | *There are available beds in the hospital.* |
| → Pre: | *There are available funds in the* Budget. |
| → Pre: | *Staff, equipment and information resources are available.* |
| Assumption: | *Patient has means to arrive at the hospital.* |
| ← Post: | *Patient received appropriate diagnosis and treatment.* |
| ← Post: | *Patient treatment costs are deducted from the* Budget. |
| Assumption: | *Patient has means to leave the hospital.* |

The main function "*treat patient*" could also be described as a chain of three operations: "*admit patient*", "*treat patient*", "*release patient*". The choice depends on what is intended to be described at the current level. The three operations would allow for explicit partitioning of the treatment process, and would provide intermediate states of the hospital. The current choice of "*treat patient*", for example, does not actually reflect beds being occupied by the patient, since it describes only when he is admitted and released. The reader should assume that the patient occupies a bed sometime during the function execution. Since the intermediate states of a patient being treated are not important at this level, the single-operation function description is chosen.

### 6.2.7 Modelling failure

A significant benefit of the top-down approach, especially when designing new systems, is the ability to specify the intended behaviour and features of the system at the abstract level: i.e. what the system is expected to be and do. The refinement steps that follow need to preserve the abstract properties, thus ensuring that even at low levels of system description the intended behaviour, dependability properties and other requirements are adhered to. The majority of this case study follows this approach of modelling a *correct* system: the refinement steps make sure that the abstract properties are satisfied. Regarding failure analysis, this approach ensures that there is no oversight in defining low-level system properties, i.e. the behaviour is covered by the specification at all times. This means that all system functions

have precise descriptions and all conditions needed for a successful operation of each function are known. Consequently, if the system is in a correct state and all preconditions of some function are satisfied, it is guaranteed to executed successfully and leave the system in a correct state.

The *correct* system approach aims to identify and eliminate faults during development. However, there are cases when modelling failure is useful. For example, to include external faults (hazards) into the model, it is important to describe them and how they affect the system. An external fault may lead to an erroneous state of the system. These states can be modelled within the proposed framework using the same properties as the *correct* states, but with values different from the ones describing the correct state. In this case study, such error states would have `false` property values, e.g. "*Hospital Equipment is operational*"=`false` (i.e. is *not* operational) in *HospitalC* (Section 6.2.5). The following box sketches such an error state as *HospitalC-ErrEquipment*.

**HospitalC-ErrEquipment ("equipment error")**

| | |
|---|---|
| Systems: | *Equipment, ...* |
| Property: | *Hospital* Equipment *is operational.* (`false`) |
| Property: | *[Other system property values may be unspecified.]* |
| Assumption: | *[Assumptions on staff and infrastructure systems may be unspecified.]* |

Note that there can be a large number of significant error states to accompany possibly a single correct state of the system.

Having defined error states of interest, external fault events can be modelled as special functions of the concerned system. For example, an external fault "*Break equipment*" would be described as a function on system *HospitalC* with a result state *HospitalC-ErrEquipment*, i.e. it takes the system from the correct to the error state. Such approach allows using the same techniques to describe both the correct operation and the various hazards.

While the external fault events take the system to an error state, corresponding functions of the system, which take the system from an error back to the correct state, can be used to describe recovery activities. For example, a "*Replace*

*non-operational equipment*" function would recover the system from *HospitalC-ErrEquipment* state back to the correct *HospitalC* state.

Finally, the concept of *failure* within this approach can be found in two places. Firstly, the correct system functions are defined for the correct system state. Therefore, if the system is in an error state, the preconditions of said functions do not hold and they should not be performed (or their result is undefined if they are executed anyway). If the system function cannot be performed, it is a failure of the system service. Secondly, the failures can be modelled explicitly, as system functions requiring an error state. Such function definition would give a precise description of an incorrect system service.

## 6.3 Planning considerations

In addition to modelling and describing the system itself, as illustrated in the steps above, one needs to consider some implications about how the system is constructed and deployed. Failures in system planning and design or the deployment processes may introduce faults into the final infrastructure system.

As described in Chapter 4, a failure of planning or design process could be making a bad design decision, introducing incorrect assumptions, etc. This would lead to faults in the planned system, because the correctness of the planned system rests on the correctness of initial assumptions and applied the development process.

To reason about the planning process, one does not necessarily need to switch to another analysis method. The planning process (and all its related notions) can also be accommodated in the "everything is a system" paradigm, thus reusing the techniques mentioned in the case study and the thesis.

**Planning**

| | |
|---|---|
| Property: | *Planning creates correct system specification.* |
| Assumption: | *All system requirements are known.* |
| Assumption: | *Planners perform their duties correctly.* |
| Assumption: | *Planners have enough expertise to make decisions about the planned system.* |

This abstract description of the planning system could go through the same

refinement and reasoning process as the planned system, some of these are outlined below:

- Extend to include human systems (design staff) and their training requirements;
- *PlanningOrg ("organisations")* could consist of several organisations developing the complex system-of-systems: treat each as a system and ensure that their interfaces match; *PlanningOrg* would refine *Planning*.
- Extend to include systems providing or ensuring expertise on infrastructure development (experts, tools, etc);
- System designers or whole organisations can manifest failures (bad decision, leave project, etc), so adequate fault tolerance (e.g. redundancy) should be considered for the planning process. This can be experts reviewing the system design, having replacement organisations or parallel interchangeable designs of the system, etc.

Note that the above extensions must satisfy the top-down approach and correct system refinement principles: they must all satisfy the assumptions and properties of the original abstract system, e.g. "*creates correct specification.*"

The assumptions recorded about the modelled system must be addressed during deployment. When configuring a complex system-of-systems, some of the assumptions are matched by the configured systems (e.g. "*requires reliable electricity source*" and "*provides reliable electricity*" in Section 6.2.2). Other assumptions are not addressed fully or are left completely unaddressed. They become assumptions of the whole system-of-systems and must be satisfied during deployment. For example, if an assumption requires that "*hospital has a car park within* X *distance*", the deployment process must verify this to be true when building the whole system-of-systems (hospital with infrastructure). Otherwise, the assumption is broken and thus the whole design may no longer hold.

# Chapter 7

# Conclusions and future work

This thesis proposes ideas for a framework to describe, analyse and design national infrastructure systems with a focus on their dependability. The established ideas from dependable computing research can be adapted to reason about infrastructure as well as other complex systems. The thesis has explored their applicability and benefits in improving clarity and preciseness of description and analysis of infrastructure systems, their relationships and failure propagation. Further computing science views, such as top-down thinking and formal specification for infrastructure systems, have been investigated to address complexity issues in modern infrastructures. Additionally, this thesis proposes to extend the scope of these ideas to include related failure-originating systems, e.g. human operator as human systems. This also covers systems which generate other systems, e.g. planning, design, construction and maintenance systems. Failures in such generating systems give a rise to latent faults and errors in the generated systems. This chapter summarises the advantages and applications of the proposed framework as well as explores avenues to complete and enrich it further.

## 7.1 Improvements to description and analysis of infrastructure systems

The research hypotheses (Chapter 1) propose to use dependable computing concepts to reason about infrastructures. Specific proposed avenues include using the top-down approach to tackle infrastructure complexity as well as considering planning systems in the overall analysis. These hypotheses comprise the main aspects of the reasoning approach proposed in this thesis and their application is investigated to a certain extent by the examples and the case study.

The main H1 hypothesis of this thesis suggests that concepts from dependable computing can clarify interdependencies between infrastructure system and this clarification would lead to increased dependability of infrastructures. Following this hypothesis, the thesis re-examines the meanings and descriptions of failure and failure propagation in infrastructure systems. These are central concepts to dependability analysis and thus need precise meanings to facilitate their use, constructive discussions and analysis in complex systems. The notion of a *fault-error-failure* chain is an established concept in dependable computing research and is used to describe the lifecycle of failure and its propagation (see Section 2.2). The thesis explores how these concepts enrich description of failure in infrastructure systems as well as help to identify how failure propagation manifests between systems. The notion of *failure* cannot be taken as an absolute concept such as "system stopped working"—its description needs to consider how the failure is judged (perceived) by other systems, what is the correct or intended service of the system, etc. When linked with system specification, failure is disambiguated. The identification of *errors* and *faults* in infrastructure systems brings benefits both in clearer understanding of cause of failure and in pointing towards appropriate fault tolerance measures to address them. These activities would lead to increased dependability of the analysed infrastructure systems, as proposed in hypothesis H1.

As proposed in the H2 hypothesis, top-down development techniques (Chapter 3) are aimed at tackling complexity in national infrastructure systems and their analysis. A top-down view provides a different perspective to mostly bottom-up approaches used in infrastructure interdependency analysis (their overview is in Section 5.2). Note that such bottom-up techniques are natural for existing infrastructure systems with networks of interconnected low-level physical components. The top-down view, as argued in this thesis, could provide an alternative way of approaching system complexity and establishing the required infrastructure system properties at the start before designing the implementations to satisfy them. The thesis argues that different levels of abstraction in top-down view facilitate reasoning about the infrastructure systems at a chosen level of detail. The properties of interest, even spanning complex systems, such as system dependability, could be defined and analysed in a model uncluttered by unnecessary details. The details are added at subsequent levels of abstraction while ensuring that abstract properties still hold (cf. reification). In such a way system complexity is tackled step-by-step advancing through different levels of abstraction. While several examples within this

thesis aim to illustrate such application to infrastructures, fully evaluating the benefits of the top-down approach to infrastructure systems would, however, require major resources for a full system development. Top-down approach applications, especially complemented with formal methods, have shown success within the dependable computing field. This gives credibility to the success of the proposed ideas in applications to national infrastructure systems.

The importance of assumptions is emphasised in the thesis (Section 3.2). They are often overlooked and/or not recorded, leading to mismatches between systems about what is required for intended operation. Illustrating with examples from failures in infrastructure systems, the thesis argues that all assumptions must be identified and recorded to enable an informed and correct argument about the system under consideration. This aims to address a significant proportion of failures in infrastructures especially when concerned with complex systems-of-systems.

Furthermore, the thesis explores usage of (formal) specification from dependable computing research to provide a framework of system description that includes assumptions, system properties, description of correct service and techniques to evolve the specification via data reification. A system can be described at its boundaries and its relationship with other systems can be formally expressed as interface contracts. The increased precision will give rise to well-grounded arguments about infrastructure system service, dependability and failure propagation. Furthermore, this approach provides a framework to identify and address faults in the system during the development process.

The thesis argues that these dependable computing techniques would benefit from the description and analysis of infrastructure systems and their dependability. Furthermore, this approach could be expanded beyond the conventional systems, as proposed in the H3 hypothesis. The argument is that related entities can be treated as systems and included in the overall analysis in a similar manner. The approach works for considering human operators in infrastructure systems (*human systems*—Section 2.4), as well as planning, design, construction and maintenance processes of infrastructure systems (*systems generating systems*—Chapter 4). These ideas allow construction of a complete context for infrastructure system dependability analysis, e.g. failure propagation origins can be traced to failures and then faults in human or planning systems. This allows consideration of fault tolerance in the failure-originating systems, e.g. address operator training; review the planning results, etc.

This thesis explores different examples to illustrate how the proposed framework can be used for description and dependability analysis of national infrastructure systems (e.g. Sections 2.6, 3.6, Chapter 6). They show how the concepts from dependability research can be transferred and expanded upon to infrastructure systems and lay groundwork for full analysis of such systems.

The ideas and techniques presented in this thesis together with practices in dependable computing could help to understand infrastructure systems better and such understanding could increase dependability of infrastructure systems. The main evidence to support the claimed benefit to infrastructure systems dependability comes from the dependable computing field. The proposed techniques and concepts form an established reasoning approach there and have been used successfully to design and analyse complex computing systems. The main contribution of this thesis is illustrating how they can be adapted (with good chance of inheriting all the benefits) to infrastructure systems. A detailed comparison with established infrastructure analysis approaches (Section 5.2) would involve a major study that is out of the scope of an MPhil thesis. Regardless, this thesis aims at providing an *alternative* approach to infrastructure analysis. A different view of the systems could lead to new insights and raise questions different from the ones in current infrastructure analysis methods.

## 7.2 Future work

The proposed techniques provide a deeper understanding of interdependent infrastructures and failure propagation. The thesis shows that the ideas from dependable computing research constitute a basis for a framework for infrastructure system description and dependability analysis. The benefits and potential of the approach are illustrated in the conclusions above as well as in the whole thesis. To achieve a mature framework, however, further work is required. While examples in the thesis serve well to illustrate the ideas, a full top-down modelling and specification of (at least partial) infrastructure systems or system-of-systems should be undertaken. Research within ITRC has collected data about current and future states of infrastructure, which would serve well for a detailed case study. The exercise would evaluate the framework and fine-tune concepts used and suggested practices.

Adding formalisms to the model and specification of infrastructure systems improves precision. By using full formal methods, one can benefit from the logical

reasoning framework and formal semantics of mathematical description to gain high assurance in the specification. Furthermore, tool support exists to ensure consistency of the model and to verify properties of the system under consideration. The framework proposed in this thesis would benefit from such formalisms and tool support. Further work is needed, however, to evaluate which formal methods are most suitable for infrastructure systems, what additional formalisms are necessary to describe and reason about such systems, etc. Jones et al. [JHJ07] have already shown that VDM specifications and rely-guarantee reasoning can be used to describe real-world systems. Formal methods use for describing various systems-of-systems is currently being investigated within the COMPASS project (e.g. in [FBP12]).

Dependability research in computing science offers further established approaches that can be adapted for use in infrastructure systems. These can also be explored as future work to include in the proposed framework or as alternative approaches. The *problem frames* approach suggests the specification of each property of interest separately instead of trying to accommodate them in a single hierarchy or view [Jac00a]. This allows us to focus on relevant details only (per property) in the analysis. Furthermore, the problem description requires all entities to be defined, thus ensuring that all necessary relationships are considered.

# Bibliography


[Abr10] Jean-Raymond Abrial. *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.

[ALRL04] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[CB08] David Copper and Janet Barnes. Tokeneer ID station EAL5 demonstrator: Summary report. Technical Report S.P1229.81.1 Issue: 1.1, Altran-Praxis, August 2008.

[CBB+11] Emiliano Casalicchio, Sandro Bologna, Luigi Brasca, Stefano Buschi, Emanuele Ciapessoni, Gregorio D'Agostino, Vincenzo Fioriti, and Federico Morabito. Inter-dependency assessment in the ict-ps network: The mia project results. In Christos Xenakis and Stephen Wolthusen, editors, *Critical Information Infrastructures Security*, volume 6712 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2011.

[Cia04] Emanuele Ciapessoni. Formal methods to handle the complexity of automation and protection systems. *Bulk Power System Dynamics and Control - VI, August 22-27, 2004, Cortina d'Ampezzo, Italy*, 2004.

[CMS+07] E. Ciancamerla, M. Minichino, W. Schmitz, T. Uusitalo, R. Linnemann, D. Wright, C. Kühnert, L. Issacharoff, and L. Buzna. Irriis: Tools and techniques for interdependency analysis. D2.2.2, ENEA, 2007.

[Cou12] Council of the European Union. Council recommendation of on the national reform programme 2012 of the united kingdom and delivering a council opinion on the convergence programme of the united kingdom, 2012-2017. Technical Report 11276/12, Brussels, 2012.

[DK04] G. Despotou and T. Kelly. Extending the safety case concept to address dependability. In *In proceedings of the 22nd International System Safety Conference (ISSC), Providence, RI USA, proceedings by System Safety Society 2004*, 2004.

[DPM06] D.D. Dudenhoeffer, M.R. Permann, and M. Manic. Cims: A framework for infrastructure interdependency modeling and analysis. In *Simulation Conference, 2006. WSC 06. Proceedings of the Winter*, pages 478–485, 2006.

[EHK08] Irene Eusgeld, David Henzi, and Wolfgang Kröger. Comparative evaluation of modeling and simulation techniques for interdependent critical infrastructures. Scientific report, ETH, 2008.

[Eme11] Emerson Network Power. Understanding the cost of data center downtime: An analysis of the financial impact on infrastructure vulnerability. Technical Report SL-24661-R05-11, 2011.

[FBP12] John S. Fitzgerald, Jeremy Bryans, and Richard Payne. A formal model-based approach to engineering systems-of-systems. In Luis M. Camarinha-Matos, Lai Xu, and Hamideh Afsarmanesh, editors, *PRO-VE*, volume 380, pages 53–62, 2012.

[Fer92] ES Ferguson. *Engineering and the Mind's Eye*. The MIT Press, 1992. Page 183 and note 25 on page 225.

[For04] U.S.-Canada Power System Outage Task Force. Final report on the august 14, 2003 blackout in the United States and Canada: Causes and Recommendations. Technical report, April 2004.

[GIJ+02] M.-C. Gaudel, V. Issarny, C. B. Jones, H. Kopetz, E. Marsden, N. Moffat, M. Paulitsch, D. Powell, B. Randell, A. Romanovsky, R.J. Stroud, and F. Taini. Final version of DSoS conceptual model. Technical Report CS-TR-782, School of Computing Science, Newcastle University, 2002.

[HB95] Michael G. Hinchey and Jonathan P. Bowen, editors. *Aplications of Formal Methods*. Prentice Hall International, 1995.

[HDO11] I. Hernández and L. Dueñas-Osorio. Sequential propagation of seismic fragility across interdependent lifeline systems. *Earthquake Spectra*, 27(1):23–43, 2011.

[Hel07] Tomas Hellstrom. Critical infrastructure and systemic vulnerability: Towards a planning framework. *Safety Science*, 45(3), 2007.

[HG99] J.D. Herbsleb and R.E. Grinter. Architectures, coordination, and distance: Conway's law and beyond. *Software, IEEE*, 16(5):63–70, 1999.

[HHHN12] J.W. Hall, J.J. Henriques, A.J. Hickford, and R.J. Nicholls, editors. *A Fast Track Analysis of strategies for infrastructure provision in Great Britain: Technical report*. Environmental Change Institute, University of Oxford, 2012.

[HJ01] Y. Haimes and P. Jiang. Leontief-based model of risk in complex interconnected infrastructure. *Journal of Infrastructure Systems*, 7(1):1–12, 2001.

[HJJ03] Ian Hayes, Michael Jackson, and Cliff Jones. Determining the specification of a control system from that of its environment. In Keijiro Araki, Stefani Gnesi, and Dino Mandrioli, editors, *FME 2003: Formal Methods*, volume 2805 of *LNCS*, pages 154–169. Springer Verlag, 2003.

[HK91] Ian Houston and Steve King. *Proceedings of the 4th International Symposium of VDM Europe on Formal Software Development-Volume I: Conference Contributions - Volume I*, chapter CICS Project Report: Experiences and Results from the use of Z in IBM, pages 588–596. VDM '91. Springer-Verlag, 1991.

[Hoa81] Charles Antony Richard Hoare. The emperor's old clothes. *Commun. ACM*, 24(2):75–83, 1981.

[HSC+08] Yacov Haimes, Joost Santos, Kenneth Crowther, Matthew Henry, Chenyang Lian, and Zhenyu Yan. Risk analysis in interdependent infrastructures. In Eric Goetz and Sujeet Shenoi, editors, *Critical Infrastructure Protection*, volume 253 of *IFIP International Federation for Information Processing*, pages 297–310. Springer US, 2008.

[Jac82] Michael A Jackson. A system development method. *Tools and Notions for Program Construction*, pages 1–26, 1982.

[Jac00a] Michael Jackson. *Problem Frames: Analyzing and structuring software development problems*. Addison-Wesley, 2000.

[Jac00b] Michael Jackson. The real world. In J Davies, B Roscoe, and J Woodcock, editors, *Millennial Perspectives in Computer Science: Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare (Cornerstones of computing)*. Palgrave Macmillan, 2000.

[JH04] Pu Jiang and Yacov Y. Haimes. Risk management for Leontief-based interdependent systems. *Risk Analysis*, 24(5):1215–1229, 2004.

[JHJ07] Cliff B. Jones, Ian J. Hayes, and Michael A. Jackson. Deriving specifications for systems that are connected to the physical world. In Cliff B. Jones, Zhiming Liu, and Jim Woodcock, editors, *Formal Methods and Hybrid Real-Time Systems: Essays in Honour of Dines Bjørner and Zhou Chaochen on the Occasion of Their 70th Birthdays*, volume 4700 of *Lecture Notes in Computer Science*, pages 364–390. Springer Verlag, 2007.

[Jon81] C. B. Jones. *Development Methods for Computer Programs including a Notion of Interference*. PhD thesis, Oxford University, June 1981. Printed as: Programming Research Group, Technical Monograph 25.

[Jon90] C. B. Jones. *Systematic Software Development using VDM*. Prentice Hall International, second edition, 1990. ISBN 0-13-880733-7.

[Jon03] Cliff B Jones. A formal basis for some dependability notions. In Bernhard K. Aichernig and Tom Maibaum, editors, *Formal Methods at the Crossroads: from Panacea to Foundational Support*, volume 2757 of *Lecture Notes in Computer Science*, pages 191–206. Springer Verlag, 2003.

[Jon05] Cliff Jones. Specification before satisfaction: The case for research into obtaining the right specification—extended abstract—. In Helen Treharne, Steve King, Martin Henson, and Steve Schneider, editors, *ZB 2005: Formal Specification and Development in Z and B*, volume 3455 of *Lecture Notes in Computer Science*, pages 1–5. Springer Berlin / Heidelberg, 2005. 10.1007/114157871.

[JR06] Cliff Jones and Brian Randell. The role of structure: a dependability perspective. In Denis Besnard, Cristina Gacek, and Cliff B. Jones, editors, *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*, pages 3–15. Springer London, 2006. 10.1007/1-84628-111-3-1.

[JW08] Cliff B. Jones and Jim Woodcock, editors. *Formal Aspects of Computing – Special Issue on the Mondex Verification*, volume 20. Springer London, January 2008.

[Kop06] H. Kopetz. On the fault hypothesis for a safety-critical real-time system. In Manfred Broy, Ingolf Krüger, and Michael Meisinger, editors, *Automotive Software – Connected Services in Mobile Networks*, volume 4147 of *Lecture Notes in Computer Science*, pages 31–42. Springer Berlin / Heidelberg, 2006.

[Kop11] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer Science and Business Media, LLC, Boston, MA, 2011.

[KW04] Tim Kelly and Rob Weaver. The goal structuring notation—a safety argument notation. In *Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.

[KZ11] Wolfgang Kröger and Enrico Zio. *Vulnerable Systems*. Springer London Dordrecht Heidelberg New York, 2011.

[Lev11] Nancy G. Leveson. Applying systems thinking to analyze and learn from events. *Safety Science*, 49(1):55 – 64, 2011.

[Lit02] Richard G. Little. Controlling cascading failure: Understanding the vulnerabilities of interconnected infrastructures. *Journal of Urban Technology*, 9(1):109–123, 2002.

[Lit03] R.G. Little. Toward more robust infrastructure: observations on improving the resilience and reliability of critical systems. In *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, 2003.

[LMW04] E.E. Lee, J.E. Mitchell, and W.A. Wallace. Assessing vulnerability of proposed designs for interdependent infrastructure systems. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, 2004.

[Mai98] M. W. Maier. Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284, 1998.

[Mas06] M. Masera. Interdependencies and security assessment: a dependability view. In *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on*, volume 1, pages 585–589, October 2006.

[MC10] L. Marques and A. Casimiro. Data validity and dependable perception in networked sensor-based systems. In *Reliable Distributed Systems, 2010 29th IEEE Symposium on*, pages 358 –362, 2010.

[MIA10] MIA. Definition of a methodology for the assessment of mutual interdependencies between ICT and electricity generation/transmission infrastructures. Technical Report Final report, Italian National Agency for New Technology, Energy and Environment, Italy, 2010.

[NAS13] NASA. Mars climate orbiter mishap investigation board. Technical Report Phase I Report, November 10, 1999. Retrieved 2011-01-13.

[Nat07] National Transportation Safety Board. Runway overrun and collision, southwest airlines flight 1248, boeing 737-7h4, n471wn, chicago midway international airport, chicago, illinois, december 8, 2005. Technical Report Aircraft Accident Report NTSB/AAR-07/06, Washington, DC, 2007.

[PBFR12] Richard Payne, Jeremy Bryans, John Fitzgerald, and Steve Riddle. Interface specification for system-of-systems architectures. Technical Report CS-TR-1323, Newcastle University, 2012.

[PDHP06] P. Pederson, D. Dudenhoeffer, S. Hartley, and M. Permann. Critical infrastructure interdependency modeling: A survey of U.S. and international research. Technical report, Idaho National Laboratory (INL), 2006.

[Per84] C. Perrow. *Normal Accidents: Living With High-Risk Technologies*. Basic Books, Inc., New York, USA, 1984.

[PF07] James P. Peerenboom and Ronald E. Fisher. Analyzing cross-sector interdependencies. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, 2007.

[PM1] Prime minister's speech on national infrastructure was delivered at the institute of civil engineering: http://www.number10.gov.uk/news/pm-speech-on-infrastructure/.

[Ran00] B. Randell. Facing up to faults. *The Computer Journal*, 43(2):95–106, 2000.

[RD06] T. Rigole and G. Deconinck. A survey on modeling and simulation of interdependent critical infrastructures. In *Proc. of the 3rd IEEE Benelux Young Researchers Symp. on Electrical Power Engineering*, pages 27–28, 2006.

[Rea90] James Reason. *Human Error*. Cambridge University Press, 1990.

[Rea97] James Reason. *Managing the Risks of Organisational Accidents*. Ashgate Publishing Limited, 1997.

[Rea00] James Reason. Human error: models and management. *BMJ*, 320(7237):768–770, 2000.

[Rin04] S.M. Rinaldi. Modeling and simulating critical infrastructures and their interdependencies. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, 2004.

[RIT+08] V. Rosato, L. Issacharoff, F. Tiriticco, S. Meloni, S. Porcellinis, and R. Setola. Modelling interdependent infrastructures using interacting dynamical models. *International Journal of Critical Infrastructures*, 4(1):63–79, 01 2008.

[RPK01] S.M. Rinaldi, J.P. Peerenboom, and T.K. Kelly. Identifying, understanding, and analyzing critical infrastructure interdependencies. *Control Systems, IEEE*, 21(6):11–25, December 2001.

[RT13] Alexander Romanovsky and Martyn Thomas, editors. *Industrial Deployment of System Engineering Methods*. Springer, 2013.

[RV06] Stephen Reinach and Alex Viale. Application of a human error framework to conduct train accident/incident investigations. *Accident Analysis & Prevention*, 38(2):396–406, 3 2006.

[SCW00] Susan Stepney, David Cooper, and Jim Woodcock. An electronic purse: Specification, refinement, and proof. Technical monograph PRG-126, Oxford University Computing Laboratory, July 2000.

[SW07] Nils Kalstad Svendsen and Stephen D. Wolthusen. Connectivity models of interdependency in mixed-type critical infrastructure networks. *Information Security Technical Report*, 12(1):44–55, 2007.

[Tre11] H.M. Treasury. National infrastructure plan 2011. Technical report, Great Britain: H.M. Treasury, Infrastructure UK, November 2011.

[VJ11] R. Velykienė and C.B. Jones. A fast track analysis of ICT constraints on evolving physical infrastructure. Technical Report CS-TR-1282, School of Computing Science, Newcastle University, UK, 2011.

[WD96] Jim Woodcock and Jim Davies. *Using Z: Specification, Refinement, and Proof*. Prentice Hall International, March 1996.

[WLBF09] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods: Practice and experience. *ACM Comput. Surv.*, 41(4):19:1–19:36, October 2009.

[Zim04] R. Zimmerman. Decision-making and the vulnerability of interdependent critical infrastructure. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 5, pages 4059–4063, 2004.