Diagnosis Properties by Design

Alban Grastien [1,2]

[1] *NICTA, Canberra Research Laboratory, Australia*
alban.grastien@nicta.com.au
[2] *The Australian National University, Canberra, Australia*

1 INTRODUCTION

1.1 Diagnosis Properties

The main focus of the diagnosis community has historically been to find appropriate mathematical and logical frameworks to model network[1] behaviours and to develop associated algorithms. As our understanding of these frameworks increased and as solutions were found to many diagnosis problems, the interest shifted to *properties* associated with diagnosis. Studying network properties pertaining to diagnosis provides useful feedback to the designer who may improve the network specifications from the diagnosis perpective.

A typical useful property is that of diagnosability. A fault is diagnosable if it can be precisely identified when it takes place; the network is diagnosable if all its possible faults are diagnosable. Designers should make sure their network is diagnosable, as being able to diagnose precisely any fault makes it simpler to fix it.

Studying diagnosis properties can provide useful recommandations for the network design. However, the network design and, even more, the network model are often available at a very late stage, and the recommandations expressed by diagnosticians must often be ignored. This issue can be illustrated by an example coming from the power grids. When a fault takes place on the grid, for instance a line-to-earth fault, the control-room operators immediately reconfigure the network in order to restore electricity to as many custumers as possible and to minimize the outage before the line is repaired. Such reconfigurations must be as fast as possible and diagnosability issues are clearly relegated to a secondary position. We would like this study to be performed beforehand, so that the network properties are ensured.

[1]In this paper, we consider systems that are built as a composition of components; the term "network" will therefore refer to the system we want to diagnose.

1.2 Design Rules

In this paper, we propose a different approach to ensuring diagnosis properties. Instead of testing whether a complete and final network model satisfies the required properties, suppose we could formulate a set of *design rules* and prove that *any* network built according to these rules does exhibit the properties. (In power grids, such design rules would e.g. specify that all transformers should be protected by circuit breakers with or without autoreclosing capabilities; that each circuit breaker upstream of a transformer should be equipped with a fault sensor; etc.) Being able to ensure diagnosis properties at every stage of a network's design, as opposed to merely testing for them after the fact, yields many advantages: First, the early detection of property violation gives an opportunity for the designer to incorporate the diagnostician recommandations in the final version of the network. Second when the network must be urgently modified, as is the case in power grids, it suffices to satisfy the design rules in order to ensure the diagnosis properties. Similarly, after the design rules have been proved appropriate for diagnosis, they can be reused as a basis for the next version of the network.

Our ambition stretches further than just providing a set of rules for a specific application domain, like power grids. What we would like is an algorithm that takes as input a description of the elements of the domain (such as component types, faults of interest, etc.) and a set of proposed rules, and determines if these rules are sufficient to ensure whatever property we are interested in. Even better, an algorithm that synthesises a set of rules, which not only ensure the expected properties, but do this at a minimal cost (e.g., that adds the least restrictions on the system design, the fewest extra sensor components, etc.)

This problem is challenging as it corresponds to testing properties on the family of networks (rather than a single network) that satisfy the design rules, or exhibiting one such network that does not satisfy the property. This family may

be infinitely large, and a similar problem for model-checking was proved undecidable in general (Browne *et al.*, 1989). Efficient implementations should be able to identify chunks of the family of networks that can be pruned, until the property has been assessed for each of its members.

2 EXAMPLE

We illustrate this paper with a simple example inspired by power networks at distribution level. We consider a single feeder which distributes electricity from a high voltage source to local substations in a tree-like structure. We distinguish the following components:

- the (single) *source* (`SRC`) provides electricity to the feeder;
- *lines* (`LN`) transmit electricity on long distances but are subject to fault;
- *circuit breakers* (`CB`) transmit electricity and operate to isolate faults (observable);
- *branching points* (`BR`) transmit electricity from one input to several outputs;
- *sensors* (`SNS`) are attached to lines or branching points and can detect faults downstream (including the current line, observable);
- *loads* (`LD`) represent local substations that receive electricity.

The source must be protected by a circuit breaker to prevent faults to propagate to the high voltage level, but, at this stage, no other assumption on the network configuration is made.

We assume that only lines can fail through permanent line-to-earth faults. When this happens, the tension drops and a current of dangerous intensity flows from the source to the fault. All circuit breakers up-stream would eventually detect the fault, but the closest one reacts first and operates by opening to isolate the fault. All sensors up-stream would also detect the fault but only the sensors between the fault and the circuit breaker that operates have time to notice the fault. We also make a single fault assumption.

Fault detection is trivial: a fault is detected when a circuit breaker trips. Diagnosis can be performed as follows: 1) the fault took place in a line in the subnetwork rooted in the triggered circuit breaker; 2) all lines directly protected by a circuit breaker that did not operate can be exonerated; 3) similarly, if the sensor immediately upstream a line did not generate an alarm, then the line is not faulty. We want to be able to diagnose precisely on which line the fault took place. Clearly, in general, the network is not diagnosable (consider for instance a network with several lines, only one circuit breaker, and no sensor).

We now propose to add design rules to ensure diagnosability. Design rules were already implicitely presented to define, e.g., how electrical components are connected in a tree, or how the source is protected by a circuit breaker. We propose to add the following design rules:

`RL1` if two lines are connected together, then a sensor is attached to the one down-stream;

`RL2` for any branching point, all outputs are either protected with a circuit breaker or monitored by a sensor.

It is trivial to show that, under these design rules, the network is diagnosable. Indeed, whenever a fault occurs, rule `RL2` allows to determine precisely the branch where the fault occurs, i.e., the part of the tree between two branching points or between a branching point and the source or a load. Furthermore, rule `RL1` allows to determine which line in this branch is faulty.

3 DIAGNOSIS PROPERTIES

The most common property that is studied in the context of diagnosis is the diagnosability of a network or of a fault. A fault is *diagnosable* if, whenever it occurs, it can be precisely identified by the diagnosis engine. The network is diagnosable if all its faults are diagnosable.

Variants of diagnosability have been defined. A requirement could be that the fault must be diagnosed before it develops in an unacceptable situation. The property can also be stated as being able to determine an appropriate recovery action after a fault took place (Cordier *et al.*, 2007). Pinpointing precisely the fault is not necessary, as long as the diagnosis is precise enough to decide how its consequences can be confined.

When the diagnosis engine can probe the network with an associated cost, the desired property is that there exists a strategy such that for any behaviour of the network, its fault mode can be determined with acceptable cost.

When the designer is concerned with tolerance to faults, the notion of $N-k$ (read "N *minus* k") property was defined. A network is $N-k$ if its behaviour remains stationary even if up to k faults take place. Certain parts of the network may not work properly, but the faults do not affect the network's integrity. This property is commonly used in power networks where faults cannot be helped; most blackouts of nation-scale have been caused by at least two major incidents (here, $k=1$).

More recently, the concept of diagnosis cost was introduced, where the cost is defined as the computational resources (memory, time) necessary to generate a diagnosis machine and to perform diagnosis with acceptable accuracy (Ribot *et al.*, 2007). In the context of discrete-event systems for instance, the Sampath diagnoser can be automatically generated from the model but its size is double exponential in the number of components (Rintanen, 2007) which means it cannot be built save for trivial networks; on the other hand, chronicles (Cordier and Dousson, 2000) can be more compact but their precision varies (Pencolé and Subias, 2009). In this context, a diagnosis property could be the existence of a procedure to generate simple yet accurate chronicles.

4 DESIGN RULES

We consider networks that are built as the composition of interconnected components. In many contexts, such as power grids, telecommunication networks, water distribution networks, assembly lines, electrical wiring in vehicules, etc., there are many standard components that can be composed with a great flexibility.

Let $\mathcal{N}$ represent the set of possible compositions, i.e., networks. Some networks are not advisable for diagnosis. Our purpose is to *restrict* set $\mathcal{N}$ in order to avoid such unappropriate compositions. Let us write $\mathcal{F} \subseteq \mathcal{N}$ the family (set) of networks that are acceptable. The restriction R on the set of networks defines a family of networks $\mathcal{F}_R$. If the restriction ensures all unadvisable networks are rejected, then the following property should be satisfied: $\mathcal{F}_R \subseteq \mathcal{F}$. Because in general $\mathcal{F}_R \neq \mathcal{F}$, networks in $\mathcal{F} \setminus \mathcal{F}_R$ are acceptable but are rejected by restriction R.

However, restriction R should be presented in a sensible and easy to understand format. Choosing an obscure restriction would be equivalent to providing only a decision procedure R_d for membership of networks to the accepted family:

$$R_d : \mathcal{N} \to \mathbf{B} \quad \text{where } R_d(n) = \top \text{ only if } n \in \mathcal{F}.$$

A "black box" restriction R would not be very attractive for the designer who would not understand what needs to be changed in a network $n \notin \mathcal{F}_R$. This is even more crucial for networks that need to be reconfigured on the fly whilst an incident is taking place, as the restrictions should drive the reconfiguration rather than slow it down.

In order to fulfil their purposes, compliance to the design rules must therefore be easy to test, both computationally but also for a human being. Ideally, design rules should avoid complex and combinatorial inter-dependencies and should involve simple, local decisions. We want to emphasize the existence of a trade-off here.

- On the one hand, if we try to use a formalism that allows us to match precisely the family of networks $\mathcal{F}$, i.e., such that $\mathcal{F}_R = \mathcal{F}$, then the design rules will be hard to understand.
- On the other hand, the more we use simple (but restrictive) design rules, the more acceptable networks will be rejected.

We definitely require simple design rules but one task of the diagnosis community will be to determine what types of rules are more appropriate for certain types of properties and networks.

Furthermore, we want to be able to do incremental modifications, for instance if the network must be reconfigured or some minor upgrade is expected. The network should be robust to these changes, i.e., for any local modification either i) it should not affect the rest of the network or ii) the cascade of modifications associated with it should be limited. For instance, assume that the network forms a tree and that exactly one branch of the tree must satisfy a specific property (e.g., the branch has some control over the network), and assume this is implemented by a particular type of component at the leaf of the particular branch. One might want to give control to another branch; this means that removing the controlling component at the leaf of the branch has an effect at the other end of the network where another component has to be connected. In this example, the cascade of modifications is still limited.

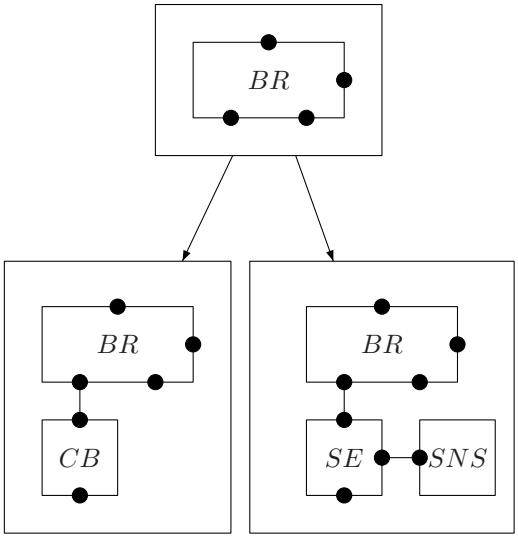

Figure 1: Design rule for the first output of a branching point.

Some Design Rules Haslum proposed in 2008 a similar approach to define classes of planning problems with polynomial complexity. He defined the restrictions of the network's structure in terms of graph grammars. A graph grammar is a formal rewriting system that operates on graphs; it is defined by an initial graph and a set of production rules which define how a graph can be transformed. A graph (or network) is accepted by the graph grammar if it can be derived from the initial graph by applying a finite sequence of transformations. Haslum proposes to use NLC grammars for which parsing can be tested in polynomial time, and showed that solving a planning problem on such a planning problem is polynomial as well.

Another possible formalism for design rules is to use pattern recognition. This is illustrated on Figure 1 which can be interpreted as follows. Whenever a branching point is found on the network (represented by the lone branching point in the top box), then its left output is either connected to a circuit breaker (bottom left box) or connected to a sensitive equipment (which may be a line or another branching point) monitored by a sensor (bottom right box). Design rules based on pattern recognition are simple to verify, including for a human. Furthermore, if one such rule is violated, for instance if the left output of a branching point is directly connected to another branching point, then the rule violation is local and can be fixed by

local modifications of the network.

Notice the limitations of both formalisms. NLC graph grammars for instance are not able to represent any lattice graph. Similarly, the pattern-based formalism cannot express in general that a graph should form a tree.

5 PROBLEM DEFINITION

5.1 Diagnosis Properties by Design

The problem we would like to solve can be formulated as follows:

> Given a set of component types C (which implicitely defines the set of networks $\mathcal{N}$ that can be created from the composition of such components), given some design rules that define a restricted family $\mathcal{F}_R$ of networks that satisfy the rules, does any network $n \in \mathcal{F}_R$ accepted by these design rules satisfy some predefined property?

If the property by design is satisfied, then any network that satisfies the design rules also satisfies the property. This means that the designer now only needs to concentrate on satisfying the design rules. By definition, it is more difficult to meet the design rules than to satisfy the expected property; however, because the design rules are (as we proposed in the previous section) expressed in a clear and understandable way, the designer will immediately detect when and identify why a design rule is not satisfied.

We also want to return useful feedback to the designer in case the design rules he gave us are not sufficiently restrictive. When this is the case, the algorithm should return a network $n \in \mathcal{F}_R$ for which the property is not satisfied; if the algorithm is not able to return a definite answer, it should return a partially instantiated network for which it suspects there exists such an extension n.

5.2 Rules Synthesis

An even more challenging problem is that of design rules synthesis. The problem could be formulated as follows:

> Given a set of component types C (which implicitely defines the set of networks $\mathcal{N}$ that can be created from the composition of such components), given a desired property find a set of design rules that define a restricted family $\mathcal{F}_R$ of networks such that any network $n \in \mathcal{F}_R$ accepted by these design rules satisfy the property.

This problem could use the results generated by the procedure for testing properties by design. Any counter example returned by the latter would be analysed by the former in order to generate a rule that prevents this situation to arise.

6 SUMMARY

In summary, we want to achieve diagnosis properties by design, i.e., a situation where, if the network satisfies some simple design rules, then some required diagnosis properties will be satisfied in the network. We need to determine design rules formalisms that will allow us to define large families of networks, and for which both the properties by design problem and ideally the design rules synthesis problem can be solved.

ACKNOWLEDGMENTS

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. Many thanks to P@trik Haslum for his help.

This paper is dedicated to Jean-Luc Le Ténia and La Grande Céline. We miss you.

REFERENCES

(Browne *et al.*, 1989) M. Browne, Ed. Clarke, and O. Grumberg. Reasoning about networks with many identical finite-state processes. *Information and Computation (IC)*, 81(1):13–31, 1989.

(Cordier and Dousson, 2000) M.-O. Cordier and Chr. Dousson. Alarm driven monitoring based on chronicles. In *Fourth IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SafeProcess-00)*, pages 286–291, 2000.

(Cordier *et al.*, 2007) M.-O. Cordier, Ya. Pencolé, L. Travé-Massuyès, and Th. Vidal. Self-healability = diagnosability + repairability. In *Eighteenth International Workshop on Principles of Diagnosis (DX-07)*, pages 251–258, 2007.

(Haslum, 2008) P. Haslum. A new approach to tractable planning. In *Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-08)*, pages 132–139, 2008.

(Pencolé and Subias, 2009) Ya. Pencolé and Au. Subias. A chronicle-based diagnosability approach for discrete timed-event systems: application to web-services. *Journal of Universal Computer Science*, 2009.

(Ribot *et al.*, 2007) P. Ribot, Ya. Pencol, and M. Combacau. Characterization of requirements and costs for the diagnosability of distributed discrete event systems. In *Fifth Workshop on Advanced Control and Diagnosis (ACD-07)*, 2007.

(Rintanen, 2007) J. Rintanen. Diagnosers and diagnosability of succinct transition systems. In *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 538–544, 2007.