Distributed Repair of Nondiagnosability

Anika Schumann and **Wolfgang Mayer** and **Markus Stumptner** [1]

1 Introduction

Automated fault diagnosis has significant practical impact by improving reliability and facilitating maintenance of systems [1]. Given a monitor continuously receiving observations from a dynamic event-driven system, diagnosis algorithms infer possible fault events that explain the observations. For many applications, it is not sufficient to identify what faults *could* have occurred; rather, one wishes to know what faults have *definitely* occurred. Computing the latter requires *diagnosability* of the system, that is, the guarantee that the occurrence of a fault can be detected with certainty after a finite number of subsequent observations [2].

This paper defines a distributed framework that assists in assessing and improving the diagnosability of discrete-event systems. In this context, a system is diagnosable iff the presence or absence of each unobservable fault event can always be deduced once sufficiently many subsequent observable events have occurred. Otherwise, the system must be altered, for example by adding additional sensors, to allow to discriminate between ambiguous system behaviours.

If the system is not diagnosable, additional sensors are required to distinguish the ambiguous system behaviours. Several past approaches deal with the problem of selecting sensor placements to ensure diagnosability of a system. However, the problem of computing an optimal sensor set with minimal size has a complexity exponential in the number of possible sensor placements [6]. Existing sensor placement algorithms are based on a global representation of the system, which may not be computable for large systems.

In this paper we address the diagnosability problem in a *distributed* way by identifying those system behaviours that require modification to restore diagnosability. In fact, we show how to determine those *subsystems* whose modification is guaranteed to make the *entire system* diagnosable.

2 Diagnosability of discrete event systems

As in [2], we consider a discrete-event system G composed of components $G_1, \ldots, G_n$ that are each modelled as finite state machine (FSM). Here the transitions are partitioned into *fault transitions* and other *locally unobservable* transitions, transitions representing *shared* events that occur simultaneously in all concerned components, and *observable* transitions. A fault of the system is *diagnosable* iff its (unobservable) occurrence can always be deduced after finite delay [2].

To decide diagnosability we use the twin plant approach presented in [3]. It computes for each component the *interactive diagnoser* $\widetilde{G}_i$ that gives the set of faults that can possibly have occurred for each sequence of observable and shared events. A local *twin plant* $\hat{G}_i$ is obtained by synchronising two instances of the diagnoser based on the observable events. Each path represents two indistinguishable system behaviours (i.e. two behaviours that emit the same sequence of observations). The twin plant states are partitioned into diagnosable and possibly nondiagnosable states [3]. A subset of the latter are the nondiagnosable states. A fault F is diagnosable in system G iff its *global twin plant* (GTP) $Sync(\hat{G}_1, \ldots, \hat{G}_n)$ has no path with a cycle containing at least one observable event and one F-nondiagnosable state[2]. Such a path is called a *critical path*. Unfortunately, computing the GTP is prohibitively expensive for large systems.

Our algorithm avoids scalability issues by computing nondiagnosable behaviours iteratively in a distributed approach, such that the global model need not be derived in many cases. We start with a set of twin plants of individual subsystems that characterise all paths that (may possibly) admit nondiagnosable behaviour (i.e. paths with a (possibly) nondiagnosable state). By composing individual models, behaviours that are infeasible or distinguishable in a larger subsystem are eliminated incrementally until (non)diagnosability can be decided or resource limits are reached. This work is an extension to the one presented in [3]. However, the latter can only verify diagnosability in a distributed way. In contrast, our approach allows to confirm diagnosability and nondiagnosability given partial models of a system.

3 Distributed (non)diagnosability assessment

Our framework is based on the two properties below. Assume a set of twin plants $\hat{G}$ is created from a partition of a discrete event system G. Then,

1. G is diagnosable if $\hat{G}$ is free of cycles that include an observable transition and a possibly nondiagnosable state, or if there is a twin plant in $\hat{G}$ where all states are diagnosable.
2. G is nondiagnosable if each plant in $\hat{G}$ includes a path to a possibly nondiagnosable state that does not have events shared with any other plant and at least one of these paths has a cycle with a possibly nondiagnosable state and an observable transition.

The first property is derived from previous results on diagnosability [3]. The correctness of the second one follows directly from the $Sync$ operation: The synchronisation of above paths from all twin plants results in a set of paths in the GTP, each containing an observable cycle with a possibly nondiagnosable state. Since every possibly nondiagnosable state in the GTP is nondiagnosable [3], such a synchronisation must contain a critical path and thus establishes the nondiagnosability of F.

[1] University of South Australia, Adelaide, Australia. Email: {schumann,mayer,mst}@cs.unisa.edu.au. This work was partially supported by the Australian Research Council under Discovery Grant DP0560183.

[2] The result of $Sync$ is a FSM whose state space is the Cartesian product of the state spaces of the components, and whose transitions are synchronised in that any shared event always occurs simultaneously in all components that define it.

4 Algorithm

We use the above results to decide whether a system is (non)diagnosable. Starting with twin plants representing individual subsystems, our algorithm iteratively removes locally nondiagnosable paths by synchronising twin plants to form larger subsystems. In case formerly indistinguishable system behaviours become discriminable through observable events of the larger subsystem, the path is removed. Otherwise, the path remains dependent (i.e. has events shared with other subsystems), but may become independent after further synchronisation. The aggregation of subsystems continues until either condition (1) or (2) is met, or until resources are exhausted. In the latter case paths exist where it is not known if the system is indeed (non)diagnosable, and we return the locally non-diagnosable paths (a superset of the truly nondiagnosable subsystem) as an approximation.

Hence, our approach admits certain anytime characteristics. Since the diagnosability problem is NP hard, some systems may require the computation of the GTP to assess diagnosability. While we cannot avoid this intrinsic complexity, we stop with an approximate solution in case resource limits are insufficient to obtain the exact solution.

5 Inferring repair alternatives

If a system cannot be proved diagnosable, an over-approximation of possibly nondiagnosable subsystems is obtained, represented by their twin plants that contain possibly nondiagnosable paths. To ensure the overall system is diagnosable, certain transitions must be modified such that the potentially nondiagnosable paths cannot manifest in the revised model.

We identify the relevant transitions using the following labelling scheme: every twin plant transition t is labelled with the set of transition identifiers that comprises all those transitions that have been synchronised to obtain t: every component transition (except fault transitions) is assigned a unique identifier label, the identifier label is propagated to the corresponding interactive diagnoser $\widetilde{G}_i$, and, subsequently, to the corresponding transition in twin plant $\hat{G}_i$.

The FSMs in Figure 1 illustrate the labelling. Every transition t of the interactive diagnoser $\widetilde{G}_i$ is labelled with the set of transition identifiers obtained from the transitions in G_i represented by t. In the twin plant every shared transition corresponds to exactly one transition in the interactive diagnoser, and every observable transition refers to two transitions (one from the left and one from the right diagnoser). For shared transitions, the labelling is kept. For observable transitions the identifier labels are obtained from the union of the two corresponding diagnoser transition labels.

Since the algorithm described below requires the synchronisation of twin plants, the transition identifiers for every twin plant $\hat{G} = Sync(\hat{G}', \hat{G}'')$ must be determined. This label propagation is similar to the propagation described previously: every transition labelled by an event that only occurs in one of the twin plants $\hat{G}''' \in \{\hat{G}', \hat{G}''\}$ carries the same identifier as the unique corresponding transition in $\hat{G}'''$. Otherwise, the identifier for a transition in $\hat{G}$ is obtained as the union of the identifiers of the two corresponding transitions.

Through transition labels those components where behavioural modification would remove a cause of nondiagnosability can be identified. For instance, the critical paths shown in Figure 1(c) can be eliminated by changing the transition from $x5$ to $x7$, which may be accomplished by modifying either component transition $t3$ or $t5$ in Figure 1(a). A system designer might choose to do this by replacing one of the sensors emitting event $o1$ by one emitting a different event, thus changing the component's behaviour. Then the behaviours represented by the two transition sequences from $a0$ to $a4$ become distinguishable.

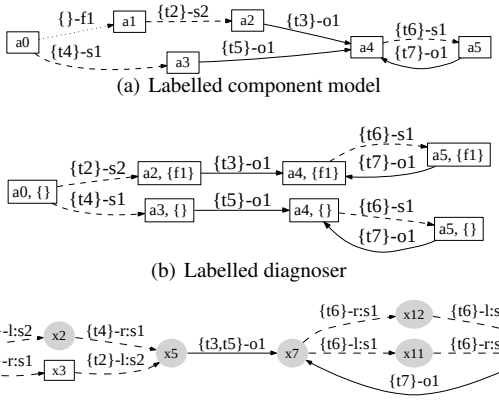

Figure 1. Assignment of transition identifiers. Solid, dashed, and dotted lines denote observable, shared, and failure transitions, respectively.

6 Conclusion and future work

We have outlined a distributed algorithm that ascertains (non)diagnosability of distributed event-driven systems. We have shown how to identify component behaviours and transitions that, if modified, render a system diagnosable.

Our approach has two distinct features: first, our algorithm can find solutions of a whole system by operating on partitions thereof, and, second, an approximation is returned if computational resources to construct the entire system are not available.

Diagnosability assessment and repair can be used to analyse physical and abstract systems such as distributed computing processes. Our work is particularly relevant for the latter, since assessing and designing monitoring capabilities of a system that are sufficient to allow compensation and reconfiguration to take place are active areas of research [4, 5].

As part of future work we intend to extend our approach to incorporate the costs for modifying the system and to explore a richer model of possible transition modifications, tailored to the analysis of distributed software systems.

REFERENCES

[1] G. Lamperti and M. Zanella, *Diagnosis of active systems*, Kluwer Academic Publishers, 2003.

[2] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, 'Diagnosability of discrete event system', *IEEE Transactions on Automatic Control*, **40**(9), 1555–1575, (1995).

[3] A. Schumann and Y. Pencolé, 'Scalable diagnosability checking of event-driven systems', in *IJCAI-07*, pp. 575–580, (2007).

[4] Rajesh Thiagarajan, Markus Stumptner, and Wolfgang Mayer, 'Semantic web service composition by consistency-based model refinement', in *The 2nd IEEE Asia-Pacific Service Computing Conference (APSCC 2007)*, pp. 336–343, Tsukuba, Japan, (December 2007).

[5] WSDIAMOND, 'WS-Diamond deliverable D5.1: Characterization of diagnosability and repairability for self-healing web services', Technical Report IST-516933, University of Torino and others, (April 2007).

[6] T. Yoo and S. Lafortune, 'On the computational complexity of some problems arising in partially-observed discrete-event systems', in *American Control Conference*, volume 1, pp. 307–312, (2001).