Distributed on-line analysis of discrete event systems: a survey of some recent results

René Boel
SYSTeMS Research Group, Universiteit Gent
Technologiepark-Zwijnaarde 914, B-9052 Gent, Belgium
rene.boel@ugent.be

Abstract

On-line supervisory control of large discrete event systems is difficult due to the exponentially growing size of the state space to be enumerated in a naive analysis. This paper surveys several attempts to reduce the computational complexity, as investigated in some recent papers by the SYSTeMS group of Ghent University. Compositionality and distributed analysis, combined with forward and backward generation of unfoldings of the set of feasible trajectories are proposed here as tools for resolving this problem.

Keywords – Petri nets, observers, fault detection, backward search, occurrence nets

1 Motivation and introduction

Recent economical and technological evolutions allow the design of plants with very high performance. The behaviour of these large man-made plants is often very complicated and subject to unexpected malfunctions, because the system typically requires the co-operation and co-ordination of many interacting components. Supervisory control of these plants requires accurate estimates of the current mode of operation of the plant as a whole. In particular a supervisory agent needs to know whether some components have failed, whether assigned tasks have been executed completely, etc. in order to take the correct decisions. For basic concepts on supervisory control, the reader can consult the book by Cassandras and Lafortune [1] or the lecture notes of Wonham [2].

This paper presents a survey of some recent work by the author and his doctoral students, addressing these problems. We assume that the plant evolves over time satisfying constraints expressed by an abstract, discrete event dynamical systems model. Different modelling paradigms may be used for different components, but in this paper all components are represented by Petri nets. In order to allow model based design of fault detectors, state estimators and supervisors, the model must include a description of those faults that must be reported to the supervisor (we assume that if a fault occurs it is permanent). The occurrence of a fault corresponds to the execution of a particular event in the Petri net model such that the behaviour of the plant following the fault is undesirable in some sense. Faults, and many other events that modify the state of the plant, are unobservable to the supervisory agent However some other events are observable, and we assume in fact that the occurrence of the observable events is always reported correctly to the local control (or fault detection) agent. In other words later observable effects

of the fault allow a supervisory agent to detect that the fault happened at least for a class of reasonable models, to be considered below.

The global plant is represented in this paper by a composition of Petri net components that interact with each other by exchanging tokens via boundary places that connect two components to each other. The main difficulty is the explosion of the size of the global state space of the plant - essentially equal to the Cartesian product of the local state spaces of each component - and the ensuing inability to use enumeration in order to achieve either off-line or on-line analysis. The purpose of the analysis, as treated in this paper, is to synthesize supervisors that achieve certain global specifications, by observing some events generated by the plant, and performing on-line calculations using these observations in order to disable certain future events that might lead to unsafe conditions..

In order to make the supervisor robust against communication failures, and in order to reduce the computational complexity of the off-line and of the on-line work, the approach proposed in this paper attempts to do as much of the calculations in a local supervisory agent, using as little communication exchange between agents as possible. The compositional model allows us to assume that each region has a local supervisor agent that knows the local Petri net model, that observes perfectly and without delay all the local observable events, and that from time to time can exchange messages with agents supervising neighbouring components. Using all the information available at time θ the local supervisor must issue local decisions so that certain global specifications on the plant behaviour are met. In order to achieve this each agents must use its knowledge available at time θ in order to estimate the set of possible values of the global state X_θ at time θ and issue decisions that are maintaining the plant in a safe state for every sequence of events that is legal in at least one of the possible current states. This paper surveys some work by the SYSTeMS group of Ghent University, mainly by G. Jiroveanu, showing that such distributed diagnosers can be synthesized efficiently [3, 4, 5] under reasonable conditions.

2 Local Observer design

This section considers one single component of a plant in isolation and shows that forward unfolding allows the design of on-line state observers. We assume that the reader is familiar with the Petri net notation $\mathcal{N} = (\mathcal{P}, \mathcal{T}, F)$ where: $\mathcal{P}$ denotes the finite set of places, $\mathcal{T}$ the finite set transitions, with marking $m(p), \forall p \in \mathcal{P}$ representing the state of the plant. Given a PN system $\langle \mathcal{N}, M_0 \rangle$ the set of all legal traces in $\langle \mathcal{N}, M_0 \rangle$ is denoted by $\mathcal{L}_\mathcal{N}(M_0)$ while the set of reachable markings is denoted by $\mathcal{R}_\mathcal{N}(M_0) = \left\{ M \mid \exists \tau \in \mathcal{L}_\mathcal{N}(M_0) \text{ s.t. } M_0 \xrightarrow{\tau} M \right\}$. We assume throughout this paper that the Petri net is such that all $m(p)$ remain bounded for all allowable traces of events. The occurrence of events t belonging to the subset $\mathcal{T}_o$ of observable events is reported immediately and without error to the local control agent. Other events in $\mathcal{T}_u = \mathcal{T} \backslash \mathcal{T}_o$ are unobservable, and are completely silent. The occurrence of unobservable events is not reported in any way to any agent.

An observer combines at time θ the knowledge about the plant model - the constraints encoded in the Petri net - with the observed (and ordered) trace of observable events that have occurred up to time θ. Thus the observer agent knows at time θ the complete model of the plant (the model of all components, and of their interaction) as well as the order of the occurrence of all the observable events that were executed up to time θ. If the set of initial states X_0 of the plant is known, then the observer calculates, at the time θ_1 of the occurrence of the first observable event $t_1 \in \mathcal{T}_o$, the set $\mathcal{R}(\theta_1, t_1)$ of all allowable traces (that is: all the traces that satisfy all the

constraints expressed by the model, and that start in an initial marking in X_0), consisting of unobservable events only, except for the last event which must be t_1. The set of markings that are generated by the traces in $\mathcal{R}(\theta_1, t_1)$ bring the plant into a state belonging to a new set X_1 of markings. The recursive operation of the observer is now obvious. Each time a new observation t_n is received at time θ_n the observer agent combines the new information expressed by the fact that an unobservable trace, satisfying all the constraints expressed by the model and starting in a state belonging to X_{n-1} has ended with te observable event t_n. This way the observer calculates a new set $\mathcal{R}(\theta_n, t_n)$ of possible traces, and a new set X_n of possible states. At each time the observer enumerates all the states that the plant can be in, given the model constraints and the past observations.

Given the sets $\mathcal{R}(\theta_n, t_n)$ one can decide whether a fault must have occurred, by checking if all traces in $\mathcal{R}(\theta_n, t_n)$ contain a fault event. A fault may have occurred if some of the traces in $\mathcal{R}(\theta_n, t_n)$ contain a fault, but one must also take into account that after time θ_n and prior to the next occurrence of an observable event, some more faults may occur. In order to check whether this is possible, the observer must also calculate all the legal traces $\overline{\mathcal{R}}(\theta_n, t_n)$. This set contains all traces satisfying all the model constraints and starting in a state belonging to X_n and executing only unobservable events.

The observer design is conceptually easy to understand provided one assumes that a set X_0 of possible initial states is known. Prior to the execution at time θ_1 of the first observed event t_1 the plant can be in any state that can be reached from any possible initial state by executing only unobservable events. At the time of the first observed event all those traces of unobservable events that lead to a state where the observed event t_1 is not enabled are discarded. This recursive updating provides a dynamic model of an observer automaton. The state space of the observer automaton consists of sets of possible states of the plant; the transitions of the observer automaton are the observable events of the plant, since the state of the observer automaton is updated each time an observable event occurs. Corresponding to the states of the observer automaton, one can also list the set of traces in the original plant that lead to these states. This observer automaton was introduced by Cieslak et al. [6]. In [7] it has been shown that the size of these observers is exponential in the size of the original Petri net model. Moreover in [7] it has been shown that faults can be detected within a finite delay (after the occurrence of at most a bounded number of events) provided that the model does not allow unobservable cycles of events to be executed (because these cycles could be executed an unbounded number of times without ever generating an observable effect that would trigger an alarm in the fault detector).

The preceding paragraph shows that the size of the state space of the observer automaton suffers from combinatorial explosion for large plants consisting of many interacting components. One reason is that many events can be executed concurrently, and the set of traces enumerates all possible orderings of these concurrent events. However this ordering is not relevant for calculating the set of possible future legal traces. Unfoldings of a Petri net are Petri nets that generate the same set $\langle \mathcal{N}, M_0 \rangle = \mathcal{L}_{\mathcal{N}}(M_0)$ of legal traces, but such that the unfolding does not contain any cycles (all the cycles in the original net have been "unfolded", which means that the unfolding can be infinite), and every place in the unfolding has only one input transition (which means that whenever a place in the original net has k input transitions, then the unfolding contains k copies of that place). One can show that there also exist finite prefixes of an unfolding, with the same initial marking as the original net, that generate the same set of traces that are possible in the original marked Petri net. If one applies this to the Petri net that contains only those transitions of $\mathcal{N} = (\mathcal{P}, \mathcal{T}, F)$ that are unobservable, and that can be executed from the known

initial marking, then one obtains an efficient representation of an observer automaton, avoiding the enumeration of all possible orderings of concurrent events [3, 5].

3 Distributed Fault Detection for a plant with two components

Analysing large plants requires that the model be decomposed into several interacting Petri net components. The extension of the above results to a compositional model is easy provided one can observe all the interactions between the components. In our model this would mean that components interact only via observable exchange of tokens: if component a sends a token to a place in component b then component b knows exactly where and when this tokens arrived. In practice this interaction is often not observable. This implies that the initial marking of a component is not known, since a component may not know whether it has received a token from a neighbouring component or not. Thus the forward unfolding method of the previous paragraph is not feasible.

This problem can be solved by using backward search for valid traces explaining a local observation. Consider the time θ when the first observable event t_1^o is reported to the observer agent of component a. This observation requires that at some time prior to θ all the input places of t_1^o contained at least one token, which in turn implies that at least one unobservable input transition of each of these places has been executed prior to θ, in turn requiring that the input places of these transitions were marked prior to θ. Continuing this backward search for traces that satisfy the Petri net model constraints, and that explain the first observed event t_1^o, one eventually reaches a place that is marked by the known initial conditions of the component, or one reaches a boundary place where one can state as a condition that some other component should have sent a token to this place prior to θ.

Unfortunately this backward search suffers from the same problem as the classical observer automaton. All concurrent event orderings will be enumerated. Moreover care must be taken whenever cycles exist in the network, and these cycles contain places and transitions that belong to different components. In [3] it is shown that one can define a backward Petri net model for each component, and that a minimal backward unfolding (a bounded net without cycles, and no places with more than one output transition) can be defined explaining the observation in an efficient way, without repeating equivalent orderings of concurrent event executions.

Whenever the backward search ends with a requirement that a token has been received from a neighbouring component, the set of possible traces may be too large. Indeed it is possible that the neighbouring agent knows that it could never have transmitted this token. Therefore the local fault detection agent may declare faults possible in cases where the global fault detection agent (that would receive all observable events of all components, and that would know the model of each component) would know that this fault cannot have occurred. The local diagnosis agent is conservative: it provides an overdiagnosis.

In order to avoid this overdiagnosis each agent may exchange messages with its neighbour. Suppose that agent a thinks a fault is possible, but knows that the occurrence of this fault is only possible if a token was sent by component b prior to time θ_n. Then agent a can send a message to agent b asking if component b could have sent this token prior to θ_n. If agent b responds that this token cannot have been sent to component a then agent a must remove all the traces that use this token from its set of possible explanations of the observed event. If agent b responds that this token transfer has indeed been possible at some time prior to θ_n, then agent a accepts the fault as possible. The response of agent b may also be that a token may have been sent by

component b provided that agent a had sent a token prior to some earlier time θ'. Both a and b may have to recalculate their set of feasible traces after a preliminary exchange of messages. This may in turn require agent b to ask if other tokens may have been sent by component a to component b, and so on. Under reasonable conditions (no unobservable cycles covering two components) one can show that this round of exchanging messages between two components ends after a finite number of messages. At the end of the communication round (assuming that the state of both component a and component b have not changed in the mean time) both fault detection agents know whether a fault may have occurred in the component they supervise if and only if a global fault detection agent would also know this. Thus the local fault diagnosis is as good as the global fault diagnosis immediately after each communication round. In between communication rounds the local fault diagnosis is a pessimistic overdiagosis that may declare some faults possible that would be excluded by a global diagnoser.

4 Extensions and open problems

Many extensions of the porblem are possible (and have only partially been solved). Finding good protocols for exchanging a finite number of messages between $k > 2$ agents controlling k components is quite difficult. Finding conditions guaranteeing that a fault will indeed be detected eventually is also very hard (since this detectability condition depends on the global model, and can not be checked locally). An important extension is the use of temporal information. Very often faults correspond to excessive delay of an event (say a valve receives an instruction "open" but gets stuck). In [3] some methods have been developed for including in the model used by the fault detector information on the normal delay between the enabling of an event and the completion of the corresponding transition . Traces may in this case become illegal because the delay in detecting some observable event is too long. This additional information allows for much more accurate fault detectors, but makes the generation of minimal unfoldings much more complicated. Finally assigning probabilities to free choices in the plant evolution may lead to practically much more useful fault detectors, but requires a global calculation in general. Finding special structures where the probabilistic calculations can be done locally would be a major improvement.

5 Bibliography

[1] C. Cassandras and S. Lafortune: Introduction to Discrete Event Systems, Kluwer Academic Publishers, 1999.

[2] W. M. Wonham: Supervisory Control of Discrete Event Systems, available on-line at: http://www.control.utoronto.ca/cgi-bin/dldes.cgi

[3] G. Jiroveanu: Fault Diagnosis for Large Petri Nets, Ph.D. thesis, Universiteit gent, 2006

[4] G. Jiroveanu and R. Boel: Distributed diagnosis for Petri net models with unobservable interactions via common places, Proceedings of CDC/ECC2005

[5] G. Jiroveanu and R. Boel: Distributed diagnosis of large interacting systems, Proceedings of the 16th International Workshop on Diagnosis (DX05), Monterey, Ca., 2005

[6] R. Cieslak, C. Desclaux, A. Fawaz and P. Varaiya: Supervisory control of discrete-event processes with partial observations, IEEE-T-AC-33, no. 3, pp.249-260, 1988

[7] M. Sampath, R. Sengupta, S. Lafortune, S. Sinnamohadeen and D. Teneketzis: Diangosability of discrete event systems, IEEE-T-AC-40, no. 9, pp.1555-1575, 1995