Diagnosing Component Interaction Errors from Abstract Event Traces

Wolfgang Mayer, Xavier Pucel, and Markus Stumptner

University of South Australia, Adelaide, SA, 5095, Australia
{mayer,xavier.pucel,mst}@cs.unisa.edu.au

Abstract. While discrete event systems have been widely applied for diagnosing distributed communicating systems, existing models may not completely satisfy the requirements for the application of fault identification and repair in software systems. This paper presents a model-based diagnosis approach that identifies possible faults based on generic fault models in abstract traces where events may be associated to multiple system components. We overcome the common limitation that precise fault models are available for each component and leverage generic fault models of classes of faults instead. We show that diagnoses representing entire classes of equivalent solutions can be computed based on local information and investigate the performance of our algorithm.

1 Introduction

The complexity and size of software systems have rapidly increased in recent years, with software engineers facing ever growing challenges in building and maintaining such systems. In particular, *testing and debugging* still constitutes a major challenge in practice, as demonstrated by the many research projects dedicated to this topic [2, 6] and the sheer number of software faults identified by commercial providers [3].

Since testing and debugging are among the most costly and time consuming tasks, a variety of intelligent debugging aids have been proposed within the last three decades. Model-based software debugging (MBSD) [9, 8] is a particular technique that exploits discrepancies between a program execution and the intended behaviour to isolate program fragments that could potentially explain an observed misbehaviour. However, most model-based techniques have been limited to single programs and have assumed complete observability of program states. The shift from single programs to distributed systems requires adaptation of debugging and diagnosis approaches.

This paper extends previous work on diagnosing failed program execution traces to distributed systems where multiple communicating software components are observed. From models of the correct interface protocol of each component potential faults that may explain the events observed in an execution are inferred. We build a system model from a set of transition systems and introduce fault transitions that reflect generic classes of faults. Every alteration of the system can be described in terms of our generic classes, which allows us to deal with totally unforeseen faults. The approach is not limited to the debugging of programs but also applies to the general class of discrete event systems.

[1] This research is partially funded by the Australian Research Council (Grant DP0881854).

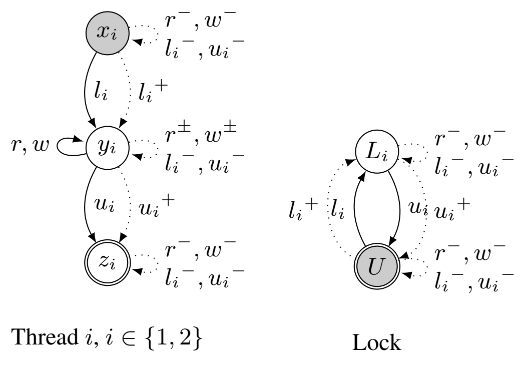
The contributions in this paper are as follows: (i) We provide a formal diagnosis model based on discrete event systems under partial observability. Different from classic assumptions our model utilizes generic fault transitions that reflect *classes* of faults rather than specific fault transitions associated with an automaton. (ii) Our model also allows for ambiguity in events in that an observed event may correspond to a number of events stemming from different automata in the model. (iii) We develop a diagnosis algorithm that computes the set of minimal explanations incrementally by minimally unfolding the global transition relation based on an observed event sequence. The algorithm computes classes of equivalent solutions rather than identify all equivalent solutions. Each solution corresponds to a repair modification in the abstract system model. This paper is organised as follows: We proceed by motivating our approach on a simple example in Section 2. The formal diagnosis model is described in Section 3. Its properties and our diagnosis algorithm is discussed in Section 4. The performance of the approach is assessed empirically in Section 5. Related work is discussed in Section 6 before summarizing the contributions in this paper in Section 7.

2 Motivating Example

The general problem context considered in this work is the diagnosis of communicating systems where only some events can be observed while others are hidden. We assume that the for each system component an automaton specifying its correct behaviour is available. Each automaton specifies the expected component behaviour in terms of event sequences. Events in this context correspond to executions of particular program instructions or messages exchanged between subsystems. Some of these events can be observed in an execution trace, for example because relevant debugging instrumentation has been furnished, while other events remain unobserved. While our model assumes all observable events that are generated appear in the trace, the information attached to each event may not be sufficient to associate an event with the emitting component.

Let us consider for example a software system composed of two parallel threads that execute the same program, which reads from and writes to some data structure protected by a shared lock. The model of the two threads and the lock are depicted in Fig. 1. The behavior specified by the automata is implemented in a program that we have instrumented in order to observe read and write accesses to the data, and lock acquisitions and releases. In this model, events l_i and u_i represent respectively the acquisition and release of the lock by thread i. They occur synchronously in all relevant automata. In contrast, events r and w represent read and write access to the data, which are not synchronized. When observing an r event, it is impossible to determine which thread actually executed it. For the sake of clarity, in this example all events are observed. As we will see later, our approach also copes with unobserved events, and in particular unobserved synchronous events.

For example, assume that the event sequence given on the right hand side in Fig. 1 has been observed. The sequence conflicts with the models of the threads and the lock, since unprotected r and w events occur between the u and the subsequent l event. Using only the models of the normal system behaviour, no precise explanation can be devised: since both threads and the lock are required to derive a conflict with the trace, all components are possible explanations.



Observed trace:

$l.r.w.u.r.w.l.r.w.u$

Minimal explanations:

$l_1.r.w.u_1^+.r.w.u_1^-.l_2.r.w.u_2$

$l_1.r.w.u_1.l_1^+.r.w.l_1^+.r.w.u_2$

Fig. 1. Model of three software components: two threads accessing data, a mutual exclusion lock, and an observed event trace. Events r and w are autonomous, and l_i and u_i are shared. Fault transitions are dotted. All non-fault events are observable. Initial states are shaded, final states are double-stroked.

We complement the nominal system behaviour with generic fault transitions that reflect the classes of faults that manifest as extraneous events (e^-) or as missing events (e^+) in the observed trace. The fault transitions are shown as dotted arcs in Fig. 1, and labels reflect which events are extraneous or missing. These fault models are created automatically from the normal system models as described in Section 3.

The fault models allow us to identify the minimal alterations that must be made to the trace to resolve the conflict with the system models. In this example there are two minimal explanations (as shown on the right hand side in Fig. 1): the unlock operation before the conflicting data access could be removed (denoted by fault event ${u_1}^-$) and a corresponding lock acquisition operation could be inserted after the access (fault event ${l_2}^+$); or additional lock and unlock operations (${l_1}^+$ and ${u_1}^+$) could be inserted before and after the unprotected data access. From these diagnoses manual investigation or further debugging techniques on the source code level could be applied in order to correct the problem.

Our diagnosis procedure can identify entire sequences of events that must be inserted and removed in order to explain an observed discrepancy. In addition to simple symmetries that may arise from ambiguity in the association of events to system components, further equivalent solutions where fault events are possible at one of multiple points in an observed trace are omitted. As a result, the number of explanations focuses on different resolutions to the problem rather than enumerate all possible fault assumptions.

3 Diagnosis Model

Our diagnosis model is based on the principle of consistency-based diagnosis [11], where a model of the nominal behavior of a system is contrasted with the actual behavior exhibited by a system. Discrepancies between the observed behavior and the behavior predicted by the model can be exploited to infer possible behavioral changes ("diagnoses") in the model that may explain the differences. The diagnosis model presented in this paper adopts the principles of Discrete Event Systems (DES) [5] to describe the nominal behavior of a system.

Definition 1 (Labeled Transition System, LTS). *A labeled transition system is a tuple (S, s_0, F, E, T) where S is a set of states, s_0 is the initial state, $F \subseteq S$ is the set of distinguished final states, E is a finite set of transition labels ("events"), and $T \subseteq S \times E \times S$ is the set of labeled transitions. We write $s \xrightarrow{a}_T s'$ for $(s, a, s') \in T$. We omit the subscript T if it is clear from the context.*

We specify the nominal behavior of a system using a set of LTSs. Each LTS corresponds to a system component and governs the sequences of events that are considered correct. Such discrete event models are particularly well-suited to specifying the expected behavior and possible interactions between components without the need to consider any particular implementation of components. We will use the terms *component* and *LTS* interchangeably if there is no ambiguity.

Definition 2 (System Model). *A system model $\left(Cs, E, E^S, E^O\right)$ (describing the system's nominal behavior) is based on a set of labeled transition systems (LTS) $Cs = \left\{C^1, \ldots, C^n\right\}$, where each C^i is an LTS $\left(S^i, s_0^i, F^i, E^i, T^i\right)$. The transition relation T^i in each LTS specifies the possible event sequences that can be exhibited by an individual component C^i if C^i is correct. Let $E = \bigcup E^i$ denote the set of events present in the system model. E can be partitioned into* shared events *$E^S \subseteq E$ and* autonomous events *$(E \setminus E^S)$. Some events are* observable *($E^O \subseteq E$).*

The behavior of the entire system is obtained from the system model by linking the individual LTSs based on the shared events in E^S. The evolution of the system is constrained such that transitions labeled with a shared event e must occur simultaneously in all components where e appears

Definition 3 (Synchronous Product). *Let $A_i = \left(S^i, s_0^i, F^i, E^i, T^i\right)$, $i \in \{1, 2\}$ be two LTS with shared events E^S. Let $\sim \subseteq E^1 \times E^2$ be an equivalence relation between events. The synchronous product transition system $A_1 ||_{E^S}^{\sim} A_2$ is defined as the LTS (S, s_0, F, E, T) where $S = S^1 \times S^2$, $s_0 = \left(s_0^1, s_0^2\right)$, $E = E^1 \cup E^2$, $F = F^1 \times F^2$, $\left(s^1, s^2\right) \xrightarrow{e}_T \left(s'^1, s'^2\right)$ if and only if (i): $e \in E^1 \setminus E^S, s^1 \xrightarrow{e}_{T^1} s'^1$ and $s'^2 = s^2$, (ii): $e \in E^2 \setminus E^S, s^2 \xrightarrow{e}_{T^2} s'^2$ and $s'^1 = s^1$, or (iii): $e, e' \in E^S, e \sim e', s^1 \xrightarrow{e}_{T^1} s'^1, s^2 \xrightarrow{e'}_{T^2} s'^2$. The synchronous product operation is commutative and associative. For brevity we will assume that $\sim = \{(e, e) \mid e \in E\}$ unless noted otherwise.*

The synchronous product may contain states that are not reachable from the initial state, or states from which no final state is reachable. In an abuse of notation, we do not distinguish the synchronous product and the automaton obtained by removing these states and the transitions related to them.

The synchronous product of all LTS in a system model, called the *global model* $G = C^1 ||_{E^S}^{\sim} \ldots ||_{E^S}^{\sim} C^n$, represents all valid event sequences that a correct system may exhibit. While this model is convenient to *define* the diagnosis problem addressed in this paper, the size of G prohibits its explicit construction for all but trivial systems.

Definition 4 (Trace). *A* trace *t of events is a finite sequence $e_1.e_2.\ldots.e_n$ of events from $e_i \in E$. Let ϵ denote the empty trace. For each trace there exists an LTS $R = (V, v_0, \{v_n\}, U, W)$ where $U = \{e_1, \ldots, e_n\}$ and $W = \left\{v_{i-1} \overset{e_i}{\to} v_i \mid i \in \{1, \ldots, n\}\right\}$. For brevity we do not distinguish between the trace and its corresponding LTS. A trace is* accepted *by an LTS A if and only if a final state in $A||^{\sim}_{E\cup U}R$ is reachable from (v_0, s_0).*

A trace reflects correct behavior if it is accepted by the global model G. However, not all events in an execution can be observed. Recall that the events in G are partitioned into observable events, E^O, and unobservable events. Therefore, an observed trace t reflects a correct execution if there is a trace t' accepted by G that exhibits the same sequence of observable events. The problem of deciding whether an observed trace is indeed accepted by G is further complicated by the fact that an autonomous event e may occur in different transition systems C^i.

If an observed trace t is not accepted by G, a fault must have occurred in the execution. Faults in a system's implementation can manifest themselves as extraneous events or as missing events in a trace. In order to isolate particular faults, the system model is amended with additional transitions that reflect extraneous events in a trace and events that have been omitted from the trace. The presence of an extraneous event e in a trace can be modeled by adding a transition to the system model that consumes e but does not change the state. Similarly, the absence of an event e from the trace can be modeled by duplicating and relabeling an existing transition for e.

Definition 5 (System Fault Model). *Let $M = \left(Cs, E, E^S, E^O\right)$ with $Cs = \left(C^1, \ldots, C^n\right)$ be a system model with $C^i = \left(S^i, s_0^i, F^i, E^i, T^i\right)$ and let G represent its global model. The System Fault Model M_F with LTSs $\left(C_F^1, \ldots, C_F^n\right)$ and $C_F^i = \left(S^i, s_0^i, F^i, E_F^i, T_F^i\right)$ is obtained from M by amending E^i and T^i to include unobserved fault events e^- and e^+ for all $e \in E$. A transition labeled e^- represents the fault where event e is present in the trace but is not admitted by G, whereas a transition labeled e^+ represents the absence of an event e in the trace:*

$$E_F^i = E^i \cup \left\{e^-, e^+ \mid e \in E^O\right\},$$
$$T_F^i = T^i \cup \left\{ s^i \overset{e^-}{\to} s^i \;\middle|\; s^i \in S^i, e \in E^O \right\} \cup \left\{ s^i \overset{e^+}{\to} s'^i \;\middle|\; s^i \xrightarrow{e}_{T^i} s'^i \right\}.$$

E^S is amended correspondingly.

A global model of the system including all possible faults can be obtained by building the synchronized product from the individual C_F^i. This global fault model represents all possible evolutions of the system, including normal and faulty behavior. We will use G_F to refer to the global system model derived from a given diagnosis problem M_F.

The diagnosis problem can now be defined by fixing the system fault model and an observed execution trace:

Definition 6 (Diagnosis Problem). *A diagnosis problem is a tuple (M_F, r) where $M_F = \left(Cs, E, E^S, E^O\right)$ is a system fault model and r is an event trace over E^O.*

A diagnosis Δ for a diagnosis problem is an event trace that is accepted by G_F.

Definition 7 (Diagnosis). *Let $P = (M_F, r)$ be a diagnosis problem with associated global model G_F, and let E^O be the set of observable events in G_F. Relation $\sim_F$ extends $\sim$ to match non-fault events in r with fault events in G_F: $\sim_F = \sim \cup \{(e, e'), (e', e) \mid e \in E^O, e' \in \{e^+, e^-\}\}$. A trace δ is a* diagnosis *for P if and only if δ is accepted by $G_F||^{\sim_F}_{E^S} r$.*

Let E^F be the set of all fault events in M_F, and let $\preccurlyeq \subseteq E \times E$ be a partial ordering of events where all non-fault events are preferred to fault events: $e \in E \setminus E^F, e' \in E^F$, $\epsilon \preccurlyeq e \preccurlyeq e'$. Relation $\preccurlyeq$ can be extended to an order on event traces such that $\sigma^1 \preccurlyeq \sigma^2$ implies $\sigma^1\sigma \preccurlyeq \sigma^2\sigma$ and $\sigma\sigma^1 \preccurlyeq \sigma\sigma^2$. A diagnosis δ is minimal *if there is no diagnosis $\delta' \preccurlyeq \delta$.*

Different ordering relations $\preccurlyeq$ can be used. For example, $\preccurlyeq$ can be defined to dynamically adjust event orderings based on the previous events in a trace to account for faults that appear multiple times in a trace or to model dependent or context-specific faults. In this paper, we focus on a static ordering where all non-fault events are preferred to any fault event. We do not order different fault events, but consider their frequency; traces with fewer fault events are preferred.

4 Diagnosis Computation

The global model is convenient to characterize the diagnosis problem and its solutions, but it does not allow us to efficiently compute diagnoses. We present an incremental approach to enumerating diagnoses that avoid constructing the global system model. Discrepancies between the observed trace and individual LTSs in the system model are exploited to extend the trace with fault transitions that may resolve the discrepancy. Equivalent diagnoses are identified from symmetries in the model and pruned.

The algorithm we present interleaves the computation of reachable states in the local models and the global model with the association of events to system components. A key element of our approach is that much of the global model can be ignored when computing diagnoses for a given observed trace.

For a given diagnosis problem (M_F, r) diagnoses can be constructed incrementally guided by the observed events in r. The idea is to minimally extend the set of partial explanations ("prefixes") to account for the next event in the observed trace. We prune execution paths that are inconsistent with the observed trace or redundant because they describe equivalent thread interleavings. By aligning the diagnosis construction with the actual observations, only the relevant states of G_F will be visited. Starting at the initial state of each LTS C^i, a prefix σ of a diagnosis can be computed incrementally.

We show that prefixes can be incrementally constructed based on local information without building the global model explicitly (Theorem 1), that enabled fault transitions partition possible explanations into equivalence classes (Theorem 2), and that symmetries in the problem can be reduced by restricting the synchronized product operation with a static ordering of components (Theorem 3).

Definition 8 (Diagnosis Prefix). *A event trace σ is a prefix of a diagnosis δ if there is an event sequence δ' such that $\delta = \sigma\delta'$.*

Since δ is a diagnosis for r, the sub-sequence of δ that contains only events in E^O is also a prefix of r: the prefix partitions r into the events that have already been accounted for (r_σ) and the remaining events ($r_{\delta'}$) in the trace. This observation can be exploited to limit the possible extensions of a given prefix σ.

Theorem 1. *Let $s = \left(s^1, \dots, s^n\right)$ be a state in G_F and let e be the next observed event in r. It is sufficient to consider the paths p^i in C^i from s^i to a transition labeled with $e' \sim_F e$ to compute all relevant extensions $\sigma' = p^i ||^{\sim}_{E^S} \dots ||^{\sim}_{E^S} p^n$ of σ in G_F. Only unobservable transitions and fault transitions need to be considered in each p^i.*

Proof. In order to consume e, it is necessary to find an enabled transition in G_F that is labeled with an event $e' \sim_F e$. Such a transition may be immediately available in s, or may become available after one or more transitions representing fault events are traversed. Therefore, it is sufficient to consider only the paths in C^i that originate in the i^{th} state component of s. If an observable non-fault transitions labeled e'' occurred in σ' it would be observed in r. Since $e \not\sim_F e''$ such a path is infeasible and need not be considered.

Fault assumptions that are unnecessary to enable subsequent fault transitions or the observed event transition for the same diagnosis prefix can also be omitted.

Theorem 2. *Let $\sigma = \langle s_1 \dots s_j t_j s_{j+1} \dots s_m \rangle$ and $\sigma' = \langle s_1 \dots s_j t_{j+1} s'_{j+2} \dots s'_m \rangle$ be two paths in G_F that differ only in the omission of transition t_j. Path σ and σ' yield equivalent diagnoses if $\langle s'_m, t_j, s_m \rangle$ is in G_F. Furthermore, if $s_m = s'_m$, then $\sigma \preccurlyeq \sigma'$.*

Proof. Paths σ and σ' differ only in the transition at position j and possibly the subsequent states. Since applying t_j in s'_m yields the same state as σ, i.e., s_m, the transitions $t_k, k > j$, are independent of t_j and hence the transitions commute. This independence induces equivalence classes of minimal diagnoses where each class includes diagnoses that differ only in the trace position where t_j is assumed. The second part of the theorem follows from $\epsilon \preccurlyeq t_j$.

Paths are further pruned by utilizing symmetries in the system model that arise when multiple instances of the same component class coexist in the system.

Theorem 3. *Let C^i and C^j, $C^i = C^j$, be two identical copies of the same LTS in the system model, and let δ be a diagnosis that contains transitions $t^i_{\{1 \dots m_i\}}$ and $t^j_{\{1 \dots m_j\}}$ associated with C^i and C^j, respectively. Then δ' obtained from δ by replacing $t^i_\cdot$ and $t^j_\cdot$ with the corresponding $t^j_\cdot$ and $t^i_\cdot$ is a diagnosis.*

Proof. Since $C^i = C^j$, any transition t^i enabled in a state s^i in C^i has a corresponding enabled transition t^j in a state s^j in C^j. This property transfers from individual LTSs C^i to G_F, as the i^{th} and j^{th} component of the global states cannot be modified by transitions not in C^i and C^j. It follows that a canonical representation of equivalent diagnoses (and their prefixes) can be obtained by imposing a fixed ordering on the LTS in the system model.

From the properties of $\preccurlyeq$ it follows that the minimal diagnosis prefixes can be enumerated using a best-first strategy. An incremental approach to constructing an LTS that accepts the minimal diagnoses is appropriate, since in most diagnosis scenarios a set of leading diagnoses is typically preferred to computing all possible diagnoses.

Our diagnosis algorithm starts in the global state $s_0 = \left(s_0^1, \ldots, s_0^n\right)$ and unfolds G_F only as much as is necessary to account for the next event in the observed trace r. Transitions and states are generated in best first order with respect to $\preccurlyeq$. Theorems 1– 3 are applied in order to avoid generating non-minimal and equivalent solutions.

The relevant reachable part of G_F is identified starting by applying the synchronous product operation transition by transition, starting in s_0. Only paths that end with a transition labeled with an event equivalent to the observed event e and transitions representing faults are considered. The expansion of a prefix is suspended if a successful extension path has been found. Each successful synchronized path constitutes a new extension σ' of σ that yields a state s' and a remaining observed trace suffix $r_{\delta''}$ where the initial e has been removed: $r_\delta = e.r_{\delta''}$. Once a prefix is extended, its ancestors on the path from s_0 to the expanded state must also be expanded further to maintain the frontier of minimal prefixes.

The resulting prefixes are organized in a LTS graph where event sequences leading to identical global states end in the same vertex. Hence prefixes that result in the same global state are *not* expanded multiple times. Each vertex is associated with the global state s in G_F, the remaining observed events $r_{\delta'}$ and the fault transitions in the best prefix leading to that state.

Once all events in r have been consumed a diagnosis has been found. Since the diagnosis prefixes and reachable states in G_F are expanded in best-first order, the first diagnosis is indeed a minimal diagnosis. To compute further diagnoses, earlier suspended prefix extension operations must be resumed in best-first order.

5 Evaluation

We conducted an empirical evaluation of the algorithm on a generalized version of the example given in the Fig. 1. The automaton for the threads was modified such that the lock can be re-acquired after release. Furthermore, the number of automata, number of faults in the trace, and the length of the trace was varied to measure the algorithm's performance (in terms of CPU time) and the number of minimal diagnoses. We tested different numbers of locks, critical sections and inconsistent data accesses. The results obtained from our implementation in Prolog are shown in the table below. The columns show (from left to right) the number of automata, faults, trace length, the number of minimal diagnoses and CPU time in seconds. For simplicity our implementation relies on an iterated depth-first strategy rather than pure best first search. The results were obtained on SWI Prolog 5.7.11 on an Intel Core2 CPU @1.8Ghz running Linux 2.6.30.

The results show that increasing the number of automata (left table) and trace length (top rows in the right table) has little impact on the algorithm. The number of explanations remains small and the result is available within a fraction of a second. Increasing the number of faults (bottom rows in the right table) however dramatically increases the computation time and number of diagnoses. This can be explained by the pathological example, where each individual fault can be explained by a number of non-equivalent faults. The results confirm that the approach is suitable for the diagnosis of typical event traces stemming from the execution of distributed loosely coupled systems, such as web services.

Automata	Faults	Length	Diags	Time (s)
4	2	20	4	0.05
12	2	20	4	0.12
52	2	20	4	0.42
5	2	20	6	0.08
7	2	28	8	0.17
9	2	36	10	0.27
11	2	44	12	0.41

Automata	Faults	Length	Diags	Time (s)
3	2	20	4	0.04
3	2	40	4	0.09
3	2	60	4	0.13
4	3	16	32	0.21
5	4	22	151	0.89
6	5	28	732	4.87

Fig. 2. Results for the ”Threads and Locks” benchmark problem.

6 Related Work

The work by Soldani [12] is most closely aligned with ours in that the framework also relies on automata specifications and models of fault classes to identify whether a system operates normally or if a fault has occurred. Our approach generalizes this work from a single fault event to multiple missing or extraneous events. Furthermore, our approach provides more detailed fault explanations and constructs the minimal diagnoses incrementally instead of relying on a diagnoser automaton built off-line.

Yilmaz and Williams [14] employ automata models to identify possible errors in software component implementations. A parametric finite state machine model is mutated to reflect possible faults, which are subsequently confirmed or refuted by comparing the mutated model to an execution trace obtained from the implementation. Our approach employs a weaker system model but allows for multiple LTSs and accounts for limited observability in observed events.

Discrete event systems have been a common tool to monitor executions of systems [4, 13] and diagnose possible faults [10]. Different from monitoring, we aim to fully explain every observed event rather than recognise known patterns. Our approach adopts a more flexible fault model than earlier work in that no a-priori limitation on possible fault transitions is necessary. Compared to dependency-based fault isolation our approach provides more detailed explanations at the sub-component level.

Similarity-based debugging of programs aims to infer possible faults in programs from observed correct and incorrect execution traces [1]. Our work differs in that faults are explained by event sequences and not simple likelihood estimates associated with individual program elements.

Sequence mining [7] has also been proposed to infer possible event sequences that are likely to lead to an error from a set of execution traces. We in contrast deal with multiple faults in a single event trace. (The approach can be extended to multiple traces.)

7 Conclusion

We introduced a diagnosis model for discrete event systems that can infer diagnoses and possible repairs in abstract event traces. The model builds on a suite of synchronized transition systems which together determine the normal as well as possible abnormal system behaviors. Our fault models are generic and are phrased in terms of added and removed transitions in the observed trace. We showed that possible explanations can be inferred locally without building the entire system model, guided by the events observed in the trace. Our algorithm interleaves the association of events to a system component, the synchronization of component models, and the resolution of discrepancies. Only distinguished explanations are computed to focus on classes of potential repairs.

In this work leading diagnoses are identified based on the assumption that faults are independent. This assumption may be relaxed in future work in order to account for systematic faults in an implementation that occur in every execution context. Furthermore, combining the event-based technique with other probabilistic debugging approaches is an avenue for further research.

References

1. Abreu, R., Zoeteweij, P., van Gemund, A.J.C.: A new bayesian approach to multiple intermittent fault diagnosis. In: IJCAI. pp. 653–658 (2009)
2. Baah, G., Podgurski, A., Harrold, M.: The probabilistic program dependence graph and its application to fault diagnosis. IEEE TSE (2010)
3. Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Henri-Gros, C., Kamsky, A., McPeak, S., Engler, D.: A few billion lines of code later: using static analysis to find bugs in the real world. Commun. ACM 53(2), 66–75 (2010)
4. Cauvin, S., Cordier, M.O., Dousson, C., Laborie, P., Levy, F., Montmain, J., Porcheron, M., Servet, I., Trave-Massuyes, L.: Monitoring and alarm interpretation in industrial environments. AI Communications 11(3/4) (1998)
5. Chen, C.H., Yücesan, E.: Introduction to discrete event systems: Christos G. cassandras and stephane lafortune; kluwer academic publishers, dordrecht, 1999, ISBN: 0-7923-8609-4. Automatica 37(10), 1682–1685 (2001)
6. Friedrich, G., Fugini, M.G., Mussi, E., Pernici, B., Tagni, G.: Exception handling for repair in service-based processes. IEEE TSE (2010)
7. Hsu, H.Y., Jones, J.A., Orso, A.: Rapid: Identifying bug signatures to support debugging activities. In: ASE. pp. 439–442. IEEE (2008)
8. Liu, Y.: A formalization of program debugging in the situation calculus. In: Fox, D., Gomes, C.P. (eds.) AAAI. pp. 486–491. AAAI Press (2008)
9. Mayer, W., Stumptner, M., Wotawa, F.: Can AI help to improve debugging substantially? automatic debugging and the jade project. Journal of the Austrian Society for Artificial Intelligence 21(4), 18–22 (2002)
10. Pencolé, Y., Cordier, M.O.: A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. Artif. Intell. 164(1-2), 121–170 (2005)
11. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32, 57–95 (1987)
12. Soldani, S., Combacau, M., Subias, A., Thomas, J.: Intermittent fault diagnosis: a diagnoser derived from the normal behavior. In: Proc. DX'07 (2007)
13. Yan, Y., Dague, P., Pencolé, Y., Cordier, M.O.: A model-based approach for diagnosing fault in web service processes. Int. J. Web Service Res. 6(1), 87–110 (2009)
14. Yilmaz, C., Williams, C.: An automated model-based debugging approach. In: ASE. pp. 174–183. ACM Press (2007)