Diagnosis from Scenarios

Loïc Hélouët, Thomas Gazagnaire, Blaise Genest
IRISA (INRIA/ENS/CNRS), Campus de Beaulieu, 35042 Rennes Cedex, France

Keywords: scenarios, partial orders, diagnosis.
Abstract: Diagnosis of a system consists in providing explanations to a supervisor from a partial observation of the system and a model of possible executions. This paper proposes a partial order diagnosis algorithm that recovers sets of scenarios which correspond to a given observation. The main difficulty is that some actions are unobservable but may still induce some causal ordering among observed events. We first give an offline centralized diagnosis algorithm, then we discuss a distributed version.

I. Introduction

The role of diagnosis is to provide information to supervisors of a system when faults occur. The objectives are manifold: either detect that the system has reached a set of critical states that should be avoided, or try to reconstruct an execution that has led to a fault. However, information retrieval is most of the time performed from partial observations: distributed systems are now so complex that monitoring every event of an execution is not realistic. In telecommunication systems, for example, the size of logs recorded at runtime grows fast, and can rapidly exceed the storage capacity, or the computing power needed to analyze them. Furthermore, the time penalty imposed by the observation to the system also advocates for a partial observation. Hence, a choice of a subset of observable events is clearly a part of the design of a complex system.

For the first kind of diagnosis, that can be defined as *fault diagnosis*, the main question is whether for given sets of faults and observable events the system is diagnosable, i.e. the occurrence of a fault can eventually be detected after a finite number of observations [8]. Diagnosis is then performed by an observer that monitors observable actions and raises an alarm when needed.

For the second kind of diagnosis, that we will call *history diagnosis* hereafter, the question is to build a set of plausible explanations of an execution from a model of a system and an incomplete observation of the faulty execution [1]. The main idea behind this approach is to exploit causality in a system to restrict the set of explanation to the smallest possible subset of runs. Then, these potential explanations can be exhaustively checked to find the actual fault.

Within this paper, we will address history diagnosis of distributed systems. The major objective of this work is to exploit concurrency in the system, and avoid combinatorial explosion using partial order models. It is well-known that interleaved models can be of size exponentially greater than concurrent model. Hence, as long as an analysis of a system does not need to study all global states, true concurrency models seem well adapted to provide efficient solutions. In this paper, we propose to model the diagnosed system with High-level Message Sequence Charts (or HMSCs for short), a scenario formalism [4]. The observation of the system(i.e. the information stored in a log file after an execution) is provided as a partial order, and the explanation is given as a set of partial order representations of all possible executions that may have generated the observation according to the model.

The authors of [1] already address history diagnosis with partial order model (safe Petri nets). In this approach, diagnosis is an incremental construction of an unfolding of the net model. The incremental aspect of this approach is clearly well adapted for online diagnosis, but does not allow for a compact representation of explanations. When unobservable events can be iterated an unbounded number of times, this incremental approach becomes impossible (unfolding may never stop).

The algorithm detailed in this paper starts from an observation O given as a partial order, an HMSC model H of the possible behaviors of the system, and the knowledge of the type of events that have been recorded in O. We also assume that the observation mechanisms that have been implemented within the distributed system are lossless. That is, if an observed event does not appear in the observation, then we have the information that it did not occur.

We do not impose restrictions on the observation architecture: observed events occurrence may be collected in a centralized way, or separately by distributed observers. However, we will consider that for a given process, all observed events are totally ordered. Furthermore, the processes may be equipped to record the respective order between events located on different processes (this ordering can be deduced for example from messages numbering, or from a vector clock). Hence the observation O may specify some particular ordering between events that is not only induced by emissions and receptions of messages. This additional information can be used to refine the set of explanations provided by the model. Indeed, if an event e happens before an event e' in the observation, then in any possible explanation provided by the model, e must be causally related to e'.

The main result of the paper is that we can still finitely represent the set of runs of a distributed system that explains a particular observation O. The explanation produced is a generator of all executions of our model for which the projection on observed events is compatible with O. More precisely, we show that the set of explanations can be described by another HMSC. This gives the basis of a centralized diagnosis algorithm.

For the distributed algorithm, we use a property showing that a global explanation can be reconstructed from local diagnosis performed for each pair of instances. Thus, each instance computes separately the set of executions that can explain what it has observed. The only (small) information that needs to be exchanged between processes is the events that were observed so far. At the end of the execution, a last step might be needed to combine together the distributed explanations. Notice that such an algorithm may also be used to track a fault on the fly, when a behavior that is not part of the model of the system is considered as faulty.

This paper is organized as follows. Section II introduces the scenario language used, and section III introduces the formal definition of an observation. Section IV defines the main algorithms for diagnosis and gives complexity results, and shows how to retrieve explanations in a distributed framework. Section V concludes this work. Due to lack of space, proffs are omitted, but can be found in an extended version from the author's webpage.

II. Scenarios

Scenarios are a popular formalism to define use cases of distributed systems. Several languages have been proposed [4], [7], but they are all based on similar representations of distributed executions with compositions of partial orders. We use Message Sequence Charts, a scenario language standardized by ITU [4]. MSCs are defined by two levels. At the lowest level, Basic Message Sequence Charts define simple interactions among components of a system called *instances*. An instance usually represent a process, or a group of processes of a distributed system. These instances exchange messages (in asynchronous mode), and can also perform atomic actions. Formally, a bMSC can be considered as a pomset which events are labeled by action names and by the instance performing the event:

Definition 1: A *Basic Message Sequence Chart* is a tuple $B = (E, \leq, A, I, \alpha, \phi, m)$, where $E = E_S \cup E_R \cup E_A$ is a set of events that can be partitioned into a set of message emissions E_S, a set of message receptions E_R, and a set of atomic actions E_A, $\leq \subseteq E \times E$ is a partial order relation (reflexive, transitive, antisymmetric), A is an alphabet of action names, I is a set of instances, α associates an action name to each event and ϕ associates a locality to each event. $m : E_S \longrightarrow E_R$ is a one to one function that pairs message emissions and receptions. Furthermore, the order on instances is a total order denoted by $\leq_i$, that is $\forall e, f \in E, \phi(e) = \phi(f) = i \implies e \leq_i f$ or $f \leq_i e$. The causal ordering among events comes from the sequential order on processes and from messages. Hence, we have $\leq = (m \cup \bigcup_{i \in I} \leq_i)^*$, where $(.)^*$ denotes the transitive closure of a relation. We will also suppose that there is no self-overtaking among messages of the same type (weak FIFO property), i.e.: for all $e \leq_i e'$, $f' \leq_j f$ with $m(e) = f$ and $m(e') = f'$, we have that $\alpha(e) \neq \alpha(e')$.

Figure 2 shows three examples of bMSCs called $M1$, $M2$ and $M3$. In bMSC $M3$, three processes $\{P1, P2, P3\}$ exchange messages m and n. Instance are simbolized by a vertical line enclosed between a white and a black rectangle. Messages are symbolized by arrows from the emitting instance to the receiving one. Atomic actions are symbolized by a rectangle enclosing the name of the action. For a more detiled description of all MSC features, we refer interested readers to [4]. In the following, we will consider that executions of a distributed system are provided as bMSCs. Note however that an incomplete observation of a distributed system is not always a bMSC: we can for example observe a message emission but forget the reception. An observation of a system is then better defined as a *labeled partial order* i.e. a tuple $O = (E_O, \leq_O, A_O, I_O, \alpha_O, \phi_O)$ where $E_O, A_O, I_O, \alpha_O, \phi_O$ have the same meaning as for bMSCs, and $\leq_O$ is given by the total ordering on each process, plus some additional ordering on different instances (deduced for example from packet numbers in a protocol). Actually, an observation is the projection of a bMSC. The *projection* of a bMSC B on a subset of its events E' is the restriction of B to E', i.e. the labeled partial order $\pi_{E'}(B) = (E', \leq \cap E'^2, A|_{E'}, I, \alpha|_{E'}, \phi|_{E'}, m|_{E'})$. Note that the projection of a bMSC is not always a bMSC, as the message mapping is not always preserved. We will often consider projection of a bMSC on a set of instances $J \subseteq I$, and denote this projection $\pi_J(B)$. More formally, $\pi_J(B) = \pi_{\phi^{-1}(J)}(B)$. We will also use the projection of a bMSC on a set of event type Σ, denoted by $\pi_\Sigma(B) = \pi_{\alpha^{-1}(\Sigma)}(B)$. For more material on scenario projections, interested readers are referred to [3].

From now on, we will consider that all bMSCs are defined on similar set of instances I, even if these instances are not active in the bMSC. We will also denote by B_ϵ the empty scenario. bMSCs alone do not have enough expressive power to describe complex behaviors. They can only define finite and very linear executions. However, the bMSC formalism has been extended with several operators to allow iterations, alternatives, and sequential composition. Sequential composition allows to glue two bMSCs along their common instance axes to build larger executions. It is formally defined as follows:

Definition 2: Let B_1, B_2 be two bMSCs. The *sequential composition* of B_1 and B_2 is denoted $B_1 \circ B_2$, and is the bMSC $B_1 \circ B_2 = (E_1 \uplus E_2, \leq_{1\circ 2}, A_1 \cup A_2, I_1 \cup I_2, \alpha_{1\circ 2}, \phi_{1\circ 2}, m_1 \uplus m_2)$, where $\leq_{1\circ 2} = (\leq_1 \cup \leq_2 \cup \{(e_1, e_2) \in E_1 \times E_2 \mid \phi(e_1) = \phi(e_2)\})^*$, with $\uplus$ denoting disjoint union.

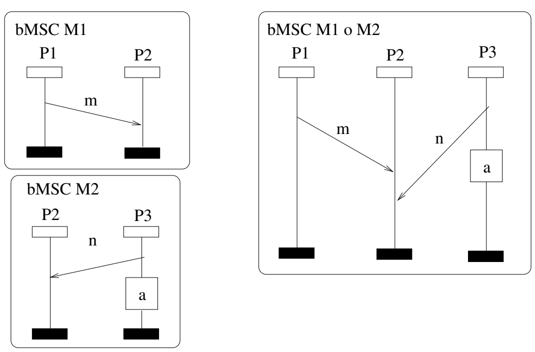

Fig. 1. Sequential composition of bMSCs

Note that sequential composition does not impose synchronization among instances: events of M_1 and M_2 can still be concurrent. Figure 1 shows an example of sequential composition of two bMSCs. In the composition $M1 \circ M2$, action a and the emission of message m, for example, are still concurrent events. The MSC formalism proposes several other operators such as alternative and iteration. These composition mechanisms are best described by a formalism called High-level Message Sequence Charts (or HMSCs for short). HMSCs are a kind of partial order automata, that should be considered as execution generators. More formally, an HMSC can be described as follows:

Definition 3: A *High-level Message Sequence Charts* (or HMSC for short) is a tuple $H = (N, \longrightarrow, \mathcal{M}, n_0, F)$, where N is a set of nodes, $\longrightarrow \subseteq N \times \mathcal{M} \times N$ is a transition relation, $\mathcal{M}$ is an alphabet of bMSCs, n_0 is an initial node, and F is a set of accepting nodes. A HMSC defines a set of successful paths $\mathcal{P}_H$ which goes from the initial node to some final node. We associate each successful path $\rho = n_0 \xrightarrow{M_1} n_1 \ldots \xrightarrow{M_k} n_k$, with a bMSC B_ρ which is the sequential composition of labels along path ρ, i.e. $B_\rho = M_1 \circ \cdots \circ M_k$.

In a HMSC, nodes define potential global states of the system, that are used to glue bMSCs. Note however that these nodes do not impose any synchronization among processes, and that a system may Figure 2 contains an example of a HMSC H. The initial node n_0 is connected to a downward triangle, and the only final node n_1 is depicted by an upward triangle. The transitions of H are $(n_0, M_1, n_0), (n_0, M_2, n_1)$ and (n_0, M_3, n_0).

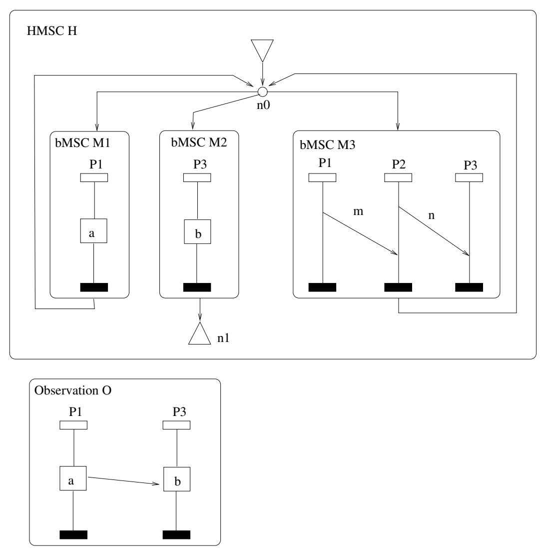

Fig. 2. A HMSC example and an observation

III. Observation

Let us now define the essential notions that will be used to find explanations of an observation. An observation O performed during an execution of a system should be an abstraction of an existing execution (i.e. an abstraction of a bMSC). We will suppose that on each instance of our distributed system, a subset of events is monitored: every time a monitored event e is executed, a message is sent by a local observer to the supervision mechanisms. In the following, we will only suppose that observations are lossless (all events that are monitored are effectively reported when they occur), and faithful (observers never send events that have not occurred to the supervising architecture, and do not create false causalities). The set of types of monitored events is Σ_{obs}. The observations can contain additional ordering information (built from local observations and additional information such as packet numbers, vectorial clocks,...), and are thus considered as labeled partial orders. We will also consider that for a given instance, the observation is a sequence, that is, the communication between local observers and the supervision architecture is FIFO. Note also that events are not observed on all instances, hence we define a set $I_{obs} \subseteq I$ on which events are monitored. Let $O = (E_O, \leq_O, A_O, I_O \alpha_O, \phi_O)$ be a partial order. We say that a set of event $E \subseteq E_O$ is a *prefix* of O if for all $a \leq_O b$ with $b \in E$, then $a \in E$. As already mentionned, an execution B is an explanation for on observation O only if they are compatible w.r.t. the sets of events observed and their causal ordering. This compatibility is defined as an embedding relation from O to B as follows:

Definition 4: Let $O = (E_O, \leq_O, \Sigma_{obs}, I_{obs}, \alpha_O, \phi_O)$ be a labeled partial order. Let $B = (E_B, \leq_B, \Sigma_B, \alpha_B, \phi_B, m_B)$ be a bMSC. We will say that O *matches* B with respect to the observation alphabet Σ_{obs} and write $O \rightarrowtail B$ whenever there exists an injective function $f : E_0 \rightarrow E_B$ such that:

- $f(E_O)$ is a prefix of $\pi_{\Sigma_{obs}}(B)$,
- $\alpha_O(e) = \alpha_B(f(e))$,
- $e \leq e' \Longrightarrow f(e) \leq_B f(e')$.

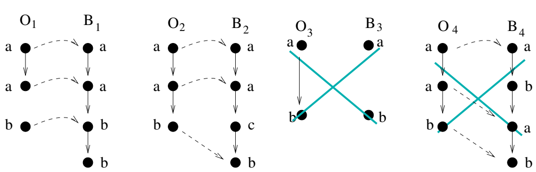

Fig. 3. Two matching examples w.r.t $\{a, b\}$ and two counter examples

More intuitively, the first requirement of this definition means that all events of an explanation have not yet been collected by the observers when the diagnosis is performed, but that when an event in an execution is observed, all its predecessors (according to the observation) have also been observed. Let us illustrate our definition on the examples of Figure **??**, where $\Sigma_{obs} = \{a, b\}$, $O_1, O_2, O_3, 0_4$ are observations, B_1, B_2, B_3, B_4 are bMSCs, and the matching relation f that sends an observation onto an execution is represented by dotted arrows. Let us consider O_1 and B_1: there is an injective mapping from the observation to a prefix of the explanation. a's b are concurrent in the observation, but the order O_1 can clearly be injected in B_1, hence $O_1 \rightarrowtail B_1$. For the pair O_2,B_2, there is also an injective mapping that maps O_2 to a prefix of the projection of B_2 onto Σ_{obs}. For the pair O_4, B_4, a and b are unordered in the explanation B_3 and hence the observation O_3 can not be injected in B_3. For the pair O_4, B_4, there is no injective mapping satisfying the three conditions. Indeed, the unmatched occurrence of b should have been observed. Hence, B_4 is not an explanation of O_4.

This matching definition is close to the definition of matching proposed by [6], [5]. It is easy to see that the function f is unique if O matches B: $f : E_O \rightarrow E_B$ is the function that sends the k-th event of E_O on instance i onto the k-th event of $\pi_{\Sigma_{obs}}(B)$ on instance i for all k and $i \in I_{obs}$. Notice that O needs not contain event of every type in Σ_{obs}, nor an event on every instance of I_{obs}. However, the fact that there are no event of some type in O rules out some possible explanations. Furthermore, an observable event located on some instance of I_{obs} in B cannot precede any event of $f(E_O)$, as otherwise $f(E_O)$ would not be a prefix of $\pi_{\Sigma_{obs}}(B)$. These properties can be used to extract explanations of an observation out of a model of the system.

Definition 5: Let O be a partial order and H be an HMSC. The set of explanations provided by H for an observation O is the set of paths $\mathcal{P} \subseteq \mathcal{P}_H$ such that $\forall \rho \in \mathcal{P}$, O matches B_ρ with respect to the alphabet Σ_{obs}.

Notice that the set of explanations provided by H is not always finite nor its linearization language is regular, but we will prove that it can be described by an HMSC in Theorem 1. As already mentioned, observations may be collected either in a centralized or a distributed way, and observed events can be sent to supervising mechanisms via asynchronous communications. Hence, the model of our system can describe runs which are longer than the observations collected so far. Note however that thanks to the prefix condition, our framework does not impose observations to be complete.

IV. Diagnosis

The main objective of our diagnosis approach is to extract from an HMSC H a generator for the set of explanations $\mathcal{P}_{O,H}$ of an observation O. This generator can be defined as a quotient HMSC of H. This quotient is computed as a product between the HMSC and the observation, with synchronization on monitored events. Hence, we will build a new automaton whose nodes are product of a node of the original HMSC with the subset of events of O observed so far, that will be called the *progress* of the observation. For instance, a path leading to the product state (v, E_O) should generate an execution that embeds O.

The main difficulty is to know the influence of unobservable events in a run of an HMSC on the respective order of observable events. As already mentioned, valid explanations may contain an infinite number of unobserved events. However, we can always keep an abstract and bounded representation of these unbounded orders. This will be modeled by a partial function $g : I \longrightarrow 2^O$ that associates to each instance $i \in I$ the observed events of O preceding the last event (observed or not) on instance i in the HMSC. Notice that this function is not redundant with the order of O since the observation and the run of the HMSC can define different orders on observed events. Let us build the following HMSC associated to an observation O and a HMSC H on an alphabet Σ_{obs}: $\mathcal{A}_{O,H} = (Q, \delta, \mathcal{M}, q_0, F')$, where δ is a new transition relation, $Q \subseteq N \times \text{Prefix}(O) \times \mathcal{F}$, and $\mathcal{F}$ is the set of functions from I to 2^O.

- $q_0 = (n_0, B_\epsilon, g_\emptyset)$,
- $F' = \{(n, E_O, g) \mid n \in F\}$,
- $\Big((n, E, g), M, (n', E', g')\Big) \in \delta$ with $E \neq O$ iff
  - $n \xrightarrow{M} n'$,
  - $E' = E \uplus \pi_{\Sigma_{Obs}}(M)$ is a prefix of O,
  - $g'(p) = g(p) \cup \{g(\phi(e)) \mid e <_M e', \phi(e') = p\} \cup \{e \in \pi_{\Sigma_{Obs}}(M) \mid e <_M e', \phi(e') = p\}$,
  - For all $a, b \in E'$ with $a <_0 b$, either $a, b \in E$, or $a <_M b$, or $\exists c <_M b$ with $a \in g(\phi(c))$.
- $\Big((n, E_O, g), M, (n', E_O, g)\Big) \in \delta$ iff $n \xrightarrow{M} n'$.

Note that $g(p)$ is updated only when the observation is incomplete. It is updated to memorize the observable events in the causal past of the last event (observed or not) executed by each instance. Similarly, we make sure during construction of a transition $\Big((n, E, g), M, (n', E', g')\Big) \in \delta$ that any order $a <_O b$ is preserved in E': either a, b precede all events of M and their ordering was already checked, or they are ordered in M, or a is in M and b precedes an event of M that happens before a. We denote by $\mathcal{P}_{O,H} \subseteq \mathcal{P}_H$ the set of paths of H that are projections on the first component of successful paths of $\mathcal{A}_{O,H}$.

Theorem 1: Let $\mathcal{A}_{O,H}$ be the HMSC computed from O and H, and $\rho \in \mathcal{P}_H$. Then $O \rightarrowtail B_\rho$ iff $\rho \in \mathcal{P}_{O,H}$. Moreover, $\mathcal{A}_{O,H}$ is of size $O(|H| \times |O|^{|I| \times |I_{Obs}|})$.

Intuitively, $\mathcal{A}_{O,H}$ is the generator of all explanations of observation O provided by the HMSC model H. The restriction of $\mathcal{A}_{O,H}$ to coaccessible states of F' is the *diagnosis* provided for observation O from the HMSC Model H. It is obvious from the construction of δ that any accepting path ρ of $\mathcal{A}_{O,H}$ generates a bMSC B_ρ such that $O \rightarrowtail B_\rho$. Note however that these path are not the minimal path embedding O. To consider only minimal path, one should consider only the relation $\delta' = \delta \cap \{\big((n, E, g), M, (n', E', M')\big) \mid E \neq E_O\}$, and the set of accepting nodes $F' = \{(n, E_O, g)\}$. Consider the HMSC H and the observation O of Figure 2. The HMSC describes the behavior of three processes $P1, P2, P3$. Let us suppose that we have equipped a distributed system to observe any occurrence of actions a and b and that we obtain the observation O. Clearly, $n_0 \xrightarrow{M1} n_0 \xrightarrow{M2} n_1$ is not an explanation of O for the observation alphabet $\Sigma_{obs} = \{a, b\}$, as a and b are not causally related in $M1 \circ M2$. The automaton $\mathcal{A}_{O,H}$ computed from O and H with this observation alphabet is given in Figure 4. The transitions with a dark cross symbolize transitions of the original HMSC that cannot be fired in the diagnosis automaton. For example, from the initial state, the transition labeled by $M2$ cannot be used, as any path p starting with this transition would not allow a matching from O to B_p. One can easily verify that O matches any bMSC composition of the form $M3^* \circ M1 \circ M3 \circ M3 \circ M3^* \circ M2$. Note that if we choose as observation alphabet $\Sigma_{obs} = \{a, b, !m\}$, the observation O has no explanation in H.

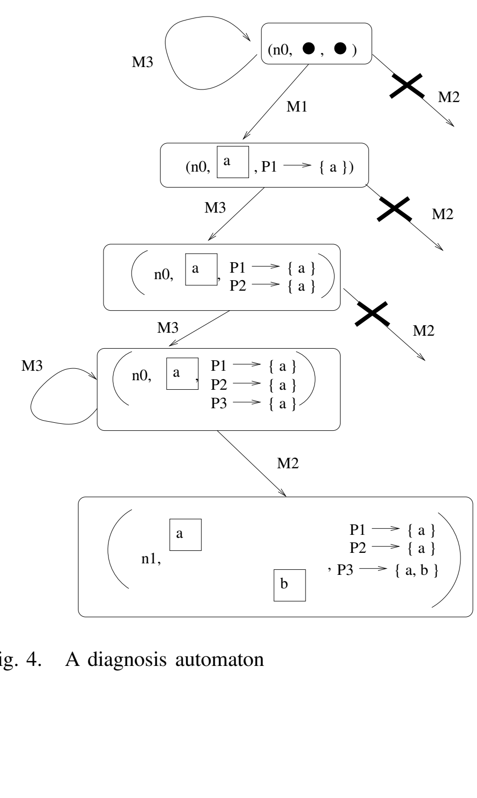

Fig. 4. A diagnosis automaton

Theorem 2: Knowing whether H contains an explanation for an observation O is NP-complete. If the set of processes is fixed, the problem is in NLOGSPACE.

Centralized diagnosis amounts to building a diagnosis automaton of exponential size in the number of processes. Furthermore, in some cases, this state space must be entirely explored to discover that no explanation exists (see theorem 2). Performing local diagnosis is a solution to reduce this complexity: each instance computes locally a partial diagnosis, that is then refined by the calculi of other instances.

Let $i, j \in I$ be a pair of instances and $O = (O, \leq_O, A_O, I_O, \phi_O, \alpha_0)$ be an observation. The local diagnosis for instances i, j is the automaton $\mathcal{A}_{i,j} = \mathcal{A}_{\pi_{i,j}(O),H}$ with the observation alphabet $\Sigma_{obs} \cap \phi^{-1}(\{i, j\})$. Since an explanation of an observation for some alphabet Σ is still an explanation for any alphabet $\Sigma' \subseteq \Sigma$, we have that $\mathcal{P}_{O,H} \subseteq \mathcal{P}_{\mathcal{A}_{i,j}}$. Hence, a finer diagnosis can be obtained from successive compositions of local diagnosis. This composition $\otimes$ is simply an intersection, defined as a synchronous product of two diagnosis automata. That is, for two HMSCs $\mathcal{A}$ and $\mathcal{A}'$, $\big((v, w), M, (v', w')\big)$ is a transition of the product $\mathcal{A} \otimes \mathcal{A}'$ iff $(v, M, v') \in \delta$ and $(w, M, w') \in \delta'$. The next proposition shows that when a run belongs to every $\mathcal{A}_{i,j}$ then it is an explanation of O:

Proposition 1: For every HMSC H and observation O, we have $\mathcal{A}_{O,H} = \bigotimes_{i \neq j \in I_{Obs}} \mathcal{A}_{i,j}$.

We know that the size of $\mathcal{A}_{i,j}$ is in $O(|O|^{2|I|}.|H|)$. An immediate idea stemming from this proposition is to split diagnosis $\mathcal{A}_{O,H}$ in several problems $\{\mathcal{A}_{i,j}\}_{i \neq j \in I_{Obs}}$ of size 2, and then compute the product of these local diagnosis. The objective is to produce a faster result when the final diagnosis is small or empty, and to avoid considering lots of intermediate states that will not lead to a final state. A solution is to build an automaton for each couple of instances and to prune them on the fly to keep only successful runs (i.e. runs that embed the observation). If any of the local diagnosis becomes empty during the computation, then we know that no explanation exists in the HMSC model for this observation. Otherwise, once the local computations have been completed, we need to compute the product of the local diagnosis to obtain the final set of runs (that is likely to be small).

Another solution is to consider first the paths that provide a matching for $\pi_i(O)$ given by the automaton $\mathcal{A}_i = \mathcal{A}_{\pi_i(O),H}$ for some $i \in I_{obs}$. Pruning this automaton may be less effective than in previous solution, because ordering cannot be used to discriminate some paths, but this initial step is performed with an initial complexity of $O(|O| \times |H|)$. Notice that $\mathcal{A}_{i,j}$ has to be computed only for those $i \neq j \in I_{obs}$ that have additional causalities implied by the observation (which can be determined online). If the only ordering between events located on i and j are derived from messages in O, then $\mathcal{A}_{i,j} = \mathcal{A}_i \otimes \mathcal{A}_j$.

An additional possibility to exploit property 1 is to perform an online distributed diagnosis, following the work of [2] for example. The strategy is then to distribute the computation of $\mathcal{A}_{i,j}$ for $i \neq j \in I_{obs}$. Noticing that $\mathcal{P}_{i,j} = \mathcal{P}_{j,i}$, we have to distribute $I_{obs}.(|I_{obs}| - 1)/2$ local diagnosis, that is each instance observing some monitored events can compute $(|I_{obs}| - 1)/2$ diagnosis HMSCs on the fly, based on the execution observed so far that is broadcasted by every instance.

V. Conclusion

This paper has proposed a scenario based diagnosis. The main objective of the approach is to perform all calculi on partial order models, and avoid the state space explosion due to an interleaved search in the execution model. We have shown that the scenario-based diagnosis can be easily distributed.

The next step is to check how this approach can be performed online. Another extension of this work would be to consider diagnosis from more powerful scenario models. Indeed, MSCs do not allow for the design of behaviors such as sliding windows. This can be considered as a limitation, as these behaviors are quite frequent in actual protocols. However, extending the scenario model inconsiderately could rapidly make diagnosis an undecidable problem (diagnosis is not decidable for communicating automata for example).

References

[1] A. Benveniste, E. Fabre, C. Jard, and S. Haar. Diagnosis of asynchronous discrete event systems, a net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5):714–727, May 2003.

[2] E. Fromentin, C. Jard, G.V. Jourdand, and M. Raynal. On-the-fly analysis of distributed computations. *Information Processing Letters*, 54:267–274, 1995.

[3] B. Genest, L. Hélouët, and A. Muscholl. High-level message sequence charts and projections. In *Proceedings of CONCUR'2003*, 2003.

[4] ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, September 1999.

[5] A. Muscholl. Matching specifications for Message Sequence Charts. In *FoSSaCS'99*, LNCS 1578, pages 273–287, 1999.

[6] A. Muscholl, D. Peled, and Z. Su. Deciding properties for message sequence charts. In *FOSSACS'98*, pages 226–242. Springer-Verlag, 1998.

[7] OMG. Uml superstructure specification, v2.0. OMG Document number formal/05-07-04, 2005.

[8] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C Teneketzis. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, 4(2):105–124, 1996.