Decomposing Minimal Observability for Transactional Services *

Debmalya Biswas and Blaise Genest,
IRISA/INRIA&CNS, Campus de Beaulieu, 35042 Rennes Cedex, France

Abstract

For complex services, logging is an integral part of many middleware aspects, especially, transactions and monitoring. In the event of a failure, the log allows us to deduce the cause of failure (diagnosis), recover by compensating the logged actions (atomicity), etc. However, for heterogeneous services, logging all the actions is often impracticable due to privacy/security constraints. Also, logging is expensive in terms of both time and space. Thus, we are interested in determining the absolute minimal number of actions that needs to be logged, to know with certainty the actual sequence of executed actions from any given partial log. This problem happens to be NP-Complete. We propose a decomposition framework in order to use a divide and conquer algorithm. This method dramatically decreases the complexity for hierarchical services (up to 2 exponentials) and can also be used in distributed services.

1 Introduction

An interesting problem for complex systems is to determine a minimal set of actions that needs to be observable such that a given property holds. Some of the properties studied in literature of discrete event systems are normality [11], observability [10], state observability [14], diagnosability [18], etc. Our system corresponds to a (composite) Web service. A Web service [1] refers to an online service accessible via Internet standard protocols. A composite service, composed of already existing (component) services, combines the capabilities of its components to provide a new service. A service schema which specifies the execution order of its components, can be modeled as a Finite State Machine (FSM), performing actions on global variables. We do not tackle here the transformation of a service into a FSM, which should be handled with care to yield a FSM of reasonable size (see [19] and example 1).

Our long-term objective is to provide a transactional framework for (composite) Web services. A transaction can be considered as a group of actions encapsulated by the operations Begin and Commit/Abort, having the following properties: Atomicity (A), Consistency (C), Isolation (I) and Durability (D). Here, we focus on the atomicity aspect, that is, either all the actions of a transaction are executed or none. In the event of a failure, atomicity is preserved by compensation [4]. Compensation consists of executing the compensating actions, corresponding to each executed action of the failed process, in reverse order of the original execution. Thus, for compensation to be feasible, we need to reconstruct each executed action or the complete history of any execution. To achieve that, we maintain a log of observable actions. In addition to the obvious space overhead of logging (in our testing, about 4 times smaller), the complete log may not always be accessible. For a composite service, the providers of its component services are different. As such, their privacy/security constraints may prevent them from exposing (part of) the logs corresponding to the execution at their sites. Also, heterogeneity may lead to the logs being maintained in different formats, rendering some of them incomprehensible. Hence, we want from such a partial log to know with certainty the actual sequence of executed actions, to be able to compensate it.

Section 2 introduces the required formal preliminaries including the precise problem statement. Clearly, we are interested in logging the smallest number of actions possible. However, it appears that determining the minimal number of actions to log, such that any execution of a system is compensable, is NP-Complete. This is not very surprising, since determining the minimal number of actions needed to achieve a property is usually NP-Complete, e.g., for sensor selections [20, 13]. What is more surprising is that it is NP-Complete even with strong restrictions on the graph (see section 3). Also, the problem cannot be approximated [16] in polynomial time.

In order to compute the minimal number of actions to log in large systems, we develop a decomposition framework. Intuitively, the algorithm we propose in section 4, first decomposes the FSM into smaller components. The problem is that even if the components are simple (with only one input and output), the union of the minimal sets of actions of different components is not necessarily an observable set

---

* This work is supported by la Region Bretagne (CREATE ACTIV-DOC) and ANR-06-MDCA-005 DOCFLOW.
of actions for the original FSM. One solution could be to resort to function summarization, but then only an overapproximation of the needed set of actions is obtained. Nevertheless, we show that it suffices to run the algorithm with slightly different parameters on each component, a number of times which depends only on the number of inputs and outputs of the components (section 5.2). That is, a fixed number of times for simple components (or for components with few inputs and outputs). We thus obtain a divide and conquer algorithm. We present a complexity analysis using the brute force method on each component which illustrates the benefit of our method, but we can use any other algorithm to compute the minimal observability, as [8]. Preliminary experiments (section 4.1) reveal that simple decompositions usually do not allow for better scaling for randomly generated graphs. However, our algorithm decomposes very efficiently the real-life examples we found into small simple components. Our implementations and examples are available at http://www.cran.u-strasbg.fr/genesv/dns/.

Interestingly, our algorithm can be used for distributed systems (section 5.3), and also in large graphs with a small compressed representation, as shown in section 5.1. We use the standard hierarchical system to depict this compressed representation, as is often used for words [15]. Finite State Machines [2], and even trees [12]. For words, e.g., hierarchical structures correspond to the LZ compression [15]. Here, the complexity can be up to two exponentials better using our decomposition method than without.

2 Preliminaries

Formally, we model a transactional service as a finite state machine, that is, a 4-tuple \( M = (Q, s_0, s_f, T) \), where:

- \( Q \) is the finite set of states,
- \( s_0 \) and \( s_f \) are the initial and final states, respectively,
- \( T \subseteq Q \times Q \) is the (partial) transition relation.

We describe our FSMs as graphs with a unique input and output point, each node and arc corresponds to a state and transition, but we ignore the alphabet. We assume that the service \( M \) does not have any unreachable states and that all states can reach the final state \( s_f \). For convenience, we also assume that there are no outgoing edges from \( s_f \) and no incoming edges to \( s_0 \).  

We say that an execution sequence \( \rho = \tau_1 \cdots \tau_n \in T^* \) is a path of \( M \) if there exists \( q_0, \cdots, q_n \in Q^{n+1} \) with \( \tau_i = (q_{i-1}, q_i) \) for all \( 1 \leq i \leq n \). A path is called initial if furthermore \( q_0 = s_0 \). We denote by \( P(M) \) the set of initial paths in \( M \). Finally, we denote by \( |M| \) the size of \( M \), that is, its number of states.

In general, for any execution \( \rho \), we call observation projections the the observation we have after \( \rho \) was executed (a sequence of actions, control points, data . . . ). We say that an observable projection \( \sigma \) is uncertain if there exists two paths having the same projection. The FSM \( M \) is execution sequence detectable iff none of its observable projections are uncertain.

Definition 1 For an FSM \( M \), let \( T_O \subseteq T \) be the set of observable transitions. The observation projection \( \text{Obs}_O : T^* \longrightarrow (T_O^*, Q) \) is the morphism with \( \text{Obs}_O(\epsilon) = \epsilon \) if \( \epsilon \in T \), and \( \text{Obs}_O(\rho) = \epsilon \) if \( \rho \in T \setminus T_O \), with \( \epsilon \) the empty word.

That is, \( \text{Obs}_O(\rho) \) is the subsequence of \( \rho \) obtained by eliminating from \( \rho \) every occurrence of a tuple which is not in \( T_O \). With such an observation projection \( \text{Obs}_O \), the only way of having execution sequence detectability is to have every transition observable. Indeed, as soon as there exists even one non-observable transition, the service is not execution sequence detectable. Else, let us take a path \( \rho \tau \) with the last transition \( \tau \notin T_O \). Then, \( \text{Obs}_O(\rho \tau) = \text{Obs}_O(\rho) \). A usual way to overcome such a problem is to ask for certainty only up to the last few events of the sequence [14]. However, this turnaround does not make sense in our framework since if we cannot compensate the very last action, then we cannot compensate any action at all. As such, we design a new observation mechanism, where the last control point reached before failure is monitored, even if the last action is not logged. In practice, it means that every state that is reached is monitored, and overstack the previous state in a special memory buffer.

Definition 2 Let \( M \) be an FSM, \( T_O \subseteq T \). The observation projection \( \text{Obs}_O^{\text{last}} : T^* \longrightarrow (T_O^*, Q) \) is the function \( \text{Obs}_O^{\text{last}}(\rho) = \text{Obs}_O(\rho, q) \) for all \( \rho \in P(M) \) ending in \( q \).

We will stick with this definition of observability for the rest of the paper. As mentioned before, we are interested in logging as few transitions as possible.

Problem statement. Given an FSM \( M = (Q, s_0, s_f, T) \), we call \( T_O \) an observable set of transitions if the service is execution sequence detectable with \( \text{Obs}_O^{\text{last}} \). We want to determine a minimal observable set of transitions \( T_O \subseteq T \).

The cardinality of such a minimal observable set \( T_O \) of an FSM \( M \) is referred to as its observable size \( MO(M) = |T_O| \). Notice that as is usual with decision and computation algorithms, it is sufficient to have an algorithm which from an FSM gives its observable size. That is, we can derive a minimal observable set of the FSM based on knowledge of its observable size in polynomial time.
3 A Difficult Problem

We first relate the problem of computing $MO(M)$ using our definition of observable projections with other known problems. We state now that computing the minimal observable set is equivalent to the unconnected subgraph problem, also called the minimal marker placement problem [13], in the meaning of the following proposition.

**Proposition 1** Let $M$ be an FSM and $T_O$ a subset of transitions of $M$. Denote by $M'$ the FSM $M$ obtained by deleting all transitions belonging to $T_O$. Then, $T_O$ is an observable set of $M$ if and only if there does not exist a pair of paths $p_1 \neq p_2$ of $M'$ with $p_1$ beginning and ending at the same states as $p_2$.

To prove the proposition 1, it suffices to prove that if there does not exist a pair of paths $p_1 \neq p_2$ of $M'$ with $p_1$ beginning and ending at the same states as $p_2$, then from any observable projection $(\sigma, q_{n+1})$, we can reconstruct in a unique way a path with $\text{Ob}_O^{\text{last}}(\rho) = (\sigma, q_{n+1})$. The converse is trivial. Indeed, it suffices to define the only path $p_1$ of $M'$ between $q_i'$ and $q_{i+1}'$ for $\sigma = (q_i, q_i')_{i=0}^n$, and $i = 0 \cdots n$ (we fix $q_0' = q_0$ the initial state of $M'$, and recall that $q_{n+1}'$ is the last observed state). Then, the path $\rho = \rho_0(q_1', q_1') \cdots (q_n, q_n')_{n}$ is the only path with $\pi^{\text{last}}(\rho) = (\sigma, q_{n+1})$. The search for each path $\rho_i$ can be made in linear time by a simple depth first search in $M'$.

The fact is that the marker placement problem is an NP-Complete problem. The question is then to know if there is a structural subclass of graphs which has a tractable algorithm to give the minimal observable size. We know from [13] that the minimal marker placement problem is NP-Complete even for acyclic graphs. However, the proof uses a graph with unbounded (in and out) degree. We show that the problem is NP-Complete even if the corresponding graph is acyclic and the sum of its in and outdegree bounded by 3 (that is, indegree 2 and outdegree 1, or vice versa). The core of the proof follows the same strategy as [13], but the encoding to get a unique starting and ending point is both easier to understand and allows a lower in and outdegree.

**Theorem 1** Let $M$ be an FSM, and $k$ a number. Knowing whether $MO(M) \leq k$ is NP-Complete, even if the corresponding graph is acyclic and the sum of in and outdegree of every node bounded by 3.

**Proof.** Let $M$ be a system. We reduce Vertex Cover to the problem of finding a subset of transitions $T_O$ of $M$, such that, there are no two paths $p_1 \neq p_2$ beginning and ending at the same nodes, and using no transitions of $T_O$.

Let us take an undirected graph $(V, E)$ and a number $k$. We want to know whether there is a subset $V_O$ of $V$ of size $\leq k$ such that for all $(v, w) \in E$, at least one of $v$, $w$ belongs to $V_O$. This problem is NP-complete even with
Making the graph a system is a little more involved. We use the graph $G$ from Fig. 2. It then suffices to create a balanced binary tree of transitions with root $s_j$ such that there are $E$ leaves. This tree has $O(2^{|E|})$ nodes, that we add to the system $S$ we built from $(V, E)$. The root of the tree is the unique initial node, and every leaf is connected to a node $(e_1)_{e \in E}$ through a copy of graph $G$. The same is done for nodes $(e_2)_{e \in E}$ connected through copies of $G$ to a balanced binary tree with root $s_j$ the unique final state. This system has $O(|V| + |E|)$ nodes, is acyclic and of total degree 3. Now, it is easy to show that if the minimal vertex cover has $k$ vertices, then the minimal observable set of transitions is of size $k + 4|E|$. Indeed, there are $2|E|$ copies of the graph $G$ each of which requires 2 observed transitions. Once these transitions are observed, the two balanced trees are totally disconnected from each other and from the first system we had built (since every path from the initial to the final state of the graph $G$ uses one of the two observed transitions), and hence we need to observe exactly $k$ more transitions. Notice that connecting directly the tree with $S$ without using $G$ would not work since it would potentially connect $s^0$, $s^1$ through two different paths $s^0 \rightarrow e_1 \rightarrow e_2 \rightarrow s^1$ and $s^0 \rightarrow f_1 \rightarrow f_2 \rightarrow s^1$, with $e, f \in E$.

This theorem does not mean that the problem is impossible to solve, but that it can be solved for small enough services only. For instance, the complexity of the brute force method which generates every subset of transitions and tests whether it is observable, is $O(2^{4|T|})$ for a service $M$ with $|M|$ transitions. The question is: how can we reduce the time taken to compute $MO$ for bigger services? It also implies that a structural restriction of an FSM $M$ which has a tractable algorithm to compute $MO(M)$ is at least very complicated to find.

4 Simple Decomposition of Graphs

The idea we use is that of divide and conquer: we would like to decompose the given graph into two parts, and compute $MO$ independently on each part. Since the best known algorithm takes exponential time (adding even one node to a graph can make the algorithm twice as slow), even an unfair decomposition is good enough. However, obtaining such an algorithm is non-trivial, since the fact that a subset of transitions is observable is global to the whole graph, not local.

Figure 3. FSM $M$.

$(V, E)$ of degree 3. The first FSM $M$ we build has a state space $S = V_1 \cup V_2 \cup E_1 \cup E_2$ where $V_1 = \{v_1 \mid v \in V\}$ and $E_2 = \{e_i \mid e \in E\}$. Furthermore, for $v, w \in V$ and $e \in E$, we have transitions

1. $(e_1, v_1) \in T$ iff $v \in e$ iff $(v_2, e_2) \in T$
2. $(v_1, w_2) \in T$ iff $v = w$.

A graphical representation of $M$ appears in Fig. 3.

Assume that there is a subset $V_O$ of $V$ of size $k$ such that for all $(v, w) \in E$, at least one of $v, w$ belongs to $V_O$. Then, defining $T_O = \{(v_1, v_2) \mid v \in V_O\}$, we have that there are no two paths $\rho_1 \neq \rho_2$ with $\rho_1$ and $\rho_2$ beginning and ending at the same nodes, and not using transitions of $T_O$. By contradiction, else we would have $\rho_1, \rho_2$ both from some $e_1 \in E_1$ to some $e_2 \in E_2$ and not using transitions of $T_O$. By definition of $T_O$, it means that for $\rho_1$, there exists a node $v \in e, v \in f$ such that $v \notin V_O$. Similarly, for $\rho_2$ with a node $w$. Since $\rho_1 \neq \rho_2$, we have that $v \neq w$; hence $e = (v, w)$ contradicts $V_O$ is a vertex cover.

Conversely, assume that there is a set of transitions $T_O$ of size $k$ such that there do not exist two paths from and to the same state without using $T_O$. We build the set of nodes $V_O = \{v \mid (v_1, v_2) \in T_O\} \cup \{v \mid \exists e_1, (e_1, v_1) \in T_O\} \cup \{v \mid \exists e, (v_2, e_2) \in T_O\}$. Clearly, $|V_O| = |T_O| = k$.

We prove now that $V$ is a vertex cover of $(V, E)$. Assume by contradiction that there exists an edge $e = (v, w)$ such that $v, w \notin V_O$. Then, we argue that $e_1v_1v_2e_2$ and $e_1w_1w_2e_2$ are two paths not using $T_O$, a contradiction.

However, so far, the graph defined is not a system since it has several states with indegree 0 (the $(e_1)_{e \in E}$), and several states with outdegree 0 (the $(e_2)_{e \in E}$). Moreover, the indegree of states $(v_1)_{v \in V}$ and the outdegree of states $(v_2)_{v \in V}$ can be 3 (the degree of the undirected graph $(V, E)$). However, it is acyclic. For the degree, one can safely transform any node $v_1$ with 3 ingoing transitions from states $e_3, f_1, g_1$ by having two nodes $v_1, v_1'$ with transitions $(e_3', f_1', g_1'), (f_1', v_1'), (v_1', v_1)$ and $(g_1', v_1)$. Hence all nodes have indegree at most 2. The same can be done for outdegree. The size of the minimal observable set of transitions will not change with such a transformation. Actually, with such a technique, we could start from an undirected graph of any degree.
∀q ∈ Q \ Q′, q′ ∈ Q′, we have (q, q′) ∈ T or (q′, q) ∈ T implies q′ ∈ \{s'_0, s'_1\}.

First, we need the following additional notations. Given \( T_O \), a path \( ρ \) is said to be an unobserved path if it does not use any transitions of \( T_O \). For a service \( M \) and a set of transitions \( T_O \) of \( M \), we define the following predicates:

- \( P_0(M, T_O) \) holds if there does not exist more than one unobserved path between any two states \( s_1, s_2 \in Q \) (\( T_O \) is an observable set of transitions).
- \( P_1(M, T_O) \) holds if (i) \( P_0(M, T_O) \) holds, and (ii) there does not exist an unobserved path from \( s_0 \) to \( s_f \).
- \( P_1(M, T_O) \) holds if (i) \( P_0(M, T_O) \) holds, and (ii) there do not exist states \( s_1, s_2 \in Q \) such that (a) there is an unobserved path from \( s_0 \) to \( s_2 \), (b) there is an unobserved path from \( s_1 \) to \( s_f \), and (c) there is an unobserved path from \( s_1 \) to \( s_2 \). We refer to such a combination of nodes and edges (e.g., Fig. 2) as an unobserved reverse cyclic pattern between \( s_1 \) and \( s_2 \) (within \( M \)).

By definition, \( P_1(M, T_O) \) \( \Rightarrow \) \( P_1(M, T_O) \) \( \Rightarrow \) \( P_0(M, T_O) \), since for all \( s \), there always exists a path from \( s \) to \( s \). Let \( ε < 0 < 1 < 1' \). We define \( \text{Best}(M, T_O) = x \in \{ε, 0, 1, 1'\} \) such that \( P_2(M, T_O) \) holds, but not \( P_{ε}(M, T_O) \) with \( x > x \), with the convention \( P_2(M, T_O) \) is always true. Informally, Best refers to the best properties a given set of transitions can ensure, if observed.

**Proposition 2** Let \( C \) be a simple component of \( M \), and \( T_1, T_2 \) be subsets of transitions of \( C \), respectively such that \( \text{Best}(C, T_1) = \text{Best}(C, T_2) \). Then, for all subset of transitions \( T_O \) of \( M \setminus C \), we have \( \text{Best}(M, T_O \cup T_1) = \text{Best}(M, T_O \cup T_2) \).

**Proof.** Let \( C = (Q', s'_0, s'_1, T') \), and \( \text{Best}(M, T_O \cup T_1) = x \). Let us assume that \( \text{Best}(M, T_O \cup T_2) = ε \), that is, there exists a pair of states \( s_1, s_2 \in M \) with unobserved (for \( T_O \cup T_2 \)) paths \( ρ_1, ρ_2 \) from \( s_1 \) to \( s_2 \), such that the states traversed by \( ρ_1 \) and \( ρ_2 \) are disjoint, but for \( s_1 \) and \( s_2 \). We show now that \( \text{Best}(M, T_O \cup T_1) = ε \).

If both \( ρ_1 \) and \( ρ_2 \) do not touch \( C \), then \( \text{Best}(M, T_O \cup T_1) = ε \). If both \( ρ_1 \) and \( ρ_2 \) belong to \( C \), then \( P_0(C, T_2) \) does not hold, which means \( P_0(C, T_1) \) does not hold, implying \( \text{Best}(M, T_O \cup T_1) = ε \).

If \( s_1, s_2 \in (Q \setminus Q') \cup \{s'_0, s'_1\} \), and \( ρ_1 = ρ_2 \) passes through \( C \), then there exists an unobserved (for \( T_O \)) path from \( s_0 \) to \( s_1 \) (a subpath of \( ρ_1 \)). Given this, \( P_1(C, T_1) \) does not hold; which implies that \( P_1(C, T_2) \) does not hold. Hence, there exists an unobserved (for \( T_O \)) path from \( s_0 \) to \( s_1 \), and an unobservable (for \( T_O \cup T_2 \)) path \( ρ \) can be constructed from this path and \( ρ_2 \). Such that, there are two disjoint paths unobservable for \( T_O \cup T_1 \) between \( s_1 \) and \( s_2 \):

\[ \text{Best}(M, T_O \cup T_1) = ε. \]
Let components of an FSM. We present here a linear time (in the number of transitions) algorithm to compute the smallest simple component \( C \) of an FSM \( M \) knowing its starting and ending state, and an initial transition from starting state.

We now give an algorithm to compute the largest simple component of \( M \).

1. For a pair of components \( D, E \in S_C \), if \( D \) is a subgraph of \( E \), delete \( D \).
2. For a pair of components \( D = (Q', s'_0, s'_f, T') \) and \( E = (Q'', s''_0, s''_f, T'') \) of \( S_C \), if \( s'_0 = s''_0 \) and \( s'_f = s''_f \), then create a new component \( F = (Q' \cup Q'', s'_0, s'_f, T' \cup T'') \). If \( F \neq M \), then delete \( D \) and \( E \) from \( S_C \), and add \( F \) to \( S_C \).
3. Return the biggest \( C \in S_C \).

4.1 Experimental Results

We tested our decomposition algorithm on acyclic graphs randomly generated, using the Synthetic DAG generation tool (http://www.loria.fr/~suter/dags.html), with 512M-B heap memory. Usually, the graphs randomly generated look very unnatural, and as expected, there are really few simple components in the graph. Moreover, very big components (with about 85% of edges) remains non decomposable. Our algorithm finds these components within seconds, which allows a speed up of about four times. Unfortunately, it does not change the exponential scaling. The algorithm runs out of memory at around 23 edges (instead of around 20 edges without decomposition).

Random graphs generated using the parameter "-regular \( x \)" \((x \leq 0.2)\), that is the distribution of tasks between the different levels is fairly regular, look more natural. For them, our algorithm finds non trivial components. For instance, for a system with 31 edges, the biggest undecomposable component has only 18 edges. The algorithm finds an observable set of 7 transitions in 3 seconds when decomposition is used, instead of 4 minutes. Without the decomposition method, the algorithm runs out of memory at about 21 edges.

\[\text{Theorem 2} \quad \text{Given } C \text{ a simple component of } M, \text{ MinimalDecomposition}(M, C) \text{ returns a minimal observable set of transitions of } M \text{ in time at most } O(2^{1|C|} + 2^{|M|-|C|}).\]

\[\text{Proof.} \quad \text{First, the brute force algorithm for computing } T_y(C) = T_0(P_y(C)) \text{ takes } O(2^{1|C|}) \text{ time, and takes } O(2^{|M|-|C|}) \text{ for computing } T_0(T_0(C), C)(M). \text{ Note that heuristics} [8] \text{ or Sat solvers could be used instead of the brute force method.} \]

\[\text{Finally, it remains to explain how to find simple components of an FSM. We present here a linear time (in the number of transitions) algorithm to compute the smallest simple components } C \text{ of an FSM } M \text{ knowing its starting and ending state, and an initial transition from starting state.}\]
edges, while it runs out of memory at about 33 edges with the decomposition algorithm. Using an optimized algorithm instead of brute force would allow a much better scaling in both cases.

We found several workflows in the literature, that we automatically transformed into graphs. For instance, we modeled the workflow from [5] using 12 nodes and 26 edges (see http://www.crans.org/~genest/dns/). Without the decomposition method, the brute force method goes out of memory. Our algorithm decomposes the graph within one second. Component $C_1$ is found. In component $C_1$, component $C_2$ is found and so on until component $C_{i-1}$. Our algorithm then uses the brute force method on $C_{i-1}$ and $C_1 \setminus C_{i-1}$ (we denote by $C_0$ the original graph). Since $C_{i-1}$ has 1 edge, and $C_1 \setminus C_{i-1}$ has 3 edges for $3 \leq i \leq 9$ and else 1 edge, the brute force algorithm was very efficient, finishing also within one second. We found that the minimal observable set of transitions has 15 edges. Moreover, logging every action of this workflow takes around 1 MB, while it takes 200 KB to log a minimal observable set of transitions.

5 More General Services

We overcome the limitations of the previous method in this section, namely by allowing several components, with more than one entry and exit state, and distributed systems.

5.1 Hierarchical Services

The decomposition into a simple component can be extended with the component being in turn decomposed and so on. Even better, if a component is used several times (as is the case for large composite services), it does not need to be recomputed several times. In order to formalize such a framework, we turn to hierarchical services.

Hierarchical services provide an efficient way to model large and complex services by allowing a modular decomposition of the problem space. In particular, we consider hierarchical services where two transitions (supertransitions) can be further refined into an FSM. A hierarchical FSM $H$ is a finite sequence $\langle M_i \rangle_{i=1\ldots n}$, where $M^i = (Q^i, s^i_0, s^i_f, T^i, (r^i_1, k^i_1), (r^i_2, k^i_2))$ is defined as follows:

- $Q^i$ is the finite set of states,
- $s^i_0$ and $s^i_f$ are the initial and final states, respectively,
- $T^i \subseteq Q^i \times Q^i$ are the transitions,
- $r^i_1, r^i_2 \in T^i \cup \{\epsilon\}$ are two supertransitions representing services $M^{k^i_1}$, $M^{k^i_2}$ respectively, with $k^i_1, k^i_2 > i$.

With each hierarchical FSM $H$, we associate an ordinary FSM $H^*$ obtained by taking $M^1$, and recursively substituting each supertransition by the FSM it represents. We define the size of a hierarchical FSM $H$ as the sum of the number of transitions of its components $M^i$. Its diameter is the number of transitions of $H$. The size $H$ can be logarithmic in the diameter of $H$. Applying dynamic programing with Theorem 2 as base case leads to the following result:

**Theorem 3** Let $H = \langle M_i \rangle_{i=1\ldots n}$ be a hierarchical FSM. It is NP-complete in the size of $H$ to compute $MO(H)$. Moreover, it takes at most time $O(\sum_{i=1}^{n} 2^{M^i_1})$.

**Proof.** It suffices to compute a minimal set $T_v(M_i)$ of transitions of $M_i$, for all $v \in V = \{0, 1, 1'\}$. These actions are performed from bottom of the hierarchy to top. To compute $T_v(C_i)$ for a module $M_i$ using $M_j, M_k, j, k > i$, we use $T_{v'}(M_j)$ and $T_{v'}(M_k)$ which have already been computed. Indeed, it suffices to compute a minimal set $T_{v'}(M_j) \cup T_{v'}(M_k)$ of transitions of $M$, for all valuations $v', v''$. Then, it suffices to output a set among $T_{v'}(M_j) \cup T_{v'}(M_k) \cup M_k(M)$ of minimal size. The best case comparison is with respect to a hierarchical service of diameter $O(2^n)$, using $n$ components of size 2. The brute force non-compositional method run on $H$ takes time $O(2^{2n})$, while our method takes $O(n)$, that is a doubly exponential improvement (one exponential due to the reuse of components, and another due to decomposition).

In addition to reducing the problem space, another motivation for considering hierarchical systems is to study systems having desirable properties. For instance, let us consider programs where goto statements (or interleaving branches) are not allowed. Such a hierarchical system representation $H = \langle M_i, C \rangle$ is given in Fig. 6. Then, $MO(H) = n \times MO(C)$, where $n$ is the number of times $C$ occurs in $H$. Basically, for $C = (Q^i, s^i_0, s^i_f, T^i)$, if there exists $m$ paths (with no interleaving amongst them) from $s^i_0$ to $s^i_f$, then $MO(C) = m - 1$. Given this, to compute $MO(H)$, for each occurrence of $C$ in $H = (Q, s_0, s_f, T)$, we can collapse $C$ and make a recursive call to compute $MO((Q \setminus Q^i) \cup \{s^i_0, s^i_f\})$. The structure guarantees that at the termination of such a recursive invocation, $H = C$.

![Hierarchical representation of programs with no goto statements](image)
5.2 Complex Decompositions

The previous results are quite encouraging, but not totally satisfactory as simple decompositions may not be enough to reduce the size of the components. Indeed, using components with several entry and exit states (complex decomposition) help reducing further the size of the components.

Assume that a service $M = (Q, s_{0,1}, \ldots, s_{0,b}, s_{f,1}, \ldots, s_{f,b}, T)$ has a set $P$ of $b = b_1 + b_2$ port states, consisting of $b_1$ input states $(s_{0,j})_{j=1}^{b_1}$ and $b_2$ output states $(s_{f,i})_{i=1}^{b_2}$. Let $T_0$ be a set of transitions of $S$. We define $b^2 + 1$ predicates $P_0(M, T_0)$, $P_{p_1,p_2}(M, T_0)$ for all $p_1, p_2 \in P$.

- $P_0(M, T_0)$ holds if there does not exist more than one unobserved path between any two states $s_1 \neq s_2 \in Q$ ($T_0$ is an observable set of transitions).
- if $p_1$ is an input and $p_2$ an output, $P_{p_1,p_2}(M, T_0)$ holds if (i) $P_0(M, T_0)$ holds, and (ii) there does not exist an unobserved path from $p_1$ to $p_2$.
- if $p_1$ is an output and $p_2$ an input, $P_{p_1,p_2}(M, T_0)$ holds if (i) $P_0(M, T_0)$ holds, and (ii) there do not exist states $s_1, s_2 \in Q$ such that (a) there is an unobserved path from $p_2$ to $s_2$, (b) there is an unobserved path from $s_1$ to $p_1$, and (c) there is an unobserved path from $s_1$ to $s_2$.
- if $p_1, p_2$ are two inputs, $P_{p_1,p_2}(M, T_0)$ holds if (i) $P_0(M, T_0)$ holds, and (ii) there does not exist a state $s$ in $M$ with unobserved paths from both $p_1$ and $p_2$ to $s$.
- if $p_1, p_2$ are two outputs, $P_{p_1,p_2}(M, T_0)$ holds if (i) $P_0(M, T_0)$ holds, and (ii) there does not exist a state $s$ in $M$ with unobserved paths from $s$ to both $p_1$ and $p_2$.

Notice that some predicates imply others, as $P_{p_1,p_2} = P_{p_2,p_1}$ for two inputs or outputs, and $P_{p_1,p_2} \implies P_{p_2,p_1}$ for $p_1$ output and $p_2$ input. However, we do not have a total order between predicates. That is, we define $\text{Best}(M, T_0) : P^2 \to \{0,1\}$ as a function with $\text{Best}(M, T_0)(p_1,p_2) = 1$ iff $P_{p_1,p_2}(M, T_0)$. We can then extend the proof of proposition 2 to obtain the following proposition (proof omitted because of lack of space):

**Proposition 3** Let $C$ be a component of $M$, and $T_1, T_2$ be subsets of transitions of $C$, respectively such that $\text{Best}(C, T_1) = \text{Best}(C, T_2)$. Then, for all subset of transitions $T_0$ of $M \setminus C$, we have $\text{Best}(M, T_0 \cup T_1) = \text{Best}(M, T_0 \cup T_2)$.

We can use the above theorem to compute a minimal observable set for an FSM $M$ having component $C$ as follows:

MinimalDecomposition-Complex($M, C$):

1. Compute a minimal set $T_0(C)$ of transitions of $C$, for all valuation $v : P^2 \to \{0, 1\}$.
2. Compute a minimal set $T_0(T_0(C), C)(M)$ of transitions of $M$, for all $v$.
3. Output a set of smallest size among $\{T_0(T_0(C), C)(M)$. It allows to extend the previous result on simple components, for instance for hierarchical services, where each component uses at most $p$ ports:

**Theorem 4** Let $p \geq 1$. It is NP-complete in the size of a hierarchical service $H = (M_i)_{i=1}^n$ using at most $p$ ports to compute $MO(H)$, with $p$ fixed. Moreover, it takes time at most $O(2^{npd} \sum_{i=1}^n 2^{M_i} + 1)$.

**Proof.** Let us consider a hierarchical system $(C_i)_{i=1,\ldots,n}$. The time complexity comes directly from Proposition 3 and the same dynamic programming algorithm as Theorem 3. It suffices to compute a minimal set $T_{\text{root}}(C_i)$ of transitions of $C_i$, for all valuation $v : P^2 \to \{0, 1\}$. We call $V$ the set of valuations. These actions are performed from bottom of the hierarchy to top. To compute $T_{\text{root}}(C_i)$ for a module $C_i$ using $C_j, C_k, j, k > i$, we use $T_v(C_j)$ and $T_v(C_k)$ which have already been computed. Indeed, it suffices to compute a minimal set $T_{\text{root}}(C_i) \cup T_v(C_j) \cup T_v(C_k)(M)$ of transitions of $M$, for all valuations $v, v'$.

Indeed, if we have such a proof, then we can choose an observable alphabet for $(C_i, v)$ of size $sze(n, v)$, and so on until an alphabet of $sze(1, v) = 0$. The correctness of such a proof can be checked in polynomial time. Assume that $MO(H) \leq k$. It remains to prove that there exists such a proof. Let $|T_0| \leq k$ be an observable alphabet.
of $\mathcal{H}$. Then on each component $C$ of $\mathcal{H}$ which corresponds to some $H_i$, we can compute $\text{Best}(C, T_O|C) = v$. We do the same with its subcomponent $D, E$. $\text{Best}(D, T_O|D) = v'$.

The problem is that there are many components $C$ corresponding to $H_i$, and several can have the same best properties $v$. We fill the entry with some component $C$ having the smallest $|T_O|$. The reason why we can do it is given by Proposition 3, that is we can replace any set of transitions involving no interaction between them (that is, one service cannot respond to another as well if the same component is reused). Moreover, strict execution sequence detection is irrelevant, and hence useless in the proof.

If there are some empty inputs, e.g., component $H_i$ for which we never find $\text{Best}(H_i, T_O|H_i) = (x, y, z)$, then it is irrelevant, and hence useless in the proof.

A simple rule of thumb to know whether it is worth applying our technique on a component is when it has a high number of interactions with respect to the square of the number of its ports. In order to find components having few ports, one can use heuristics on graph partitioning [9].

5.3 Distributed Services

We consider distributed services, given in the form of the product of two services $M$ and $N$. We first on services having no interaction between them (that is, one service cannot write a global variable that is read or written by another service). We explain later how to deal with interacting services having a non interacting component.

The composition schema of such a composite service is specified as a product of the FSM’s (corresponding to the composition schema) of the component composite services [3, 6]. Given services $M = (Q_1, s_0, s_f, T_1)$ and $N = (Q_2, s_0, s_f, T_2)$, $T_1 \cap T_2 = \emptyset$, we define their product $M \times N = (Q, s_0, s_f, s_1, s_2, T)$ with $Q = Q_1 \times Q_2$ and

$$T = T_1 \cup T_2 \cup T_s$$

where $T_s$ is the set of self loop transitions $(s_i, s_i)$ for $s_i \in Q_k, k=1,2$.

The observation and logging for each component service are done locally. Hence, our decomposition method cannot be applied on the product since choosing to observe a transition in a component of the product might force it to be observed in another as well (if the same component is reused). Moreover, strict execution sequence detection is not required, since not knowing the exact interleaving between two “equivalent paths” is not needed. More formally, two consecutive transitions $\tau_i$ and $\tau_{i+1}$ of a path $p$ of $M \times N$ can commute if $\tau_i$ is from $M$ and $\tau_{i+1}$ from $N$, or vice versa. Two paths $p_1 \neq p_2$ of $M \times N$ are equivalent $p_1 \equiv p_2$, if $p_1 = p_2$ after a finite sequence of commutations on $p_1$ (or $p_2$). We say that the product $M \times N$ is observation sequence detectable with the observation of $T_1$ in $M$ and $T_2$ in $N$ if for all paths $p_1, p_2$ of $M \times N$ such that $\text{Obs}_{T_1 \cup T_2}(p_1) = \text{Obs}_{T_1 \cup T_2}(p_2)$, we have $p_1 \equiv p_2$. In this case, compensation can be performed using any of the equivalent (reversed) runs, this will result in the same consistent state. We then have the following desirable property:

**Proposition 4** For a pair of non interacting FSM’s $M$ and $N$, and one of their respective minimal observable sets $T_O_M$ and $T_O_N$, $T_O_M \cup T_O_N$ is a minimal observable set of $M \times N$.

Now, let us consider the more general case where $M$ and $N$ interact, but $M$ has a component $C$ that does not interact with $N$. Then, one can decompose $M$ into $C$ and $M \setminus C$, and compute an observable set $T_i$ of transitions of $C$. We can choose the observations of $C \times T_{2k}$ to be one of the $T_i$.

**Example 2** We consider FSM’s $M = (Q_1, s_0, s_f, T_1)$ (Fig. 7a) and $N = (Q_2, s_0, s_f, T_2)$ (Fig. 7b) representing e-services which allow searching and listening to songs online [3]. The e-services allow different modes of payment and searching for song files by singer/title. Their product $M \times N$ is shown in Fig. 7c. The FSM contain a simple component $C = (Q', s_0, s_f, T')$ such that $T' \cap T_2 = \emptyset$. Now, let us consider minimal observable sets $T_O_M = \{a\}$ and $T_O_N = \{e\}$ of $C$ and $M \setminus C \times N$, respectively. Then, $T_O_M \cup T_O_N$ is a minimal observable set of $M \times N$ as shown in Fig. 7c (with the dashed arrows representing observable transitions).
6 Conclusion

We studied compensation under partial log visibility. To the best of our knowledge, this problem has never been considered in the context of transactional services. With respect to (federated) multi-databases, the problem is analogous to designing a global concurrency control protocol in the absence of complete information of the conflicts at different sites [7]. Here, we take the alternate approach and try to determine the absolute minimal set of actions that needs to be logged such that the service is always compensable (execution sequence detectable). We give a general divide and conquer framework which works on complex hierarchical distributed systems, and gives the absolute minimum number of transition to observe in order to get observability. It provides good complexity results (up to two exponential-s better than the brute force method). As future work, we are investigating fast algorithms which give an approximated size of the minimal observable set of transitions. That is, they give an observable set of transitions, but it may not be minimal. Notice that our decomposition algorithm could be used to improve accuracy (decrease the size) in this approximated framework.

References