Minimal Observability and Privacy Preserving Compensation for Transactional Services

Debmalya Biswas, Blaise Genest

▶ To cite this version:

Debmalya Biswas, Blaise Genest. Minimal Observability and Privacy Preserving Compensation for Transactional Services. Discrete Event Dynamic Systems, 2014, 24 (4), pp.611-646. 10.1007/s10626-013-0177-z . hal-00916645

HAL Id: hal-00916645
https://inria.hal.science/hal-00916645

Submitted on 10 Dec 2013

Noname manuscript No.
(will be inserted by the editor)

Privacy Preserving Minimal Observability for Composite Transactional Services

Debmalya Biswas · Blaise Genest

the date of receipt and acceptance should be inserted later

Abstract For complex services composed of many (component) services, logging is an integral middleware aspect, especially for providing transactions and monitoring. In the event of a failure, the log allows us to deduce the cause of failure (diagnosis) and recover by compensating the executed services (atomicity). However, for heterogeneous services with parts of the functionality provided by multiple organizations, logging details of all executed services 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 minimal number of services that need to be logged, and which is still sufficient to know with certainty the actual sequence of executed services from any given log. Further to privacy issues, the complexity of determining a minimal set of such services to log is actually NP-Complete. To solve *both issues*, we resort to considering each component service as a grey box. Logs are recorded and kept local to each component, and a black-box view of the implementation details of each component is provided. In particular, a service which is reused as a component several times (often observed in real-life services) need not be re-computed each time. We show that this dramatically decreases the complexity up to 2 exponentials. For large monolithic component services that cannot be decomposed simply, we also provide heuristics to compute a small (but not necessarily minimal) number of services to log, and experimentally analyze their accuracy and performance.

1 Introduction

An interesting problem for complex systems is to determine a minimal set of actions that needs to be visible such that a given property holds. This is a well researched problem, and some of the properties for which researchers have tried to determine the required minimal visibility are: observability [21], normality [19], diagnosability [30], testability [3,20].

Our system corresponds to a composite (workflow) Web service. A Web service [2] 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. The composition schema which specifies the execution order of its components, can be modeled as a graph, performing actions on global variables. We do not tackle here the modelization of a composite service as a graph, which should be handled with care to yield a graph of reasonable size (see Section 1.2 and [36]). The component services of a composite service may themselves be composite, leading to a hierarchical composition.

The property that we are interested in is that of transactional atomicity for Web services. A transaction can be considered as a group of actions encapsulated by the operations Begin and Commit/Abort,

Parts of this work appeared in the extended abstracts [5,6]. This work is supported by Create ACTIVEDOC and ANR DOCFLOW projects. Most of this work has been done while the first author was at IRISA/INRIA Rennes.

D. Biswas
Iprova, Lausanne, Switzerland

B. Genest
CNRS, IRISA UMR, Campus de Beaulieu, 35042 RENNES cedex, France.

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 [37]. A compensating operation is an operation capable of semantically undoing the effects of the original operation, e.g. "Cancel Flight Booking" is a compensating operation of "Book Flight". In the event of a failure, compensating operations corresponding to each executed action are executed in the reverse order; leading to a previous stable configuration. To enable such compensation, a log is maintained containing the executed operations and their execution order. This compensation mechanism is on the lines of the Sagas framework [14,10]. This is also the default compensation mechanism followed by e.g. BPEL [7], the current industry standard to model and and execute Web services compositions. For composite service executions distributed over multiple providers, with each provider executing a subset of the (component) services, compensation is enabled by distributed commit specifications such as WS-Coordination, WS-AtomicTransaction and WS-BusinessActivity [35].

The main drawback of the methods used by BPEL and advocated by the WS-specifications, is their assumption that the whole execution log is visible globally. This is clearly impractical, especially in a distributed scenario, where the individual confidentiality concerns of the different providers may prevent them from exposing part of the logs corresponding to the services executed by them. Finally, logging is an overhead for run-time execution, and as such is inefficient with respect to both time and space.

1.1 Contributions

In this work, we address distributed and heterogeneous environments where providers are interested in logging only the required minimum to reduce run-time overhead. Further, the composition schema and logs of each service are confidential as they contain sensitive design and run-time information regarding that service, respectively. Given this, we require that the composite services have (almost) no knowledge about execution aspects of their component services. In this context, we consider two problems (formalized in Section 2):

1. finding a set of services to log as small as possible such that compensation can still be performed (referred to as the minimal compensable set), without requiring any knowledge of the schemas of component services; and
2. perform the undo operations in a hierarchical manner, based only on the (partial) logs of the component.

Towards this end, we make the following contributions:

- The problem of determining a minimal compensable set is unfortunately NP-complete. We show that the problem is NP-complete even if the graph corresponding to the composition schema of a service, is acyclic [23], and is bounded by indegree and outdegree less than 3[5]. The novelty of our proof, which follows the same strategy as [23], is that the encoding to get a unique starting and ending point is both easier to understand and allows a lower in and outdegree (Section 3).
- We then introduce a hierarchical modeling of composite services. Intuitively, a composite service is constructed hierarchically, with each hierarchical level describing the interactions at a different level of abstraction. Hierarchical modeling allows us to reuse services in such a way that the hierarchical algorithm will not re-compute twice the same service, even though it is specified as a component many times in the overall composition. This allows us to obtain algorithms up to one exponential faster than without reusing components. We further show how component services can be replaced by much smaller ones leading to up to another exponential gain in complexity [5].

  In this work, we propose a *privacy preserving variant* of [5]. From a privacy perspective, the main disadvantage of the algorithm in [5] is that it requires a global co-ordinator having visibility over (i) the composition schemas of all services while computing the minimal compensable set, and (ii) then over the logs generated by all services in the hierarchy. Here we restrict the need for information sharing to that between parent-child services only. Moreover, the shared information does not contain any sensitive information, e.g. composition schema, execution log, of the component services. The privacy-preserving algorithm is given in Section 4.3. We present a theoretical complexity analysis which illustrates the benefit of our method both from a privacy-preserving and complexity perspective (Section 4.4), which is later verified experimentally in Section 4.4.

– We finally provide heuristics to compute a small, but not necessarily minimal, compensable set in Section 5. Depending on the use-case, it may be acceptable for some application scenarios to log a few more actions as compared to the computationally expensive process of computing the exact minimum. We propose two heuristics in Sections 5.1 and 5.2, and experimentally analyze their complexity and accuracy in Section 5.3.

1.2 Motivational Scenario

We illustrate the challenges based on a motivational real-life Travel Funds Request service.

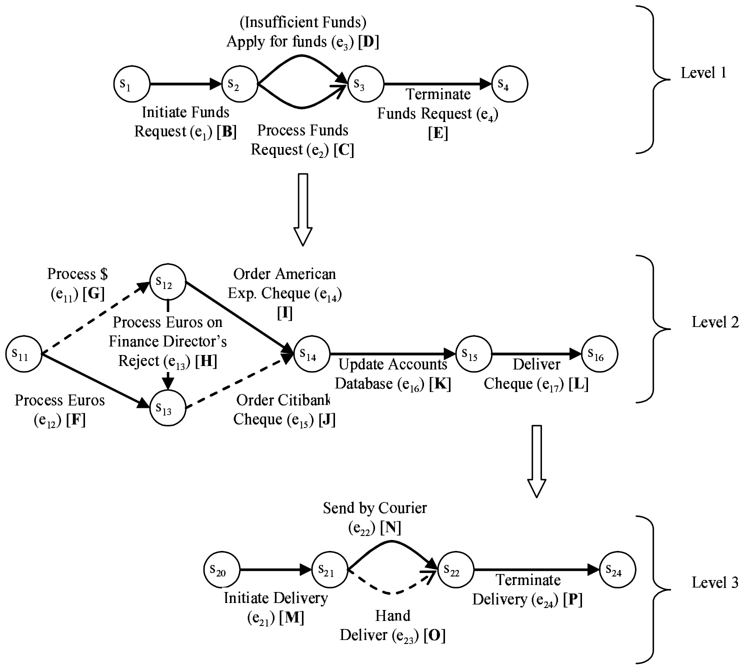
(Insufficient Funds) Apply for funds (e_3) [D]
Initiate Funds Request (e_1) [B]
Process Funds Request (e_2) [C]
Terminate Funds Request (e_4) [E]
Level 1
Process \$ (e_{11}) [G]
Order American Exp. Cheque (e_{14}) [I]
Process Euros on Finance Director's Reject (e_{13}) [H]
Process Euros (e_{12}) [F]
Order Citibank Cheque (e_{15}) [J]
Update Accounts Database (e_{16}) [K]
Deliver Cheque (e_{17}) [L]
Level 2
Send by Courier (e_{22}) [N]
Initiate Delivery (e_{21}) [M]
Hand Deliver (e_{23}) [O]
Terminate Delivery (e_{24}) [P]
Level 3

Fig. 1 Hierarchical Travel Funds Request service

We consider a Travel Funds Request service H in Fig. 1, inspired by the workflow in [29]. It involves different departments across organizations, and it is hierarchical in that the Process Funds Request and the Deliver Cheque services are hierarchically defined. We refer to the services at levels $1, 2$ and 3 as M^1, M^2 and M^3, respectively.

Our first objective is to compute a minimal compensable set of the hierarchical service. Let us try computing minimal compensable sets of M^1, M^2 and M^3 independently. Let $\mathcal{T}_{O^1} = \{e_3\}$, $\mathcal{T}_{O^2} = \{e_{11}, e_{15}\}$ and $\mathcal{T}_{O^3} = \{e_{23}\}$ be the computed minimal compensable sets of M^1, M^2 and M^3, respectively. It is easy to see that $\mathcal{T}_{O^1} \cup \mathcal{T}_{O^2} \cup \mathcal{T}_{O^3}$ is a compensable set of H. However, a minimal compensable set is $\mathcal{T}_O = \{e_{11}, e_{15}, e_{23}\}$ (shown by dashed arrows in Fig. 1), and there is no need for e_3 to be visible in M^1. That is, $\mathcal{T}_{O^1} \cup \mathcal{T}_{O^2} \cup \mathcal{T}_{O^3}$ is not a *minimal* compensable set. While it is not possible for M^1 to take such global decisions based on its local schema information, we show that it is in fact possible for M^1 to do so even without knowing the composition schema of its child M^2 entirely, based only on knowledge of some properties of M^2.

Second, let us consider the problem of performing compensation based on the partially visible log. Note that if we consider the minimal compensable set $\mathcal{T}_O$ computed above, then M^1 does not have even a single visible transition. Given this, it is not possible for M^1 to locally determine its execution sequence from its log (it would for instance not be able to differentiate between the execution paths $e_1e_2e_4 \neq e_1e_3e_4$). It can however do so by interacting with its child service M^2, that is by checking if M^2 was invoked in an execution instance.

1.3 Application Context

From an application perspective, the problems we address here are middleware problems, which would be relevant to say a BPEL execution engine. We have implemented the proposed algorithms and integrated them into the ActiveBpel [1] implementation. In BPEL, each invoke operation can be compensated. In practice, each invoke operation implicitly creates a scope and compensating operations are associated with the scope. In ActiveBpel, a scope is implemented by the class *AeActivityScopeImpl* and the information that might be needed to compensate it is stored in a class variable *mCompInfo* of type *AeCompInfo*. An *AeCompInfo* object basically contains (i) a reference to the scope that completed (execution log), and (ii) a variable snapshot object to record the state of all of its variables (data log). We are mainly interested in the execution log. We do not deal with the data log, as it is not directly useful for recovering the actual sequence of actions. We assume that each service locally maintains its data log properly. Further, as with execution logs, each service does not have any visibility over its components' data logs; visibility over its local data log is sufficient. For instance, service e_{16} in Figure 1 will be responsible for maintaining in its (data) log the amount for the check. If an undo operation is requested, it will update the accounts database without passing this private amount information to other peers. In the event of a failure, the method *getMatchingScopes()* determines the list of scopes to be compensated. Of course, in case of ActiveBPEL, all executed scope references are visible, and the functionality of *getMatchingScopes()* is simply to traverse the hierarchy of scope references to generate the list of executed scopes. In our case, the list of executed scopes would need to be reconstructed from the visible log. The problem is then to determine a minimal set of scopes whose references need to be logged, based on which the complete list of executed scopes can be determined. As with BPEL, we assume that the service providers adhere to the applicable Service Level Agreement (SLA), with respect to not only providing the service functionality but also their failure recovery requirements. This assumes that the providers internally maintain the necessary logs in a durable fashion (replicating, as necessary) so that their services' execution can be compensated, when necessary. In the rare occasion where this is not feasible, the failure (to compensate) will be raised as an 'exception' to its parent. The parent can handle the exception at its level, based on pre-defined exception handlers, or propagate the exception further upwards. Exception handling is beyond the scope of this work, and it suffices to say that this behavior is consistent with existing services composition frameworks e.g. BPEL, and advanced transaction models e.g. [13,9,10] are applicable here.

1.4 Related Works

The minimality problem such that a given system property holds under partial observation has received considerable attention in the Discrete Event Systems (DES) literature. A Discrete Event System [20] can be represented by the pair $G = (M, \mathcal{T}_c)$. The first component M denotes a Mealy Automaton $M = (\Sigma, Q, Y, \delta, h)$ where Σ is the set of events, Q is the set of states, and $\delta : \Sigma \times Q \to 2^Q$ is the state transition function. Y is the output space and $h : \Sigma \times Q \to Y$ is the output function ($h(\sigma, q)$ is the observed output when σ occurs at q). In our case, outputs are the events observed themselves, that is, $Y = \Sigma_O \cup \{\epsilon\}$ where $\Sigma_O \subseteq \Sigma$. The second component $\mathcal{T}_C \subseteq \mathcal{T}$ is the set of controllable events, where the controllability of events is interpreted in a strong sense: a controllable event can be made to occur if physically possible.

With this definition, the problem is to select an optimum output function (or minimal observable set of events with $Y = \Sigma_O \cup \{\epsilon\}$) such that certain properties, e.g., observability, normality, diagnosability, etc. hold. For observability [21] to hold, if the control events following a pair of event traces are different, then the events observation information provided by the output function should be able to discriminate between such a pair of traces. The same property, in a distributed setting, translates to at least one controller being able to distinguish between such a pair of traces, known as Co-observability [28,12]. If in addition to the observability condition, we further require that all events that a controller can control be observable as well, then we have the stronger property of normality [19]. We cannot use the above research results directly as we do not have any such "special" events in our framework, that is, we require the capability to distinguish between any *pair* of events traces (and not only those which enable some special transitions). For diagnosability [30], we need to be able to distinguish between a normal and faulty trace within a bounded number of steps from the time of occurrence of a fault. Our systems stop execution as soon as a transition fails, as such we need the capability to be able to distinguish between traces within 0 steps of failure occurrence. A variant of the above properties is to determine the exact state at any point of time, that is, we need to be able to distinguish between only those event traces which lead to different states, also known as state-observability [25]. The states can be further divided into a set of partitions T, and as long as we can at least distinguish between the partitions using the events observation information, then we have testability [3,20].

The problem of determining a minimal observable set has been shown to be NP-complete for all the above properties. [38,26,18] prove the NP-completeness based on the vertex cover problem, directed graph *st*-cut problem, and *SATISFIABILITY* problem, respectively. Of course, the most relevant NP-completeness proof for us is that of [23] which shows that determining a minimal observable set for compensability is NP-hard even for directed acyclic graphs. We extend the proof to show that the problem is NP-hard even for directed acyclic graphs with indegree and outdegree bounded by 3.

To the best of our knowledge, we are not aware of any heuristics or approximation algorithms for the minimal sensor selection or the uniconnected graph problem. Some works which have considered heuristics for related observability problems are mentioned below. Given observable events with an associated (installation) cost, [39] presents a greedy algorithm based on the observation cost, to select the minimal set of observed events. [27] discusses approximation algorithms to select the minimal set of events to be communicated in a distributed supervisory controllers setting. [26] gives a polynomial time approximation algorithm for a special case of the observability problem. Recent works [11,32,33] have considered the problem of proposing policies for *dynamic* sensor selection which minimize the number of activated sensors at a given time, compared to our motivational scenario where the sensors activated are fixed after the computation stage. Among them, [33] proposes optimal policies.

From a pure transactional perspective, we are close to logging for nested transactions. For the basics of (single level) undo/redo recovery, the interested reader is referred to [17]. [24,22] discuss the logging required to perform undo/redo recovery for nested transaction. Note that redo is required during recovery if a failure occurs before all transactional updates have been written to the stable storage (after commit). In our scenario, transaction updates are applied as and when they occur. Thus, we interested in an undo/no-redo strategy [8], where undo is performed using compensating operations. Of course, the above works assume that all the logs are accessible (visible), and the focus is on specifying the log format so that the nested structure of transactions/subtransactions is maintained, based on which the correct set of updates to be undone/redone can be determined and executed in the right order. Curiously, while the effect of partial log visibility has not received much attention from an atomicity perspective, it is close to the concurrency control problem of providing global serializability in the absence of a central/global Concurrency Control Manager [16]. However, the proposed solutions are more on the lines of mechanisms (e.g., tickets) to make the conflicts at different sites explicit, rather than determining a minimal set of conflicts/sites which need to be visible to be able to provide global serializability.

2 Formalizing Compensability

Formally, we define a transactional service as a 4-tuple Finite State Machine (FSM) $M = (Q, s_0, s_f, \mathcal{T})$, where $(Q, \mathcal{T})$ is a graph ($q \in Q$ is called a state and $t \in \mathcal{T}$ a transition) and $s_0 \in Q$ and $s_f \in Q$ are the initial and final states respectively. Note that unlike the usual definition of FSMs, we ignore the use of alphabets to label transitions in the FSM definition as we do not need to read words. We also assume that there are no outgoing transitions from s_f and no incoming transitions to s_0 (We could deal with FSMs without these requirements, but the proofs would be more technical.) The converse does not hold, that is, there can be a non-initial (final) state with no incoming (outgoing) transitions. Our FSMs are thus graphs with a unique input and output point, also known as the source and sink states respectively. A sequence of transitions $\rho = \tau_1 \cdots \tau_n \in \mathcal{T}^*$ 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 $\mathcal{P}(M)$ the set of initial paths in M. We denote by $|M|$ the size of M, that is, its number of transitions.

Under restricted visibility, M may not have visibility over the execution logs of all its transitions (invocations). Thus, we further need to consider the subset of transitions $\mathcal{T}_O \subseteq \mathcal{T}$ visible to M. For an execution sequence ρ of M, we call visibility projection the execution visibility we have after ρ was executed. We say that a visibility projection σ is uncertain if there exists two paths having the same projection. The service M is execution sequence detectable iff none of its visibility projections are uncertain.

Definition 1 For a service $M = \{Q, s_0, s_f, \mathcal{T}\}$, let $\mathcal{T}_O \subseteq \mathcal{T}$ be the set of visible transitions. The visibility projection $Obs_O : \mathcal{T}^* \longrightarrow \mathcal{T}_O^*$ is the morphism with $Obs_O(a_1 \ldots a_n) = o_1 \cdots o_n$ with $o_i = a_i$ if $a_i \in \mathcal{T}_O$, and $o_i = \epsilon$ if $a_i \in \mathcal{T} \backslash \mathcal{T}_O$, with ϵ the empty word.

That is, $Obs_O(\rho)$ is the subsequence of ρ obtained by eliminating from ρ every occurrence of a transition which is not in $\mathcal{T}_O$. With such a visibility projection Obs_O, the only way of having compensability is to have every transition visible. Indeed, as soon as there exists even one invisible transition, the service is non-compensable. Else, let us take a path $\rho\tau$ with the last transition $\tau \notin \mathcal{T}_O$. Then, $Obs_O(\rho\tau) = Obs_O(\rho)$. A usual way to overcome such a problem is to ask for certainty only up to the last few transitions of the sequence [25]. However, this workaround does not make sense in our framework since if we cannot compensate the very last transition, then there is no point in compensating any transition at all. As such, we design a new visibility mechanism, where the last state reached before failure is monitored, even if the last transition is not logged. In practice, it means that every state that is reached is logged, and overwrite the previous state in a special memory buffer.

Definition 2 (Visibility Projection) Let $M = \{Q, s_0, s_f, \mathcal{T}\}$ be a service, $\mathcal{T}_O \subseteq \mathcal{T}$. The visibility projection $Obs_O^{last} : \mathcal{T}^* \longrightarrow (\mathcal{T}_O^*, Q)$ is the function $Obs_O^{last}(\rho) = (Obs_O(\rho), q)$ for all $\rho \in \mathcal{P}(M)$ ending in q.

Definition 3 (Compensability) Given a service $M = (Q, s_0, s_f, \mathcal{T})$, we call $\mathcal{T}_O \subseteq \mathcal{T}$ a compensable set of transitions if the service is execution sequence detectable with Obs_O^{last}.

We will stick with this definition of compensability for the rest of this work. As mentioned before, we are interested in minimal visibility, that is, visibility over as few transitions as possible.

Problem statement. Given a service $M = (Q, s_0, s_f, \mathcal{T})$, we would like to determine a minimal set of transitions $\mathcal{T}_O \subseteq \mathcal{T}$ which need to be visible such that the system is still compensable.

The cardinality of such a minimal compensable set $\mathcal{T}_O$ of a service M is referred to as its compensable size $MO(M) = |\mathcal{T}_O|$. Note that as is usual with decision and computation algorithms, given a service, it is sufficient to have an algorithm which gives its compensable size. That is, we can derive in polynomial time a minimal compensable set of the service based on an oracle algorithm given the compensable size.

In the next section, we discuss the complexity of our problem.

3 Problem Hardness

We first relate the problem of computing $MO(M)$ using our definition of visibility projections with other known problems. We state now that computing a minimal compensable set is equivalent to the *uniconnected subgraph problem* [15], also called the *minimal marker placement problem* [23], in the meaning of the following proposition.

Proposition 1 *Let $M = \{Q, s_0, s_f, \mathcal{T}\}$ be a service and $\mathcal{T}_O$ a subset of transitions of M. Denote by $M' = \{Q, s_0, s_f, \mathcal{T} \backslash \mathcal{T}_O\}$ the service M obtained by deleting all transitions belonging to $\mathcal{T}_O$. Then, $\mathcal{T}_O$ is a compensable set M iff there does not exist a pair of states $q_1 \neq q_2$ in M' with more than one path between them.*

Proof First, we show that if there does not exist a pair of states $q_1 \neq q_2$ in M' with more than one path between them, then from any visibility projection (σ, q_{n+1}), we can reconstruct in a unique way the path ρ of M with $Obs_O^{last}(\rho) = (\sigma, q_{n+1})$ using the following algorithm:

Algorithm to reconstruct the execution sequence from a given visibility projection.

Input. FSM $M = \{Q, s_0, s_f, \mathcal{T}\}$, visible subset $\mathcal{T}_O \subseteq \mathcal{T}$, FSM $M' = \{Q,\ s_0,\ s_f,\ \mathcal{T} \backslash \mathcal{T}_O\}$ and visibility projection (σ, q_{n+1}).

Output. The unique path ρ of M with $Obs_O^{last}(\rho) = (\sigma, q_{n+1})$.

Initialization. Set $\rho := \epsilon$, current state $s := s_0$ and we use index i to iterate the projection $\sigma = \tau_1\tau_2 \cdots \tau_n$.

```
if ($n == 0$) /* No transitions were logged */
then set $\rho$ to the unique path connecting $s$ to $q_{n+1}$ and return;
else
begin
  for $i = 1 \cdots n$ do
  begin
     if $\tau_i$ is an outgoing transition of $s$
     then append $\tau_i$ to $\rho$;
     else
     begin
        Determine the unique path $\rho_1$ of $M'$ connecting $s$ to ${}^*\tau_i$;
        Append $\rho_1 \tau_i$ to $\rho$;
     endif
     Set $s := \tau_i^*$;
  endfor
  if $s = \tau_n^* = q_{n+1}$
  then return $\rho$;
  else
  begin
     Determine the unique path $\rho_1$ connecting $s$ to $q_{n+1}$;
     Append $\rho_1$ to $\rho$, and return $\rho$;
  endif
endif
```

The converse is trivial. Clearly, if there exists more than one path between two states $q_1 \neq q_2$ of M', then we can find more than one execution sequence corresponding to a visibility projection which passes through q_1 and q_2. $\square$

The fact is that the marker placement problem is an NP-complete problem. We know from [23] 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 graph is both 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 [23], 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 $M = \{Q, s_0, s_f, \mathcal{T}\}$ be a service, 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 $M = \{Q, s_0, s_f, \mathcal{T}\}$ be a system. We reduce Vertex Cover [15] to the problem of finding a subset of transitions $\mathcal{T}_O$ of M, such that there are no two paths $\rho_1 \neq \rho_2$ beginning and ending at the same pair of states, and not using any transitions of $\mathcal{T}_O$.

Let us take an *undirected* graph (V, E) and a number k. We would like to know whether there exists 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 (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 $V_i = \{v_i \mid v \in V\}$ and $E_i = \{e_i \mid e \in E\}$. Furthermore, for $v, w \in V$ and $e \in E$, we have transitions:

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

A graphical representation of M appears in Fig. 2. 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 $\mathcal{T}_O = \{(v_1, v_2) \mid v \in V_O\}$, we have that there are no two paths $\rho_1 \neq \rho_2$ with ρ_1 and ρ_2 beginning and ending at the same pair of nodes, and not using transitions of $\mathcal{T}_O$. By contradiction, else we would have ρ_1, ρ_2 both from some $e_1 \in E_1$ to some $f_2 \in E_2$, and not using transitions of $\mathcal{T}_O$. By definition of $\mathcal{T}_O$, it means that for ρ_1, there exists a node $v \in e$, $v \in f$, such that $v \notin V_O$. Similarly, for ρ_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.

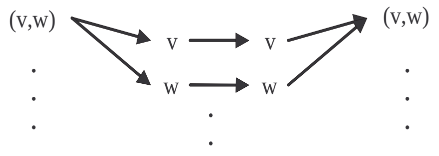

Fig. 2 Illustration for the proof of Theorem 1

Conversely, assume that there is a set of transitions $\mathcal{T}_O$ of size k, such that there do not exist two distinct paths between a pair of nodes without using $\mathcal{T}_O$. We build the set of nodes $V_O = \{v \mid (v_1, v_2) \in \mathcal{T}_O\} \cup \{v \mid \exists e, (e_1, v_1) \in \mathcal{T}_O\} \cup \{v \mid \exists e, (v_2, e_2) \in \mathcal{T}_O\}$. Clearly, $|V_O| \leq |\mathcal{T}_O| = k$. 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 $e_1 v_1 v_2 e_2$ and $e_1 w_1 w_2 e_2$ are two paths not using $\mathcal{T}_O$, a contradiction.

However, so far, the graph defined is not a system since it has several nodes with indegree 0 (the $(e_1)_{e \in E}$), and several nodes with outdegree 0 (the $(e_2)_{e \in E}$). Moreover, the indegree of nodes $(v_1)_{v \in V}$ and the outdegree of nodes $(v_2)_{v \in V}$ 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 incoming transitions from nodes e_1, f_1, g_1 by having two nodes v_1, v_1' with transitions $(e_1, v_1'), (f_1, v_1'), (v_1', v_1)$ 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 compensable 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.

Making the graph a system is a little more involved. We use the graph G from Fig. 1. It then suffices to create a balanced binary tree of transitions with root s_i, 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_f (the unique final node). 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 compensable set of transitions is of size $k + 4|E|$. Indeed, there are $2|E|$ copies of the graph G each of which requires 2 visible transitions. Once these transitions have been deleted, 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 node of the graph G uses one of the two visible transitions), and hence we need exactly k more transitions to be visible. Note that connecting directly the tree with S without using G would not work since it would potentially connect s^0, s^f through two different paths $s^0 \longrightarrow e_1 \longrightarrow e_2 \longrightarrow s^f$ and $s^0 \longrightarrow f_1 \longrightarrow f_2 \longrightarrow s^f$, with $e, f \in E$. $\square$

This theorem does not mean that the problem is impossible to solve, but that it cannot be solved for services with too many transitions. For instance, the complexity of the brute force method which generates every subset of transitions and tests whether it is compensable, is $O(2^{|M|})$ for a service with $|M|$ transitions. A commonly used approach for NP-hard problems is to determine structural properties that make the problem easier to solve, and often hold in real-life scenarios. We propose hierarchical services as a candidate, and outline hierarchical decomposition algorithms for our problem, in the next section.

4 Hierarchical Services

Composite services provide an efficient way to model large and complex services by allowing a modular composition and reusability of services. For simplicity, we model composite services as hierarchical FSMs where two transitions (supertransitions) can be further refined into another FSM. As we will illustrate below, hierarchical FSMs are well suited to represent large systems that are inherently modular. This allows breaking down the minimal compensable problem in terms of each one of its components, which are small enough such that it is computationally feasible to solve the problem for each component. This leads to a computationally feasible answer for the whole (large) system. This is not in contradiction with the NP-complete worst case complexity, as not all systems can be described by a hierarchical FSM with small components. One supertransition per service suffices to break down the system in modular fashion. We additionally *allow* for 2 supertransitions per service to show the computational benefit that can be obtained as a result of reusing component services.

Definition 4 (Hierarchical FSM) A hierarchical FSM H is a finite sequence $\langle M^i \rangle_{i=1\cdots n}$, where $M^i = (Q^i, s_0^i, s_f^i, \mathcal{T}^i, (\tau_1^i, k_1^i), (\tau_2^i, k_2^i))$ is defined as follows:

– $(Q^i, \mathcal{T}^i)$ is a finite graph,
– s_0^i and s_f^i are the initial and final states respectively, and
– $\tau_1^i, \tau_2^i \in \mathcal{T}^i \cup \{\epsilon\}$ are two supertransitions representing FSMs $M^{k_1^i}, M^{k_2^i}$ respectively, with $k_1^i, k_2^i > i$.

Notice that a service can have less than 2 supertransitions by setting $\tau_1^i = \epsilon$ and/or $\tau_2^i = \epsilon$. If both $\tau_1^i = \tau_2^i = \epsilon$, then the module is a (usual) FSM (without supertransition). On the other hand, if $\tau_1^i = (s_1, s_2) \in \mathcal{T}^i$, it means that the transition from state s_1 to state s_2 is actually a supertransition representing the FSM $M^{k_1^i}$.

For instance, the hierarchical service in Fig. 1 can be described as a hierarchical FSM $\langle M^1, M^2, M^3 \rangle$, where the transition e_{17} of M^2 is a supertransition representing M^3, and the transition e_2 of M^1 is a supertransition representing M^2. M^2 will be called a module of M^1, and M^3 a module of M^2.

With each hierarchical FSM H, we associate an ordinary FSM $\mathcal{H}$ obtained by taking M^i, and recursively substituting each supertransition τ^i by the FSM $\mathcal{M}^{k^i}$ it represents. Let $\tau^i = (s_1, s_2)$ and $\mathcal{M}^{k^i} = \{Q, s_1', s_2', \mathcal{T}\}$, then on substituting τ^i of M^i by $\mathcal{M}^{k^i}$, we have $\mathcal{M}^i = \{Q', s_0', s_f', \mathcal{T}'\}$ where

– $Q' = Q^i \setminus \{s_1, s_2\} \cup Q$
– if $s_0^i = s_1$, then $s_0' = s_1'$, else $s_0' = s_0^i$
– if $s_f^i = s_2$, then $s_f' = s_2'$, else $s_f' = s_f^i$
– $\mathcal{T}' = \mathcal{T}^i \setminus \{(s_1, s_2)\} \cup \mathcal{T} \cup \mathcal{I}$, where $\mathcal{I} = \{(q, s_y') \mid (q, s_y) \in \mathcal{T}^i \wedge y \in \{1,2\}\} \cup \{(s_y', q) \mid (s_y, q) \in \mathcal{T}^i \wedge y \in \{1,2\}\}$.

Given a hierarchical service $\langle H_n \rangle$, $\mathcal{H}_j$ is a component of $\mathcal{H}_i$, if H_j is a module of H_i. We define a component FSM as follows:

Definition 5 (Component FSM) An FSM $C = (Q', s_0', s_f', \mathcal{T}')$ is a component of $M = (Q, s_0, s_f, \mathcal{T})$ if $Q' \subsetneq Q$ and $\mathcal{T}' \subsetneq \mathcal{T}$ and $\forall q \in Q \setminus Q'$, $q' \in Q'$, we have $(q, q') \in \mathcal{T}$ or $(q', q) \in \mathcal{T}$ implies $q' \in \{s_0', s_f'\}$.

For example, an FSM C which is isomorph to $\mathcal{M}^2$, is a component of $\mathcal{M}^1$, for the hierarchical services $\langle M^1, M^2, M^3 \rangle$ of Fig. 1 ($\mathcal{M}^1$ is the flat FSM represented by the hierarchical FSM M_1, and $\mathcal{M}^2$ is the flat FSM represented by the hierarchical FSM M_2). In general, there will be n components equivalent with $\mathcal{M}^j$ in $\mathcal{M}^i$, if M^j is used n times in M^i. We define the size $|H|$ of a hierarchical service H as the sum of the number of transitions of its components M^i. Its diameter $||H||$ is the number of transitions of $\mathcal{H}$. Components can be reused several times (for instance, a supertransition of H_3 and two supertransitions of H_4 can represent H_{10}, in which case one does not need to redefine H_{10} three times). Also, the diameter $||H||$ of H can be exponential in the size of H. For instance, consider a simple service H_1 using twice the service H_2, which uses twice the services H_3 etc. till H_n. That is, while defining only n simple composite services, a service H_1 using 2^n service H_n has been defined, with $||H_1|| \geq 2^n$.

Hence, using only two supertransitions per service, one can obtain an exponential compression of the descriptive size of a huge service. Note that if we allow only one supertransition, services cannot be reused as the supertransition satisfies $k^i > i$ (that is, a service can only use a component that has been defined earlier). Indeed, as there is only one supertransition per service, we get $k^i = i + 1$. Given this, if two components M^i, M^j, $i < j$ use the same service k, we get $i + 1 = k = j + 1$; hence $i = j$. Note that if we did not impose $k_1^i > i$, the description would be recursive, and it would represent an infinite system (or equivalently, a family of finite systems), which is undesirable. Finally, many (but not all) systems with more than 2 supertransitions per service can be modeled using additional intermediate services with 2 supertransitions. For instance, a composite service M^1 with three consecutive supertransitions a, b, c, representing respectively services M^2, M^3, M^4, can be encoded as a composite service N^1 with two consecutive supertransitions a', c, where a' represent service N^2 and c still represents M^4. In turn, the service N^2 has two consecutive supertransitions a, b, representing N^2, N^3. It is simple to check that $\mathcal{N}^1 = \mathcal{M}^1$, and that each service of N has at most 2 supertransitions.

We now define properties of components of FSM, before explaining how to replace modules of a hierarchical FSM one by one (even if a module is used twice).

4.1 Component Service FSM Properties

We first define properties of a set of transitions that allow us to know whether it is a minimal compensable set. We say that a path $\rho = \tau_i|_{i=1\cdots n}$ *passes through* an FSM $M = (Q, s_0, s_f, \mathcal{T})$ if $\exists \tau_i \in \mathcal{T}$. We say that a path $\rho = \tau_i|_{i=1\cdots n}$ *belongs to* an FSM $M = (Q, s_0, s_f, \mathcal{T})$ if $\forall \tau_i, \tau_i \in \mathcal{T}$. We say that a path $\rho = \tau_i|_{i=1\cdots n}$ *does not touch* an FSM $M = (Q, s_0, s_f, \mathcal{T})$ if $\forall \tau_i, \tau_i \notin \mathcal{T}$. Further, for an FSM $M = (Q, s_0, s_f, \mathcal{T})$ and subset of transitions $\mathcal{T}_O \subseteq \mathcal{T}$, we define the following predicates: A path ρ is referred to as an invisible path if it does not use any transitions of $\mathcal{T}_O$.

- $P_0(M, \mathcal{T}_O)$ holds if there does not exist more than one invisible path between any two states $s_1 \neq s_2 \in Q$ ($\mathcal{T}_O$ is a compensable set of transitions).
- $P_1(M, \mathcal{T}_O)$ holds if (i) $P_0(M, \mathcal{T}_O)$ holds, and (ii) there does not exist an invisible path from s_0 to s_f. Basically, the existence of an invisible path from the initial to final state of a component C might be a problem for the compensability of a composite service using M as a subservice, if there exists a pair of states $s_1 \neq s_2$ of M with one path passing via C and the other not touching C as shown in Fig. 3(a).
- $P_{1'}(M, \mathcal{T}_O)$ holds if (i) $P_0(M, \mathcal{T}_O)$ holds, and (ii) there do not exist states s_1, $s_2 \in Q$, such that:
  - there is an invisible path from s_0 to s_2,
  - there is an invisible path from s_1 to s_f, and
  - there is an invisible path from s_1 to s_2.

  We refer to such a combination of states and transitions as an invisible reverse cyclic pattern between s_1 and s_2 (within M). Here also, the existence of an invisible reverse cyclic pattern within a component C of M, might be a problem with respect to the compensability of a composite service using M as a subservice, if there exists a path from the final to initial state of C which does not touch C as shown in Fig. 3(b) (because then there are two paths from s'_1 to s'_2: (i) a direct path using (s'_1, s'_2) and (ii) a path via s'_f and s'_0).

By definition, $P_{1'}(M,\mathcal{T}_O) \Rightarrow P_1(M, \mathcal{T}_O) \Rightarrow P_0(M, \mathcal{T}_O)$, since for all s, there always exists a path from s to s. Let $\epsilon < 0 < 1 < 1'$. We define $\text{Best}(M, \mathcal{T}_O) = x \in \{\epsilon, 0, 1, 1'\}$, such that $P_x(M, \mathcal{T}_O)$ holds but not $P_y(M, \mathcal{T}_O)$ with $y > x$, with the convention $P_\epsilon(M, \mathcal{T}_O)$ is always true. Informally, Best refers to the best properties a given set of transitions can ensure, if visible. For instance, in Fig. 1 with $\mathcal{T}_O = \{e_{11}, e_{15}\}$,

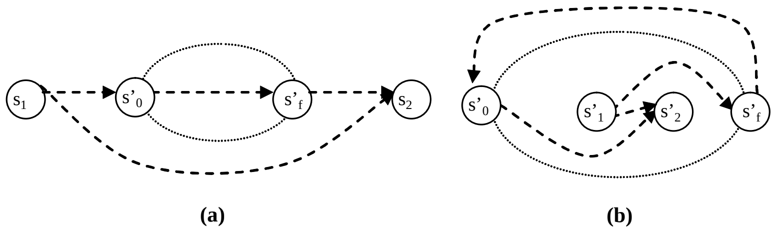

Fig. 3 Significance of $P_1(M, \mathcal{T}_O)$ and $P_{1'}(M,\mathcal{T}_O)$

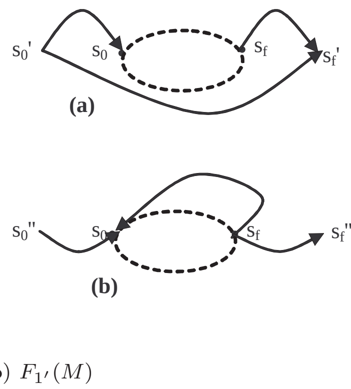

Fig. 4 Computation of (a) $F_1(M)$ and (b) $F_{1'}(M)$

Fig. 5 Minimal sizes computation algorithm
Input. FSM $M = \{Q, s_0, s_f, \mathcal{T}\}$.
Output. Minimal sizes $MO(M), MO_1(M), MO_{1'}(M)$ such that $P_0(M, \mathcal{T}_0), P_1(M, \mathcal{T}_1), P_{1'}(M, \mathcal{T}_{1'})$ hold.
Initialization. Set $MO(M) = MO_1(M) = MO_{1'}(M) = |\mathcal{T}|$.

```
if ($|\mathcal{T}| == 1$)
   then set $MO(M) = 0$ and return;
Compute the powerset $\mathcal{P}(\mathcal{T})$;
for $i = 0 \cdots |\mathcal{T}|$ do
begin
    Determine $\mathcal{P}^i(\mathcal{T}) \subset \mathcal{P}(\mathcal{T})$, such that each set $S \in \mathcal{P}^i(\mathcal{T})$ is of size $|S| = i$;
    for each $S \in \mathcal{P}^i(\mathcal{T})$ do
    begin
       if ($i < MO(M)$)
       then check if $P_0(M, S)$ holds
            then set $MO(M) = i$, else continue; /*'continue' to the next for loop iteration */
       if ($i < MO_1(M)$)
       then check if $P_1(M, S)$ holds
            then set $MO_1(M) = i$, else continue;
       if ($i < MO_{1'}(M)$)
       then check if $P_{1'}(M, S)$ holds
            then set $MO_{1'}(M) = i$ and return;
    endfor
endfor
```

$P_1(M^2, \mathcal{T}_O)$ holds but $P_{1'}(M^2, \mathcal{T}_O)$ does not hold as there is an invisible reverse cyclic pattern between s_{12} and s_{13}. Thus, $\text{Best}(M^2, \mathcal{T}_O) = 1$.

A brute force algorithm to compute the minimal sizes $MO(M), MO_1(M), MO_{1'}(M)$ is given in Fig. 5.

We are now in a position to present our first technical result, stating that two subsets of visible transitions can be interchangeably used in any context as long as they have the same *Best* value.

Proposition 2 *Let C be a component of M, and $\mathcal{T}_1, \mathcal{T}_2$ be subsets of transitions of C respectively, such that Best$(C, \mathcal{T}_1)$ = Best$(C, \mathcal{T}_2)$. Then, for all subset of transitions $\mathcal{T}_O$ of $M \setminus C$, we have Best$(M, \mathcal{T}_O \cup \mathcal{T}_1)$ = Best$(M, \mathcal{T}_O \cup \mathcal{T}_2)$.*

Proof Let $C = (Q', s'_0, s'_f, \mathcal{T}')$ and $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = x$. Let us assume that $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = \epsilon$, that is, there exists a pair of states s_1, s_2 of M with invisible (for $\mathcal{T}_O \cup \mathcal{T}_2$) 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 $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \epsilon$.

If both ρ_1 and ρ_2 do not touch C, then $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \epsilon$. If both ρ_1 and ρ_2 belong to C, then $P_0(C, \mathcal{T}_2)$ does not hold, which means $P_0(C, \mathcal{T}_1)$ does not hold, implying $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \epsilon$.

If $s_1, s_2 \in (Q \setminus Q') \cup \{s'_0, s'_f\}$, and $\rho = \rho_1$ or ρ_2 passes through C, then there exists an invisible (for $\mathcal{T}_2$) path from s'_0 to s'_f (a subpath of ρ). Given this, $P_1(C, \mathcal{T}_2)$ does not hold, which implies that $P_1(C, \mathcal{T}_1)$ does not hold. Hence, there exists an invisible (for $\mathcal{T}_1$) path from s'_0 to s'_f, and an invisible (for $\mathcal{T}_O \cup \mathcal{T}_1$) path ρ' can be constructed from this path and ρ. As such, there are two disjoint paths invisible for $\mathcal{T}_O \cup \mathcal{T}_1$ between s_1 and s_2: $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \epsilon$.

If $s_1, s_2 \in Q'$, and $\rho = \rho_1$ or ρ_2 passes through C, then there exists an invisible reverse cyclic pattern (for $\mathcal{T}_2$) between s'_0 and s'_f. Given this, $P_{1'}(C, \mathcal{T}_2)$ does not hold, which implies that $P_{1'}(C, \mathcal{T}_1)$ does not hold. Hence, there exists an invisible (for $\mathcal{T}_1$) reverse cyclic pattern between s'_0 and s'_f, and an invisible (for $\mathcal{T}_O \cup \mathcal{T}_1$) path ρ' can be constructed from this pattern and ρ. As such, there are two disjoint paths invisible for $\mathcal{T}_O \cup \mathcal{T}_1$ between s_1 and s_2: $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \epsilon$.

The cases where $s_1 \in Q' \setminus \{s'_0, s'_f\}$, $s_2 \notin Q'$, or both ρ_1, ρ_2 pass through C, are not possible because then the paths would meet in s'_0 and/or s'_f.

Hence, $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = \epsilon \implies \text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \epsilon$. By symmetry between $\mathcal{T}_1$ and $\mathcal{T}_2$, we have the equivalence: $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) \geq 0$ iff $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) \geq 0$. Now, for all $x \in \{1, 1'\}$, we can enrich M to $F_x(M)$ with $\text{Best}(M, \mathcal{T}_O) \geq x$ iff $\text{Best}(F_x(M), \mathcal{T}_O) \geq 0$. Applying it to M, we get $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = \text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1)$. The functions F_x are given schematically in Fig. 4. □

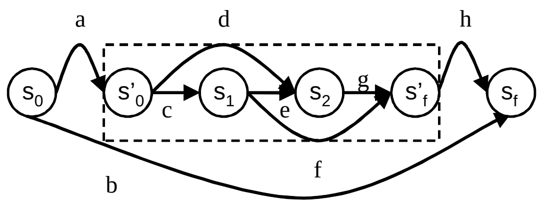

Fig. 6 Service $M = (Q, s_0, s_f, \mathcal{T})$ having simple component $C = (Q', s'_0, s'_f, \mathcal{T}')$.

Example 1 We illustrate the above proposition based on the FSM $M = (Q, s_0, s_f, \mathcal{T})$ in Fig. 6, having simple component $C = (Q', s'_0, s'_f, \mathcal{T}')$.

Let us consider subsets $\mathcal{T}_1 = \{c\}, \mathcal{T}_2 = \{g\}$ of C such that $\text{Best}(C, \mathcal{T}_1) = \text{Best}(C, \mathcal{T}_2) = 1$. That is, P_0 and P_1 hold, but $P_{1'}$ does not hold on deleting either transition c or g from C.

We now show that $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2)$ irrespective of the chosen subset $\mathcal{T}_O$ of $M \setminus C$.

- $\mathcal{T}_O = \emptyset$: $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = 0$.
- $\mathcal{T}_O = \{a\}$: $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = 0$.
- $\mathcal{T}_O = \{h\}$: $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = 0$.
- $\mathcal{T}_O = \{b\}$: $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = 1$.
- $\mathcal{T}_O = \{a, h\}$: $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = 0$.
- $\mathcal{T}_O = \{a, b\}$: $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = 1'$.
- $\mathcal{T}_O = \{b, h\}$: $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = 1'$.
- $\mathcal{T}_O = \{a, b, h\}$: $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = 1'$.

Such Properties and Propositions can be extended to more complex component services with several input and output states, see [4].

4.2 Replacement of a Component

Proposition 2 implies that the only piece of information, a composite service needs to know about the set of transitions $\mathcal{T}$ logged by one of its components C, is $\text{Best}(C, \mathcal{T})$ - a very desirable property from a privacy perspective. In addition to preserving privacy, this also allows us to replace a potentially large component C by a component D of fixed size, improving the complexity. We formalize this in the sequel. Given a component $C = (Q', s'_0, s'_f, \mathcal{T}')$ of $M = (Q, s_0, s_f, \mathcal{T})$ and $D = (Q'', s''_0, s''_f, \mathcal{T}'')$, we denote by $M_C(D)$ the FSM obtained on substituting C by D, that is, $M_C(D) = (\bar{Q}, \bar{s_0}, \bar{s_f}, \bar{\mathcal{T}})$ with

- $\bar{Q} = Q \setminus Q' \cup Q''$,
- if $s_0 \in \bar{Q}$ then $\bar{s_0} = s_0$, else $\bar{s_0} = s''_0$,
- if $s_f \in \bar{Q}$ then $\bar{s_f} = s_f$, else $\bar{s_f} = s''_f$,
- $\bar{\mathcal{T}} = \mathcal{T} \backslash \mathcal{T}' \cup \mathcal{T}'' \cup \mathcal{I}$, where $\mathcal{I} = \{(q, s''_y) \mid (q, s'_y) \in \mathcal{T} \wedge y \in \{0, f\}\} \cup \{(s''_y, q) \mid (s'_y, q) \in \mathcal{T} \wedge y \in \{0, f\}\}$.

Note that $M_C(C) = M$ for all C. Also, note that $|M_C(D)| = |M| - |C| + |D|$. We now define the common characteristics C and D should have, such that C can be safely replaced by D.

Theorem 2 *Let C be a component of an FSM $M = M_C(C)$. We select an FSM D, such that $\forall x \in \{0, 1, 1'\}$, $MO_x(D) - MO(D) = MO_x(C) - MO(C)$. Then, $MO(M) = MO(M_C(D)) + MO(C) - MO(D)$.*

Proof Let $M = (Q, s_0, s_f, \mathcal{T})$ and $C = (Q', s'_0, s'_f, \mathcal{T}')$. Further, let $\mathcal{T}_{OM}$ be a minimal compensable set of $M_C(C)$. Then, $\mathcal{T}_{OM}$ can be partitioned as follows: $\mathcal{T}_{OM} = \mathcal{T}^1_O \uplus \mathcal{T}_{OC}$, where $\mathcal{T}^1_O = \{t \mid t \in \mathcal{T}_{OM} \wedge t \in (\mathcal{T} \backslash \mathcal{T}')\}$ and $\mathcal{T}_{OC} = \{t \mid t \in \mathcal{T}_{OM} \wedge t \in \mathcal{T}'\}$. Let $\text{Best}(C, \mathcal{T}_{OC}) = x$. By definition, $|\mathcal{T}_{OC}| \geq MO_x(C)$. Now, we consider a set $\mathcal{T}_{OM'} = \mathcal{T}^1_O \uplus \mathcal{T}_{OD}$, where $\text{Best}(D, \mathcal{T}_{OD}) = x$ and $\mathcal{T}_{OD}$ is a minimal set such that $P_x(D, \mathcal{T}_{OD})$ holds, that is, $|\mathcal{T}_{OD}| = MO_x(D)$. Then, by Proposition 2, $\mathcal{T}_{OM'}$ is at least a compensable set of $M_C(D)$, and $\mathcal{T}_{OM'} \geq MO(M_C(D)$.

Given this, we can apply the hypothesis

$MO_x(D) - MO(D) = MO_x(C) - MO(C)$
$\Rightarrow |\mathcal{T}_{OD}| - MO(D) \leq |\mathcal{T}_{OC}| - MO(C)$ (as $MO_x(D) = |\mathcal{T}_{OD}|$ and $MO_x(C) \leq |\mathcal{T}_{OC}|$)
$\Rightarrow |\mathcal{T}^1_O| + |\mathcal{T}_{OD}| - MO(D) \leq |\mathcal{T}^1_O| + |\mathcal{T}_{OC}| - MO(C)$ (adding $|\mathcal{T}^1_O|$ on both sides)
$\Rightarrow |\mathcal{T}_{OM'}| - MO(D) \leq |\mathcal{T}_{OM}| - MO(C)$
$\Rightarrow MO(M_C(D)) - MO(D) \leq MO(M) - MO(C)$ (as $MO(M_C(D) \leq |\mathcal{T}_{OM'}|$ and $|\mathcal{T}_{OM}| = MO(M)$).
Symmetrically, D is a component of $N = M_C(D)$, and we have with the same reasoning $MO(N_D(C)) - MO(C) \leq MO(N) - MO(D)$. Noting that $N_D(C) = M$, we get $MO(M) = MO(M_C(D)) + MO(C) - MO(D)$. □

We now give exhaustively every possible graph D one can need to replace a component C. We denote by $\mathcal{D}$ the set of such graphs. We may need an FSM D having one of the following characteristics:

- $MO_{1'}(D) - MO(D) = 0$ (which implies $MO_1(D) - MO(D) = 0$).
- $MO_{1'}(D) - MO(D) = 1$ and $MO_1(D) - MO(D) = 0$. The component FSM M^2 in Fig. 1 exhibits this characteristic.
- $MO_{1'}(D) - MO(D) = 1$ and $MO_1(D) - MO(D) = 1$. It is the case for the service $D = (\{s_1, s_2\}, s_1, s_2, \{(s_1, s_2)\})$.
- $MO_{1'}(D) - MO(D) = 2$ and $MO_1(D) - MO(D) = 1$. The service $D = (Q, s_1, s_4, \mathcal{T})$ where $Q = \{s_1, s_2, s_3, s_4\}$ and $\mathcal{T} = \{(s_1, s_3), (s_1, s_2), (s_2, s_3), (s_2, s_4), (s_3, s_4), (s_1, s_4)\}$ is such an example.

Indeed, for all $C, MO_1(C) - MO(C) \leq 1$. Taking $\mathcal{T}$ with $|\mathcal{T}| = MO_1(C)$ and $P_0(C, \mathcal{T})$, implies that there is at most one invisible path from the initial to the final node of C. It then suffices to add any of the transitions σ of that path, and set $\mathcal{T}' = \mathcal{T} \cup \{\sigma\}$, to get $P_1(C, \mathcal{T}')$. Moreover, if $MO_{1'}(C) - MO_1(C) \geq 1$, it is sufficient to consider a D having $MO_{1'}(D) - MO_1(D) = 1$. Basically, for a component C of M, the presence of several invisible reverse cyclic patterns within C is an issue with respect to the compensability of M, iff there exists an invisible path ρ of M_C connecting the final state of C to its initial state. However, such a path ρ is unique, and it is sufficient for a transition τ of ρ to be visible, irrespective of the actual number (≥ 1) of invisible reverse cyclic patterns within C.

Example 2 Let us consider the FSM $M_C(C)$ in Fig. 7 having component C as shown in Fig. 8. A minimal compensable set of C is shown by dashed arrows in Fig. 8. Then, for C, $MO(C) = 11, MO_1(C) - MO(C) = 0$ and $MO_{1'}(C) - MO(C) = 2$. Fig. 9 shows $M_C(D)$ on substituting component C with a suitable $D = (Q', s'_i, s'_j, \{a', b', c', d', e'\})$. Now, $MO(D) = 2$, and $MO(M_C(D)) = 4$ as shown by the dashed arrows in Fig. 9. Then, applying Theorem 2, $MO(M) = MO(M_C(D)) + MO(C) - MO(D) = 4 + 11 - 2 = 13$. □

Based on the above discussion, we can now state the following proposition:

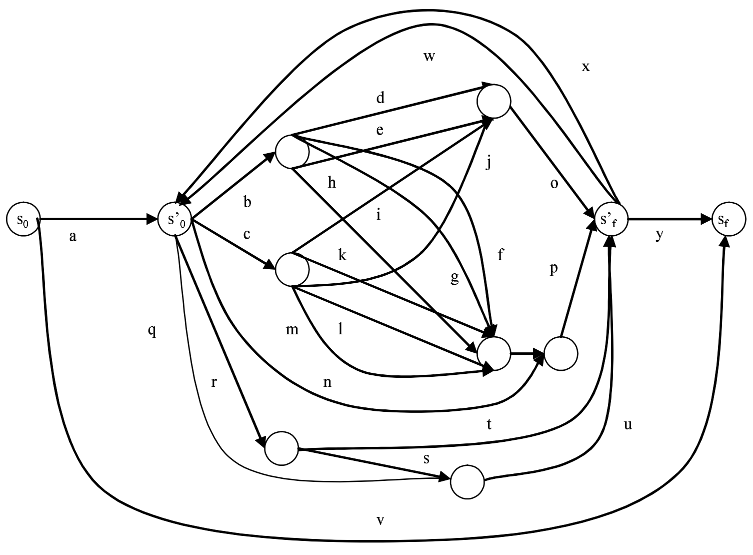

Fig. 7 Sample $M_C(C)$.

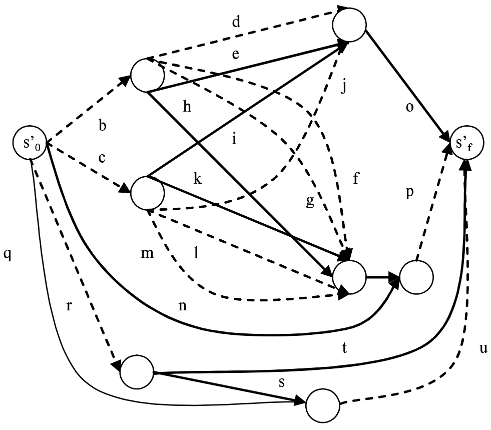

Fig. 8 Component C of M in Fig. 7 (dashed arrows show a minimal compensable set).

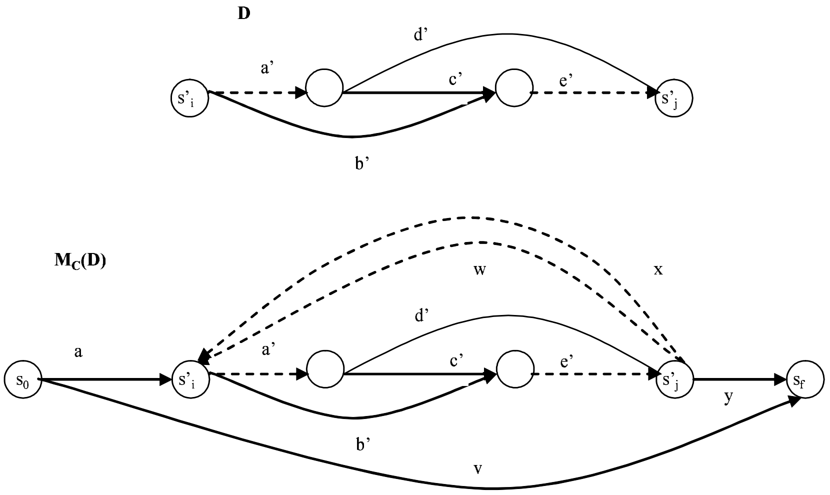

Fig. 9 FSM D and $M_C(D)$ corresponding to the $M_C(C)$ in Fig. 7 (dashed arrows show a minimal compensable set).

Proposition 3 *There exists a constant $Cst \leq 6$ such that for all FSMs C, there exists an FSM D with $|D| \leq Cst$ and $MO_x(D) = MO_x(C)$ for all $x \in \{0, 1, 1'\}$. Also, MO_x can be computed efficiently using an algorithm computing MO.*

Proof It remains to show how to compute MO_1 and $MO_{1'}$. We simply extend M to FSMs M_1 and $M_{1'}$, such that, $MO_1(M) = MO(M_1)$ and $MO_{1'}(M) = MO(M_{1'})$ respectively. The corresponding M_1 and $M_{1'}$ to compute $MO_1(M)$ and $MO_{1'}(M)$, are shown in Fig. 4(a and b) respectively. □

4.3 Privacy Preserving Minimal Compensability Algorithm

So far, we have explained how to replace a component of a FSM. In this section, we present a privacy preserving framework with services in a hierarchy having only as much visibility over others as required. We discuss the information sharing (visibility) required between services in such a restricted visibility environment. During the computation of a minimal set of transitions, the interactions occur in a bottom-up fashion with children sharing some information with their parents. Then, each service assigns a minimal compensable set of transitions, not visible to anyone else than him. Finally, the actual execution, including any compensation required in the event of a failure, also proceeds in a top-down fashion. To

Fig. 10 Privacy preserving minimal compensable set computation of a hierarchical service
Input. Composite service $M_C(C) = (Q, s_0, s_f, \mathcal{T})$ having child service $C = (Q', s'_0, s'_f, \mathcal{T}')$.
Output. Minimal compensable set $\mathcal{T}_O$ of $M_C(C)$.

1. C computes $MO_i(C)$ for all $i \in \{0, 1, 1'\}$, based on the algorithm in Fig. 5.
2. C shares two numbers $x, y \in \{0, 1\}$ with its parent M, such $x = MO_1(C) - MO(C)$ and $y = \max(1, MO_{1'}(C) - MO(C))$.
3. M substitutes C with smaller $D = (Q'', s''_0, s''_f, \mathcal{T}'')$, such that $MO_1(D) - MO(D) = x$ and $MO_{1'}(D) - MO(D) = y$ (if there are two supertransitions, then both are replaced by substitute FSMs).
4. M computes a minimal compensable set $\mathcal{T}_{O'}$ of $M_C(D)$.
5. M partitions $\mathcal{T}_{O'}$ as follows: $\mathcal{T}_{O'} = \mathcal{T}_{O1} \uplus \mathcal{T}_{O2}$, where $\mathcal{T}_{O1} = \mathcal{T}_{O'} \cap (\mathcal{T} \setminus \mathcal{T}'')$ and $\mathcal{T}_{O2} = \mathcal{T}_{O'} \cap \mathcal{T}''$.
6. M assigns the transitions in $\mathcal{T}_{O1}$ as visible and sends $z = \text{Best}(D, \mathcal{T}_{O2}) \in \{0, 1, 1'\}$ to C.
7. C assigns as minimal set of transitions $\mathcal{T}''_O$ as visible, such that $P_z(\mathcal{T}''_O)$ holds ($|\mathcal{T}''_O| = MO_z(C)$).
8. $\mathcal{T}_O = \mathcal{T}_{O1} \cup \mathcal{T}''_O$ is a minimal compensable set of $M_C(C)$.

ease illustration, we only present the information sharing (visibility) required between a pair of a parent service $M_C(C) = (Q, s_0, s_f, \mathcal{T})$ and a child service $C = (Q', s'_0, s'_f, \mathcal{T}')$:

Computation Stage: By computation stage, we refer to the static computation of the minimal compensable set $\mathcal{T}_O$ of M, and assigning the transitions in $\mathcal{T}_O$ as visible.

The privacy preserving algorithm is given in Fig. 10

Note that the only information exchanged between M and C are the numerical values x, y, z. Recall that we substitute a component C, for which $MO_{1'}(C) - MO_1(C) \geq 1$, with an FSM D having $MO_{1'}(D) - MO_1(D) = 1$. Thus, we need to make an exception to the above visibility requirement of C (Step 7) if $\text{Best}(D, \mathcal{T}_{O2}) = 1'$ and $MO_{1'}(C) > MO_1(C)$. Then, we need to add a transition τ of the invisible path of M connecting final state s'_f to initial state s'_0 of C, to $\mathcal{T}_{O1}$ (make τ visible). And, with this addition, it is sufficient for C to have visibility over a minimal subset $\mathcal{T}_{OC} \subseteq \mathcal{T}'$ which satisfies $P_1(C, \mathcal{T}_{OC})$.

Example 3 We illustrate the algorithm in Figure 10 by running it on the hierarchical service M of Figure 7, with a unique service C which M does not have visibility over.

step 1 First M asks C to compute x, y. Thus C computes $MO(C) = 11, MO_1(C) = 11, MO_{1'}(C) = 13$.

step 2 Service C then sends the information $x = 0, y = 1$ to M. Notice that C sends $y = 1$ as $max(1, 13 - 11) = 1$.

step 3 : These x, y mean that C can be considered to behave like D in Figure 9. Thus M computes the FSM $M_C(D)$ as shown in Figure 9.

step 4 Now, M computes a minimal compensable set on $M_C(D)$ and obtain $T_{O'} = \{a', e', x, w\}$ as on Figure 9.

step 5 M sets $T_{O1} = \{x, w\}$ and $T_{O2} = \{a', e'\}$.

step 6 M sets $\{x, w\}$ as visible (it will log these transitions), and computes $z = Best(D, \{a', e'\}) = 1$. It sends $z = 1$ to service C.

step 7 C assigns a minimal set T''_Osuch that P_1 holds, that is $T''_O = \{b, c, r, d, f, g, j, l, m, p, u\}$ (see Figure 8), and set this set as visible (it will log these transitions).

Execution Stage: In this stage, we discuss the interactions required to reconstruct the actual execution sequence from the log (or observation projection) in the event of a failure, and performing the actual compensation. Without global visibility, the data log (which parameter this service has been called with (price of a ticket, etc)) and the execution log (which visible services have been performed and in which order), are maintained locally by each service, and these logs are visible only to that service. Other services, including even their parents or children, do not have visibility over the logs. Given this, we need some mechanism to synchronize the logs of parent-child services. Initially, we assume the use of global timestamps to synchronize parent-child logs (alternate strategies are discussed later). First, we give the execution sequence reconstruction algorithm in Fig. 11. Once the execution sequence of M has been computed, M compensates its executed transitions in the reverse order. For each invocation of component C, it asks C to perform the necessary compensation, which would again involve C computing the execution sequence corresponding to that invocation of itself, and compensating its executed transitions. Finally, M needs to compensate the reconstructed execution sequence ρ. M starts compensating in the usual reverse manner. As soon as it encounters a (s'_0, s'_f), it asks C to perform the necessary compensation. As $\text{Best}(C, \mathcal{T}_{OC}) \geq 0$ (otherwise, $\text{Best}(D, \mathcal{T}_{O2}) < 0$ and there exists greater than one path between a pair

Fig. 11 Compensation Algorithm under Limited Visibility

Input. Visibility projection $(\sigma = \tau_1\tau_2\cdots\tau_n, q_{n+1})$. We assume that $q_{n+1} = s_f$ if a failure did not occur with respect to M (but failure occurred at a higher level, and the successful execution of component FSM M needs to be compensated).

Output. The unique path ρ to be compensated with $Obs_O^{last}(\rho) = (\sigma, q_{n+1})$.

Initialization. Set $\rho := \epsilon$, index $i := 1$, current state $s_{curr} := s_0$. The next state $s_{next} := q_{n+1}$ if $n = 0$ (that is, no transitions were logged), else $s_{next} := in(\tau_1)$ where $\sigma = \tau_1\tau_2\cdots\tau_n$.

M' designates M where the visible transitions $\mathcal{T}_O$ were deleted and the component C with states Q' and transitions $\mathcal{T}'$ was replaced by a transition (s'_0, s'_f). That is, $M' = \{Q \setminus Q' \cup \{s'_0, s'_f\}, s_0, s_f, \mathcal{T} \setminus (\mathcal{T}' \cup \mathcal{T}_O) \cup \{(s'_0, s'_f)\}\}$.

```
while ($s_{curr} \neq q_{n+1}$) do
   begin
      % "Determine the executed path $\rho_1$ from $s_{curr}$ to $s_{next}$"
         if ($\mathcal{P}_{M'}(s_{curr}, s_{next}) = \{\rho_2\}$) (exactly one path between $s_{curr}$,$s_{next}$)
            set $\rho_1 := \rho_2$;
         if ($\mathcal{P}_{M'}(s_{curr}, s_{next}) = \{\rho_2, \rho_3\}$), where $\rho_3$ contains $(s'_0, s'_f)$
            if $C$ was invoked between $s_{curr}$,$s_{next}$ (ask $C$)
               set $\rho_1 := \rho_3$;
            else set $\rho_1 := \rho_2$;
         if ($|\mathcal{P}_{M'}(s_{curr}, s_{next})| > 2$) (implies cycle $s_{curr} \xrightarrow{\rho_4} s_c \xrightarrow{\rho_5} s'_0 \xrightarrow{(s'_0,s'_f)} s'_f \xrightarrow{\rho_6} s_c$)
            Determine the alternate path $\rho_7$ connecting $s'_f$ to $s_{next}$;
            Ask the number $m$ of times $C$ was invoked between $s_{curr}$,$s_{next}$;
               if ($m = 0$)
                  set $\rho_1$ the unique path from $s_{curr}$ to $s_{next}$
                         which does not contain $(s'_0, s'_f)$;
               if ($m = 1$)
                  set $\rho_1$ := $\rho_4\rho_5(s'_0, s'_f)\rho_7$;
               if ($m > 1$)
                  set $\rho_1$ := $\rho_4\rho_5(s'_0, s'_f)\rho_6 \cdots [\rho_5(s'_0, s'_f)\rho_6]_{m-1}\rho_5(s'_0, s'_f)\rho_7$;
      % $\rho_1$ does not depend upon the choice of $s_c$ ($\rho_4, \rho_5, \rho_6$ changes accordingly).
      Append $\rho_1$ to $\rho$;
      if ($n \neq 0$)
         append $\tau_i$ to $\rho$;
      if ($s_{next} = q_{n+1}$)
         set $s_{curr}$ := $q_{n+1}$ (to terminate);
      else set $s_{curr}$ := $out(\tau_i)$;
         if ($i < n$)
            increment $i$ and set $s_{next}$ := $in(\tau_i)$;
         else set $s_{next}$ := $q_{n+1}$;
   end;
```

of states in D with $\mathcal{T}_{O2}$ visible), C can determine its execution sequence corresponding to an invocation based on its local synchronized log, and perform the necessary compensation.

Proposition 4 *With reference to the execution sequence reconstruction algorithm in Fig. 11, if there exists greater than one path between a pair of states $\tau_1^* \neq^* \tau_2, \tau_1, \tau_2 \in \mathcal{T}_{O1}$ of M', then C can determine the number of times $m \geq 0$ it was invoked between an execution of τ_1 followed by τ_2.*

Proof If there exists two paths between $\tau_1^* \neq^* \tau_2$, then $\mathcal{T}_{OC}$ satisfies at least $P_1(C, \mathcal{T}_{OC})$. Otherwise, Best$(D, \mathcal{T}_{O2}) = 0$, and there exists two paths between $\tau_1^* \neq^* \tau_2$ in $M_C(D)$ as well with $\mathcal{T}_{O'}$ visible. The implication of $\mathcal{T}_{OC}$ satisfying at least $P_1(C, \mathcal{T}_{OC})$ is that at least one transition of C is logged each time it is invoked. Given this, C can answer the number of times $m \geq 0$ it was invoked between an execution of τ_1 followed by τ_2, based on its log between the logged times (global timestamps) of τ_1 and τ_2. □

If global timestamps are infeasible, an alternate strategy would be as follows: The FSM M, in addition to logging the transitions in its visible set $\mathcal{T}_O$, also inserts a special marker (say X) in its local log each time it invokes component C. The use of markers leads to some redundancy, and as our goal is to minimize logging, we give another strategy which in most cases leads to a shorter combined log. With this strategy, M no longer needs to log a marker X each time it invokes C. Rather, for each invocation of C, M logs a marker X stamped with its local time (or some unique local identifier) *only if* the last element in its log is a visible transition (and not another marker). For each invocation of C that M inserts a marker X in its local log, it passes the same to C, and C also inserts X in its local log (before logging anything corresponding to that invocation). Given this, each time a marker X is encountered while parsing M's log, we know that the portion of C's log between X and the next marker (or end of the log) corresponds to the execution between that of the visible transitions t_1 and t_2, logged before and after X in M's log respectively, and hence can determine the number of times C was invoked between t_1's and t_2's execution.

Example 4 We give a sample run of the hierarchical recovery mechanism outlined in this section on the hierarchical FSM $\langle M^1, M^2, M^3 \rangle$ in Fig. 1.

To start with (in a bottom-up fashion), M^3 computes $x \in \{0, 1, 1'\}, MO_x(M^3)$, and sends the differences $MO_{1'}(M^3) - MO(M^3) = MO_1(M^3) - MO(M^3) = 1$ to M^2. Then, M^2 substitutes e_{17} with an FSM $D = \{\{s'_0, s'_f\}, s'_0, s'_f, \{(s'_0, s'_f)\}\}$ having similar characteristics (by Theorem 2). Let the substituted M^2 be M', then M^2 computes $x \in \{0, 1, 1'\}, MO_x(M')$, and sends the differences $MO_{1'}(M') - MO(M') = 1$ and $MO_1(M') - MO(M') = 0$ to M^1. On receiving this, M^1 substitutes e_2 by the FSM $E = (Q'', s''_0, s''_f, \mathcal{T}'')$ where $Q'' = \{s''_0, s''_1, s''_2, s''_f\}$ and $\mathcal{T}'' = \{(s''_0, s''_1), (s''_0, s''_2), (s''_1, s''_2), (s''_1, s''_f), (s''_2, s''_f)\}$. Let the substituted FSM M^1 be M'', then we have a minimal compensable size $MO(M'') = 2$.

Once the minimal compensable size has been computed, the next step is to assign the compensable sets, which proceeds in top-down order. Let $\mathcal{T}_{O^1} = \{(s''_0, s''_1), (s''_2, s''_f)\}$ be a minimal compensable set of M''. Given this, M^1 does not need to assign any of its transitions as visible. As $\text{Best}(E, \{(s''_0, s''_1), (s''_2, s''_f)\}) = 1$, then M^2 needs to assign a minimal compensable set $\mathcal{T}_{O^2}$ as visible, such that $\text{Best}(M^2, \mathcal{T}_{O^2}) = 1$, say $\mathcal{T}_{O^2} = \{e_{11}, e_{15}\}$. On the same lines, with only the transitions in $\mathcal{T}_{O^1}$ of M^2 visible, none of D's transitions are visible, as such $\text{Best}(D, \emptyset) = 0$. Then, M^3 only needs to assign a minimal set $\mathcal{T}_{O^3}$ as visible, such that $\text{Best}(M^3, \mathcal{T}_{O^3}) = 0$, say $\mathcal{T}_{O^3} = \{e_{23}\}$. The visible transitions are denoted by dashed arrows in Fig. 1.

Now, let us assume that a failure occurs while executing e_4, and the execution sequence till then is $e_1 e_{11} e_{14} e_{16} e_{21} e_{22} e_{24}$. For simplicity, we also assume that global timestamps are used. Then, the logs at M^1, M^2, M^3 are s_3 (the state before failure), e_{11} and empty, respectively. Given this, M^1 starts the execution sequence reconstruction. There exists two paths $e_1 e_3 \neq e_1 e_2$ between the states s_1 and s_3, one containing the supertransition e_2. Then, the component M^2 corresponding to e_2 is asked to check the number of times it was invoked between the start time and logging time of s_3. M^2 checks its log, finds e_{11} logged during that time, and replies that it was invoked once. So, the execution sequence at M^1 is set to $e_1 e_2$, and M^1 starts compensation in the reverse order. As the first transition to be compensated e_2 is a supertransition, the corresponding component M^2 is asked to compensate its execution first. Before it can compensate, M^2 also first needs to reconstruct its execution sequence. As the first logged transition e_{11} is an outgoing of its initial state s_{11}, e_{11} is appended to its execution sequence ρ. Then, it checks for paths between $e^*_{11} = s_{12}$ and its final state s_{16}. Recall that it is checking for paths in the FSM M^2 from which the visible transitions (including e_{15}) have been deleted. As there exists only one such path $e_{14} e_{16} e_{17}$, it is appended to ρ, leading to the execution sequence $e_{11} e_{14} e_{16} e_{17}$. With the execution sequence determined, M^2 starts compensation in the usual reverse order. As the first transition to be compensated e_{17} is again a supertransition, it asks the corresponding component M^3 to compensate its execution first. For M^3, its log is empty and there exists only one path $e_{21} e_{22} e_{24}$ (after the visible transition e_{23} has been deleted) between its initial state s_{20} and final state s_{24}, leading to the execution sequence $e_{21} e_{22} e_{24}$. The rest of the process is simply invoking the respective compensating transitions of the sequence in the determined execution order. □

4.3.1 Privacy Preserving Property Analysis

Before proceeding, we review the data sharing requirements of our algorithms, and show that they indeed preserve privacy. In a composition, we consider privacy requirements of the component service providers in terms of keeping their composition schemas confidential - not exposing them to their parent providers. This is in sync with current Web services frameworks, where functional details of the service are advertised in a UDDI[1] registry, to enable discovery of services for composition. The advertised details include parameters, e.g. category of services provided, access endpoints needed to invoke the service, etc. The internal composition schema (workflow) is never exposed. This definition holds even for the more semantically enabled frameworks, e.g. OWL-S[2], which allow dynamic composition.

A privacy preserving composition should thus ideally reveal only a black box view of component composition schema to its parent. To achieve minimal compensation, i.e. optimize the logging overhead, our algorithms require a grey box view where the parent $M_C(C)$ needs/gets to know the following information regarding its child C:

– During the computation stage, M_C receives the numbers $x = MO_1(C) - MO(C)$ and $y = MO_{1'}(C) - MO_1(C)$ from C.
– During execution, M can observe $\pi_{M_C}(v)$, the subsequence of transitions of execution sequence v that are in M_C (that is, which are not in C), and in the same order.
– Finally, while performing compensation, C sends to M_C the number m of times C was executed between two time points.

It is clear that the third piece of information, the number of times C was executed, can be deduced easily from $\pi_{M_C}(v)$ (it suffices to count the number of times a transition leading to C occurs in $\pi_{M_C}(v)$ between the two states). This last piece of information is actually needed when M_C only observes a minimal compensable set. However, in violation of the privacy preserving scheme, the parent M could cheat and record every possible invocation of C. We show that even in this case, the privacy of C, in terms of keeping its composition schema confidential, is preserved. For this, we introduce the notion of indistinguishability.

Definition 6 Let $M_C(C)$ be a parent composite service with child component service C. We say that services C, D are *indistinguishable* for M_C if $MO_1(C) - MO(C) = MO_1(D) - MO(D)$, $MO_{1'}(C) - MO_1(C) = MO_{1'}(D) - MO_1(D)$, and for all run v of $M_C(C)$ (resp. $M_C(D)$), there exists a run w of $M_C(D)$ (resp. $M_C(C)$) such that $\pi_{M_C}(v) = \pi_{M_C}(w)$.

Intuitively, indistinguishability implies that M cannot distinguish between its component C and a random FSM D, even with the additional knowledge shared by C. Hence, the confidentiality of C's structure is preserved.

Proposition 5 *Let $M_C(C)$ be a parent composite service with child component service C. Then, for all D such that $MO_1(C) - MO(C) = MO_1(D) - MO(D)$, $MO_{1'}(C) - MO_1(C) = MO_{1'}(D) - MO_1(D)$, we have that C, D are* indistinguishable *for M_C.*

Proof It suffices to prove that for all run v of $M_C(C)$ (resp. $M_C(D)$), there exists a run w of $M_C(D)$ (resp. $M_C(C)$), such that $\pi_{M_C}(v) = \pi_{M_C}(w)$.

Hence, let us take a run v of $M_C(C)$ (the case where v is a run of $M_C(D)$ is symmetrical). We can decompose ρ into $\rho_0 u_1 v_1 u_2 \cdots u_n v_n$, such that for all i, v_i is a maximal factor of v containing only transitions in M_C, and u_i is a maximal factor of v containing only transitions in C. Let u' be an execution of D, from the initial to the final state of D. It is easy to check that $w = v_0 u' v_1 u' \cdots u' v_n$ is an execution of $M_C(D)$, and that $\pi_{M_C}(v) = \pi_{M_C}(w)$. $\square$

The above proposition means that M can only know which type C is, among the four types $x = y = 0$, or $x = 0, y = 1$, or $x = y = 1$, or $x = 1, y = 2$. And, this does not leak the composition schema of C, as C cannot be differentiated from a random FSM D based on this information. Hence, our algorithms are privacy preserving.

4.4 Complexity Analysis and Experiments

We show the complexity of the algorithm to compute a minimal compensable set for hierarchical systems.

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

[1] Universal Description Discovery & Integration (UDDI), `http://uddi.xml.org/`
[2] OWL-S: Semantic Markup for Web Services, `http://uddi.xml.org/`

Proof To obtain the deterministic time complexity, it suffices to apply the decomposition algorithm to both components in the same time. For each level, in a bottom up way, we compute $\mathcal{T}_y(M_i)$ and store it in a table, in a dynamic programming fashion such that we do not have to recompute it every time. As the components are replaced by dummy graphs D_1, D_2, the graphs considered at level i are at most of size $|M_i| + 2Cst$. Applying the brute force algorithm of testing all the set of transitions in the graph to obtain the minimal one, we obtain the complexity for step 3 at level i of $C_i = 2^{|M_i|+2Cst} = O(2^{|M_i|})$. Complexity of step 2 and 4 are negligible, while complexity of step 1 is just reading in the table what has previously been computed at an earlier stage, which is negligible also at stage i. Overall, we need to do this 3 times for each $y \in \{0, 1, 1'\}$ and for each i, giving the complexity stated. We verified it experimentally and give the results in section 4.4.

Clearly, if there exists a compensable set of size n, then it gives us such a polynomial size table. Conversely, it is easy to check in polynomial time whether a polynomial size table is a witness for the fact that there exists a compensable set of size n. It suffices to hierarchically compute Best$(M_i, \mathcal{T}_i)$ and check that Best$(M_n, \mathcal{T}_n) \neq \epsilon$. This gives the non deterministic polynomial complexity of the problem.

At last, the NP-hardness comes from the fact that the problem is already NP-hard even in the much easier case where the hierarchy is restricted to one level, that is we have a graph. It gives the NP completeness. $\square$

It is important to note that since a service is in reality a hierarchical service (with hierarchy height of 1), we know that the problem is at least NP-hard. However, the complexity could be exponentially worse for hierarchical graphs, since a small hierarchical graph can represent an exponentially larger flat graph. We prove that this is not the case. Moreover, we prove that the complexity is linear in the number of hierarchy levels, and exponential only in the size of each component. That is, we prove that with a smart algorithm, one can compute efficiently the absolute minimal compensable size even for huge hierarchical systems, as long as each component is small enough. The best case comparison is with respect to a hierarchical service of diameter $O(2^n)$, having n components of size 2. The brute force non-compositional method run on $\mathcal{H}$ takes time $O(2^{2^n})$, 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).

We also tested our divide and conquer algorithm on hierarchical graphs. First, we choose a number (between one and nine) of base subcomponents in the graph. Then, we generate each of them randomly using the Synthetic DAG generation tool [31]. We then generate inductively a hierarchical graph having these base subcomponents randomly using the same tool, by assigning two edges to these components. There is no reuse of components. For each value, we generate each hierarchical graph and each base subcomponent five times to compute the mean values (because of variation in graph size, runtime and compensable size). We then run both a brute force algorithm and our hierarchical algorithm on these graphs.

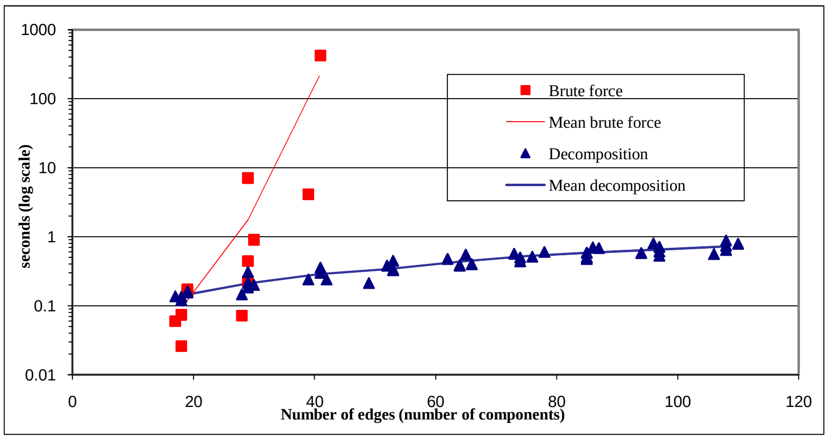

Fig. 12 Execution time vs. Number of edges

Fig. 12 shows the times (in logarithmic scale) needed to compute a minimal compensable set using brute force and our decomposition algorithm with respect to the number of edges (which is linear with respect to the number of base components in this example as we did not reuse components). The decomposition algorithm is clearly much faster than the brute force algorithm. While the brute force is exponential in the number of edges, already timing out at a little over 40 edges, the decomposition algorithm on an average takes $0.73s$ for 108 edges.

5 Approximation

In this section, we study heuristics to obtain a compensable set of transitions, which enables compensation. Indeed, for many application scenarios, it is acceptable to log a few more transitions than the absolute minimum which we did compute with the hierarchical algorithm of the previous section. More importantly, we want to provide a complete framework that can handle composite services even with very large monolithic component services, for which the absolute minimum cannot be computationally computed. For simplicity, we only describe here how to obtain a small (but not necessarily minimal) compensable set of transitions for non-hierarchical FSMs, as it will mainly be used for leaf (bottom) level services in the hierarchy (and hence without components of their own). Extension to compute a compensable set for the whole hierarchy is then trivial.

We now describe two heuristic algorithms for approximating the absolute minimum, with different trade off in term of time and accuracy. This allows users to choose the approach suitable for them.

5.1 (Positively) Discriminating Algorithm

Our first idea is to use observability as a discriminator. A path can be (positively) discriminated from other paths if, for all (intermediate) states it passes through, we know the outgoing transition which was executed. Clearly, for intermediate states having one outgoing transition, the choice is unique and obvious. For intermediate states having $n > 1$ outgoing transitions, it is sufficient if $n - 1$ are visible. Below, we present the polynomial time discriminating algorithm:

Discriminating Algorithm.
Input. FSM $M = (S, s_0, s_f, \mathcal{T})$.
Output. A compensable set $\mathcal{T}_O \subseteq \mathcal{T}$.
Initialization. $\mathcal{T}_O = \epsilon$.

```
for each state $s \in S$ do
     if ($s^* = \{\tau_1 \cdots \tau_n\}, n > 1$)
     then $\mathcal{T}_O = \mathcal{T}_O \cup \{\tau_1 \cdots \tau_{n-1}\}$;
     endif
endfor
```

Example 5 For example, let us consider the FSM M in Fig. 13, which is the flattened representation of the hierarchical FSM in Fig. 1. An output $\mathcal{T}_O$ of the discriminating algorithm with M as input would be $\{e_1, e_2, e_5, e_8\}$. Fig. 14 shows M after the transitions in $\mathcal{T}_O$ have been deleted from M, and is indeed a compensable set of M. □

Proposition 6 *For a given FSM $M = (S, s_0, s_f, \mathcal{T})$, the output $\mathcal{T}_O$ of the discriminating algorithm is a compensable set of M. And, $|\mathcal{T}_O| = |\mathcal{T}| - |S| + 1$.*

Proof The compensability of $\mathcal{T}_O$ follows from the fact that at termination of the algorithm, all states (except the final) of M are left with exactly one outgoing transition. Then, $|\mathcal{T}| - |\mathcal{T}_O| = |S| - 1$, and the compensable size $|\mathcal{T}_O| = |\mathcal{T}| - |S| + 1$. □

5.2 Distinguishing Algorithm

Thinking in terms of positive discrimination is not always good. For example, for the FSM M in Fig. 13, the minimal compensable size is $MO(M) = 3$ ($\{e_3, e_5, e_8\}$ is a minimal compensable set of M), whereas observing any 2 of the 3 outgoing transitions of s_0 always leads to a compensable size greater than 3. Thus, we need an alternate strategy.

We first remark that the deletion of a transition from the given FSM M, can lead to M having more than one initial and/or final state. We call initial (final) state a state which has no incoming (outgoing) transitions. For example, in Fig. 14, s_1 also becomes an initial state after the deletion of transition e_2. The strategy we propose in this section is to select transitions which, for a pair of initial and final states $s_1 \neq s_2$ connected by a path using transition τ, maximize the number of paths not using τ but connecting s_1 to s_2. That is, we now want to maximize the number of paths distinguished by a visible transition. However, computing exactly that number of paths is really inefficient. We thus use an efficient but slightly less accurate version in the sequel.

We want to choose a transition τ which maximizes the number of initial (final) states from (to) which the following condition holds: For an initial (final) state s_1, there exists a final (initial) state s_2 with at least two paths from s_1 to s_2 (s_2 to s_1), one using τ and one not using τ. Clearly, if a transition τ does not figure in one of two such paths with respect to any initial or final state, then it is useless for compensability.

We present the algorithm *Count* based on Depth First Search (DFS). Starting DFS from an initial state s_1, if the target state *t.dest* of the current transition t has been previously explored, then we know that there exists another path not using t from s_1 to *t.dest*. That is, we have two paths connecting s_1 to a final state reachable from *t.dest* (say s_2), one using t and one not using t. We keep a counter *t.count* associated with each transition t, and increase it as soon as we reach an already explored state s via t. Furthermore, for each such hit, we also need to increment the counter of the transition $s.first$ via which s was first reached. Note that we only need to increment the counter of $s.first$ once for each initial state s_1, and not each time s is reached via a different path from s_1. To accommodate this, we use the flag $s.firstflag$, which indicates whether the counter of transition $s.first$ has been incremented with respect to an initial state. We cannot increment the counter of $s.first$ initially, because then we do not know if there exists another path from s_1 to s not using $s.first$. The algorithm keeps a stack K, a hash table H of states which have already been explored by the search, and each transition t has an associated flag to tag it as explored or unexplored. *K.head* designates the head of the stack and push/pop are standard stack operations.

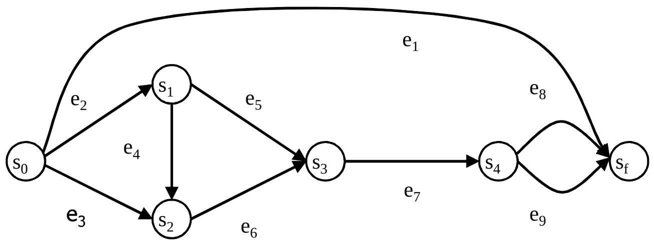

Fig. 13 Flattened representation of the hierarchical FSM in Fig. 1

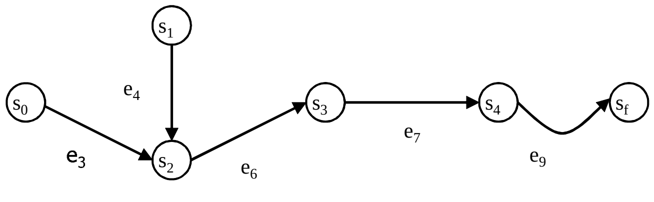

Fig. 14 The FSM M in Fig. 13, after a run of the discriminating algorithm

Algorithm Count.
Input. FSM $M = (S, s_0, s_f, \mathcal{T})$.
Output. For each transition $\tau \in \mathcal{T}$, the count *t.count* of pairs of states $s_1 \neq s_2$, such that there exists two paths connecting s_1 to s_2, one containing τ and the other not containing τ.
Initialization. Create hash table H and $\forall t \in \mathcal{T}, t.count = 0$.

```
for each initial state $s_0$ of $M$ do
    Initialize $H$ to empty, $K$ to $s_0$;
    Set all transition tags to unexplored and $\forall s \in S, s.firstflag$ = false;
    /* Run DFS from $s_0$ */
    while $K$ is nonempty do
       while there is an unexplored transition $t$ from state $K.head$ do
       begin
          Tag $t$ as explored;
          if ($t.dest \in H$)
          begin /* then */
             Increment $t.count$;
             if ($t.dest.firstflag$ == false)
             then increment $t.dest.first.count$ and set $t.dest.firstflag$ = true;
             endif
          end /* then */
          else set $t.dest.first$ = $t$, insert $t.dest$ into $H$, and push $t.dest$ on $K$;
          endif
       endwhile
       Pop $K$;
    endwhile
endfor
```

Obviously, an algorithm *CountBack* can be similarly devised, running from all final states and traversing transitions backwards. Now, our distinguishing algorithm proceeds as follows, choosing randomly to apply *Count* or *CountBack*, as we experimentally observed that it was better than sticking with *Count* only or *CountBack* only:

Algorithm Distinguish.
Input. FSM $M = (S, s_0, s_f, \mathcal{T})$.
Output. Compensable set $\mathcal{T}_O$.
Initialization. Set $\mathcal{T}_O = \emptyset$.

```
loop
begin
   Set all transition counters to $0$;
   Run at random Count or CountBack;
   if all transition counters are $0$
   then return $\mathcal{T}_O$;
   else
   begin
      Select one transition $t$ with maximal counter;
      Add $t$ to $\mathcal{T}_O$;
      Delete $t$ from $\mathcal{T}$;
   endif
endloop
```

The next proposition holds for both *Count* and *CountBack*, hence choosing one, the other, or to inductively randomly applying one or the other does not change the statement.

Proposition 7 *For a service $M = (S, s_0, s_f, \mathcal{T})$, the distinguishing algorithm returns a compensable set $\mathcal{T}_O$ of transitions. Moreover, its size is always $|\mathcal{T}_O| \leq (|\mathcal{T}| - |S| + 1)$.*

Proof The compensability of $\mathcal{T}_O$ follows from the termination condition. Clearly, if all transition counters are 0 at the end of a *Count/Countback*, then there does not remain greater than one path between any pair of states.

The cardinality of $\mathcal{T}_O$ follows from the fact that M remains *connected* at any stage of the algorithm. Let us consider the transitions deleted from $\mathcal{T}$ (added to $\mathcal{T}_O$) during the algorithm. Each such transition t is either the incoming of a state having indegree more than one (by *Count*) or outgoing of a state having outdegree more than one (by *CountBack*). As such, even after the deletion of t from $\mathcal{T}$, there are no disconnected subgraphs, and there still exists a path from the source state of t to a final state (or from an initial state to the target state of t). Given this, $|\mathcal{T}_O| \leq (|\mathcal{T}| - |S| + 1)$ follows from the property that any undirected connected graph has at least $|S| - 1$ edges. □

Example 6 Let us consider a run of the distinguishing algorithm on the FSM M in Fig. 13. Initially, let us assume that *Count* is run. Then, the counter values of the transitions $e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9$ at the end of *Count* are $1, 0, 1, 1, 1, 1, 0, 1, 1$ respectively. Given this, let e_8 is chosen and added to $\mathcal{T}_O$. Fig. 15(a) shows M after the deletion of e_8. Again, let *Count* is run for the next iteration. The counter values of the transitions $e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_9$ then at the end of *Count* are $1, 0, 1, 1, 1, 1, 0, 1$ respectively. Give this, let e_6 is chosen and added to $\mathcal{T}_O$. Fig. 15(b) shows M after the deletion of e_6. For the next iteration, let *CountBack* is run. Note that M at this stage has two final states s_2 and s_f. The counter values of the transitions $e_1, e_2, e_3, e_4, e_5, e_7, e_9$ at the end of *CountBack* are $1, 2, 1, 0, 0, 0, 0$ respectively, leading to the addition of e_2 to $\mathcal{T}_O$. At this stage (Fig. 15(c)), irrespective of whether *Count* or *CountBack* is run, the counter values of all remaining transitions are 0, terminating the algorithm. Indeed, $\mathcal{T}_O = \{e_2, e_6, e_8\}$ is a compensable set of M (actually minimal in this case). □

Note that there is a certain amount of randomness inherent in the algorithm. For example, $\mathcal{T}_O = \{e_1, e_2, e_5, e_8\}$ is also a valid output of the distinguishing algorithm on M. To offset this randomness, we took the minimum of ten runs during our experimental evaluation (discussed in the next section).

5.3 Experimental Evaluation

We have some theoretical clues about how our algorithms fair against each other, and how close they can approximate the absolute minimal compensable size. Because our discriminating and distinguishing algorithms are polynomial time, and the problem is not approximable [26], we know that there are FSMs on which they give an answer far away from the optimum. The first question is how far they are, and how often it happens. Also, the distinguishing algorithm gives a set never bigger than the one given by the discriminating algorithm. The second question then is: is it better, and if yes, by how much and how often is it much better.

The first question is difficult to answer accurately, since obtaining the absolute minimal compensable size is intractable. One solution could be to look at small enough FSMs to get the values, but the problem is a variation of one visible transition having a big impact percentage wise in small sets, so the results would not be very meaningful. Instead, we can focus on hierarchical FSMs where all the modules are small, and use the algorithm presented in Section 4.3 to compute its minimal compensable size. As we want to deal with FSMs having very large components, this is not very meaningful. We refer to [6] for experimental results. In short, the three algorithms fare almost identically with respect to compensable size, with slightly smaller compensable sets obtained using the algorithm of Section 4.3, and slightly bigger sets for the discriminating algorithm. Computational time is reversed, that is the distinguishing algorithm takes 2.5 seconds at most, the discriminating algorithm is almost instantaneous, and the algorithm of Section 4.3 takes around half an hour for 700 transitions.

We now turn to general FSMs (on which we cannot compute the absolute minimum). We use the *Synthetic DAG Generation Tool* to generate them, with random parameters for each size of FSM from 80 to 1200 transitions.

Fig. 16 shows the results using the discriminating (called matrix) and distinguishing algorithms, according to the number of transitions of the FSM. The graph shows chaotic picture. Furthermore, the distinguishing algorithm seems to often do much better than the discriminating algorithm. Still,

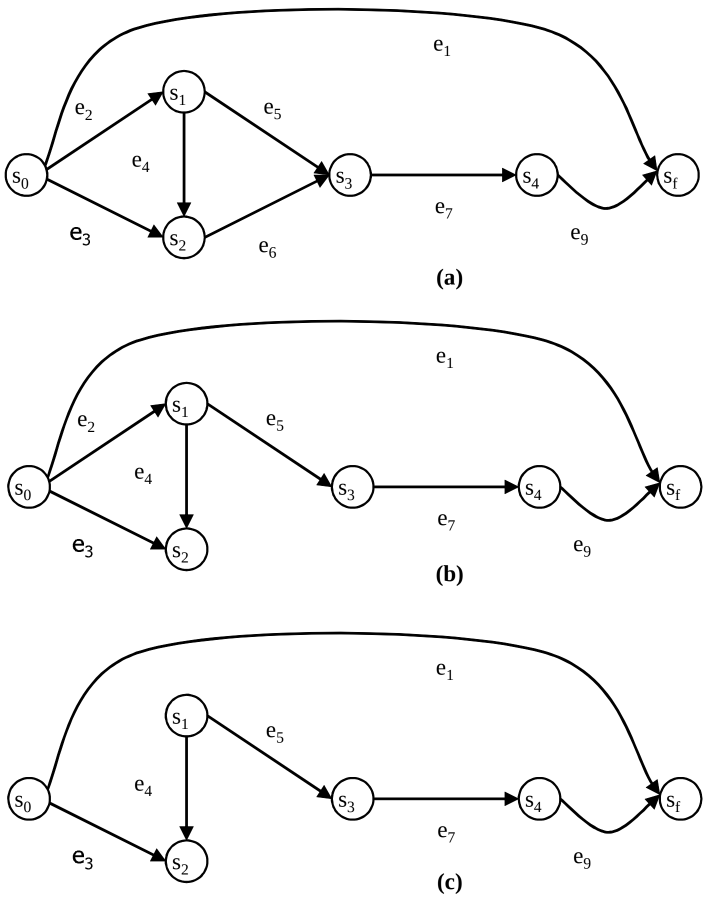

Fig. 15 The FSM in Fig. 13 at different stages of the distinguishing algorithm

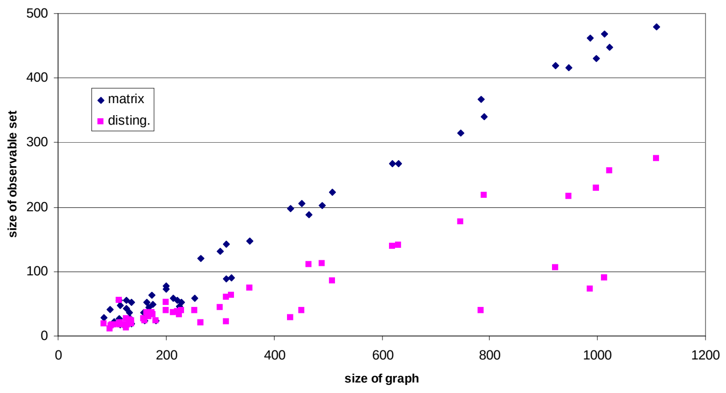

Fig. 16 Compensable size vs. number of transitions over general FSMs

there are several cases (around 100 transitions) where both give the same results. Concerning time, the distinguishing algorithm takes at most 15 seconds to perform.

Let us now analyze the percentage of transitions logged by the different algorithms (Fig. 17). We see the chaosness of the picture, ranging from 15% to 50% of transitions logged by the discriminating algorithm (mean value 34%), and from 4% to 50% for the distinguishing algorithm (mean value 18%). The percentage of transitions can vary from 1 to 10

Finally, we give a summing up graph in Fig. 18, where we put each random FSM we generated according to the percentage of transitions logged by the discriminating algorithm and by the distinguishing

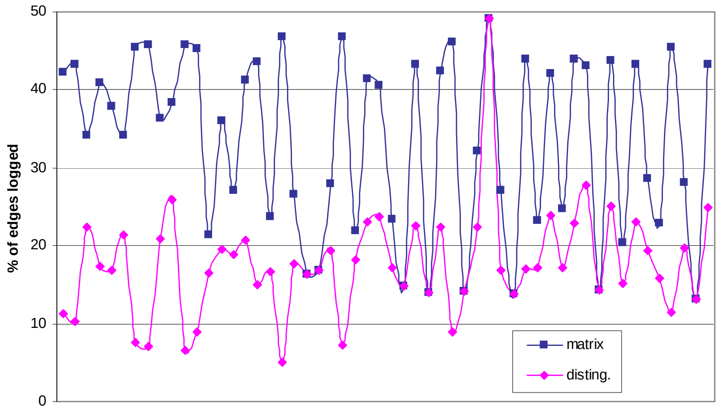

Fig. 17 Percentage of edges logged by the discriminating and distinguishing algorithms over general FSMs

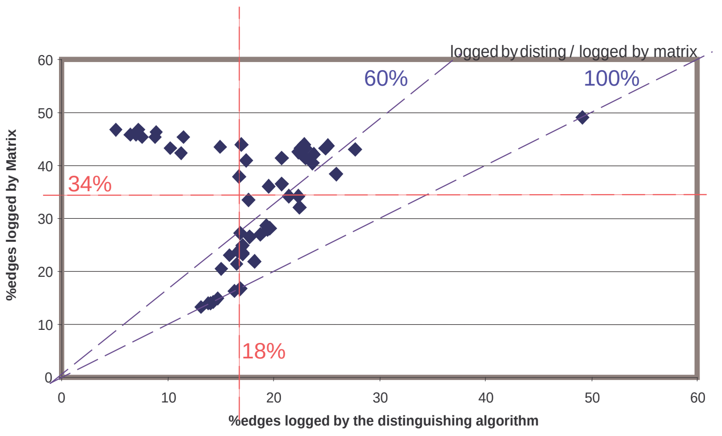

Fig. 18 Percentage of edges logged by the discriminating vs. distinguishing algorithm over general FSMs

algorithm, together with two broken lines labeled by 18% (vertically for the distinguishing algorithm) and 34% (horizontally for the discriminating algorithm) showing the mean values of the percentage of transitions logged. On this graph, we can draw the line (broken line labeled 100%) on which both algorithms perform similarly. We can see that it happens several times, but mainly when the discriminating algorithm already gives good results (from 12% to 18% of transitions logged, which shall be close if not equal to the absolute minimum). Only once, the discriminating is bad and so is the distinguishing algorithm (around 50% of transitions logged). It was to be expected that such cases occur, since we know that the absolute minimum is not approximable, but luckily, it is pretty rare.

Overall, the distinguishing algorithm gives a compensable size 0.6 times the size returned by the discriminating one (we draw a broken line labeled by 60% to separate the experiments under and over that value), and almost all of its answers are within 0.7 times of the discriminating algorithm. Moreover, it sometimes gives one tenth the number of transitions to log compared to the discriminating algorithm (which implies that the discriminating algorithm can be very inaccurate). Also, note that only once, the distinguishing algorithm gives more than 30% of transitions to log (1.5% of the FSMs), while it is the case for 50% of the FSMs with the discriminating algorithm.

6 Conclusion

We showed in this paper how to develop evolved algorithms for services computing, the new paradigm of computing. This new paradigm enforces strong constraints on privacy, security, reliability and heterogeneity that have to be handled. We presented an efficient compensation framework that is privacy preserving as well. We designed the algorithms in a pure hierarchical way. We showed how these algorithms preserve the privacy of each component (including log privacy and the way the services are implemented internally). Furthermore, we obtained as a byproduct much faster complexity. In addition, we also gave heuristics to compute a compensable set. There are probably some more heuristics to apply to get a more accurate algorithm. Nevertheless, in mean value, it seems that the distinguishing algorithm gives results close to the absolute minimum (18% of transitions, while we get 20% for the absolute minimum, looking at hierarchical FSMs), so efforts to optimize it further would probably not be worth it but for very few pathological cases, and would slow down the algorithm.

References

1. Activebpel BPEL Implementation. *http://www.activebpel.org.*
2. Gustavo Alonso, Fabio Casati, Harumi Kuno and Vijay Machiraju. Web services: Concepts, Architecture and Applications. *ISBN: 3540440089 (Springer Verlag)*, 2004.
3. Sanjiv Bavishi and Edwin K. P. Chong. Automated Fault Diagnosis using a Discrete Event Systems Framework. *in: Proceedings of the 9th IEEE International Symposium on Intelligent Control (IC) (IEEE Computer Society Press)*, pages 213–218, 1994.
4. Debmalya Biswas. Visibility in Hierarchical Systems. *IRISA/INRIA PhD Thesis*, 2009, *http://perso.crans.org/~genest/ThesisDebmalya.pdf.*
5. Debmalya Biswas and Blaise Genest. Minimal Observability for Transactional Hierarchical Services. *in: Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 531–536, 2008.
6. Debmalya Biswas, Thomas Gazagnaire and Blaise Genest. Small Logs for Transactional Services: Distinction is much more Accurate than (Positive) Discrimination. *in: Proceedings of the 11th IEEE High Assurance Systems Engineering Symposium (HASE) (IEEE Computer Society Press)*, pages 97–106, 2008.
7. Business Process Execution Language for Web Services (BPEL) Specification. *http://www-106.ibm.com/developerworks/library/ws-bpel/.*
8. Debmalya Biswas and Krishnamurthy Vidyasankar. A Nested Transaction Model for LDAP Transactions. *in: Proceedings of the 1st International Conference on Distributed Computing and Internet Technology (ICDCIT), Lecture Notes in Computer Science vol. 3347 (Springer Verlag)*, pages 117–126, 2004.
9. Debmalya Biswas and Krishnamurthy Vidyasankar. Optimal Compensation for Hierarchical Web Services Compositions under Restricted Visibility. *in: Proceedings of the 4th IEEE Asia-Pacific Services Computing Conference (APSCC) (IEEE Computer Society Press)*, pages 293–300, 2009.
10. R. Bruni, H. Melgratti and U. Montanari. Theoretical Foundations for Compensations in Flow Composition Languages. *in: Proceedings of the 32nd ACM Symposium on Principles of Programming Languages (POPL) (ACM Press)*, pages 209–220, 2005.
11. Franck Cassez and Stavros Tripakis. Fault Diagnosis with Static and Dynamic Observers. *Fundamenta Informaticae, 88*, pages 497540, 2008.
12. Randy Cieslak, C. Desclaux, Ayman S. Fawaz and Pravin Varaiya. Supervisory Control of Discrete Event Processes with Partial Observation. *IEEE Transactions on Automatic Control, 33(3)*, pages 249–260, 1988.
13. C. Hagen and G. Alonso. Exception Handling in Workflow Management Systems. *IEEE Transactions on Software Engineering, 26(10)*, pages 943–958, 2000.
14. Hector Garcia-Molina and Kenneth Salem. Sagas. *ACM SIGMOD Record 16(3)*, pages 249–259, 1987.
15. Michael R. Garey and David S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. *ISBN: 9780716710455 (W. H. Freeman and Company)*, 1979.
16. Dimitrios Georgakopoulos, Marek Rusinkiewicz and Amit P. Sheth. Using Tickets to Enforce the Serializability of Multidatabase Transactions. *IEEE Transactions on Knowledge and Data Engineering, 6(1)*, pages 166–180, 1994.
17. Vassos Hadzilacos Philip A. Bernstein and Nathan Goodman. Concurrency Control and Recovery in Database Systems. *ISBN: 0201107155 (Addison-Wesley)*, 1987.
18. Shengbing Jiang, Ratnesh Kumar and Humberto E. Garcia. Optimal Sensor Selection for Discrete-event Systems with Partial Observation. *IEEE Transactions on Automatic Control, 48(3)*, pages 369–381, 2003.
19. Ratnesh Kumar and Vijay K. Garg. Modeling and Control of Logical Discrete Event Systems. *ISBN: 9780792395386 (Springer)*, 1994.
20. Feng Lin. Diagnosability of Discrete Event Systems and its Applications. *Discrete Event Dynamic Systems (Springer Netherlands), 4(2)*, pages 197–212, 1994.
21. Feng Lin and W. Murray Wonham. On Observability of Discrete-event Systems. *Information Sciences (Elsevier Science), 44(3)*, pages 173–198, 1988.
22. David B. Lomet. MLR: A Recovery Method for Multi-level Systems. *in: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD Record 21(2) (ACM Press)*, pages 185–194, 1992.
23. S. Maheshwari. Traversal Marker Placement Problems are NP-Complete. *Research Report, Colorado Boulder University, USA*, 1976.
24. J. E. B. Moss. Log-based Recovery for Nested Transactions. *in: Proceedings of the 13th International Conference on Very Large Data Bases (VLDB)*, pages 427–432, 1987.
25. Cuneyt M. Ozveren and Alan S. Wilsky. Observability of Discrete Event Dynamical Systems. *IEEE Transactions on Automatic Control, 35(7)*, pages 797–806, 1990.
26. Kurt Rohloff, Samir Khuller and Guy Kortsarz. Approximating the Minimal Sensor Selection for Supervisory Control. *Discrete Event Dynamic Systems (Springer Netherlands), 16(1)*, pages 143–170, 2006.
27. Kurt Rohloff and Jan van Schuppen. Approximating Minimal Communicated Event Sets for Decentralized Supervisory Control. *in: Proceedings of the 16th IFAC World Congress (Elsevier Science)*, 2005.

28. Karen Rudie and W. Murray Wonham. Think Globally, Act Locally: Decentralized Supervisory Control. *IEEE Transactions on Automatic Control, 37(11)*, pages 1692–1708, 1992.
29. Wasim Sadiq and Maria E. Orlowska. Analyzing Process Models using Graph Reduction Techniques. *Information Systems 25(2) (Elsevier Science)*, pages 117–134, 2000.
30. Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinaamohideen and Demosthenis Teneketzis. Diagnosability of Discrete Event Systems. *IEEE Transactions on Automatic Control, 40(9)*, pages 1555–1575, 1995.
31. Synthetic Directed Acyclic Graph Generation Tool. *http://www.loria.fr/~suter/dags.html.*
32. Weilin Wang, Stéphane Lafortune, Feng Lin, and Anouck R. Girard. Minimization of Dynamic Sensor Activation in Discrete Event Systems for the Purpose of Control. *IEEE Transaction on Automatic Control, 55(11)*, pages 2447-2461, 2010.
33. Weilin Wang, Stéphane Lafortune, Anouck R. Girard, Feng Lin. Optimal sensor activation for diagnosing discrete event systems. *Automatica, 46(7)*, pages 1165-1175, 2010.
34. Gerhard Weikum, Andrew Deacon, Werner Schaad and Hans-Jorg Schek. Open Nested Transactions in Federated Database Systems. *IEEE Data Engineering Bulletin 16(2)*, pages 4–7, 1993.
35. Web Services Transactions Specifications. *http://msdn2.microsoft.com/en-us/library/ms951262.aspx.*
36. Andreas Wombacher, Peter Fankhauser and Erich Neuhold. Transforming BPEL into Annotated Deterministic Finite State Automata for Service Discovery. *in: Proceedings of the 2nd International Conference on Web Services (ICWS) (IEEE Computer Society Press)*, pages 316–323, 2004.
37. Debmalya Biswas. Compensation in the World of Web Services Composition. *in: Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC) (LNCS 3387)*, pages 69–80, 2004.
38. Tae-Sic Yoo and Stéphane Lafortune. NP-completeness of sensor selection problems arising in partially observed discrete-event systems. *IEEE Transactions on Automatic Control, 47(9)*, pages 1495–1499, 2002.
39. Stanley D. Young and Vijay K. Garg. Optimal sensor and actuator choices for discrete event systems. *in: Proceedings of the 31st Allerton Conference on Communication, Control, and Computing*, 1993.

7 Appendix

Computing the Largest Component In this section, we address the problem that the hierarchical structure specifying the components at each level is not available. It can be the case if the service was build in a monolithic fashion, or more pragmatically if it is not accessible anymore. That is, given a (flat representation of a) composite service, we would like to recover the hierarchical structure from it. Also, the effectiveness of our divide and conquer algorithms (Section 4.2) are clearly proportional to the size the components, that is, the larger the components the better as large components can possibly be refined further (which would imply that we should be interested in the smallest component C, however then the M_C would be large). Towards this end, we show how to recover the largest component from a given composite service M in Section 7. First, we present a linear time (in the number of transitions) algorithm to compute a smallest component C of an FSM M, knowing its initial, final state and an outgoing transition of the initial state.

Algorithm to compute a smallest component C of the given service M.
Input. A service $M = (Q,\, s_0,\, s_f,\, \mathcal{T})$, $\tau = (s, s_1) \in \mathcal{T}$ and $t \in Q$.
Output. A smallest component $C = (Q', s, t, \mathcal{T}')$ of M with $\tau \in \mathcal{T}'$.
Initialization. $\mathcal{T}' = \phi$, $S = \{\tau\}$, $Q' = \{s, s_1, t\}$.

1. Select a transition $\tau' = (s'_1, s'_2) \in S$. If $s'_2 \neq t$, then $S = S \cup \{(s'_2, s'_3) \mid (s'_2, s'_3) \in \mathcal{T} \setminus \mathcal{T}'\}$. If $s'_1 \neq s$, then $S = S \cup \{(s'_3, s'_1) \mid (s'_3, s'_1) \in \mathcal{T} \setminus \mathcal{T}'\}$. Finally, $S = S \setminus \{\tau'\}$, $\mathcal{T}' = \mathcal{T}' \cup \{\tau'\}$, and $Q' = Q' \cup \{s'_1, s'_2\}$.
2. If $S \neq \emptyset$, repeat step 1.
3. If $(s \neq s_0 \wedge s_0 \in Q') \vee (t \neq s_f \wedge s_f \in Q')$, then return that a component between s and t with respect to τ does not exist. Else, return C.

The above algorithm can be iteratively invoked to compute the set S_C of all components of an FSM M. We now give an algorithm to compute a largest component of M.

Algorithm to compute a largest component C of the given service 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 $D = (Q', s'_0, s'_f, \mathcal{T}')$ and $E = (Q'', s''_0, s''_f, \mathcal{T}'')$ of S_C, if $s'_0 = s''_0$ and $s'_f = s''_f$, then create a new component $F = (Q' \cup Q'',\, s'_0,\, s'_f,\, \mathcal{T}' \cup \mathcal{T}'')$. If $F \neq M$, then delete D and E from S_C, and add F to S_C.
3. Return the biggest $C \in S_C$.

Using the above algorithm, a largest component of given service M can computed in quadratic time. The algorithm can thus be called inductively until there are no more components in the determined component FSM N of a level, and then the hierarchical structure of M has been obtained.