Hierarchical Algorithms for Preserving Privacy of Composite Services Compensation

Debmalya Biswas\textsuperscript{1} and Blaise Genest\textsuperscript{2}

\textsuperscript{1} SAP Research, Vincenz-Priessnitz-Strasse 1, Karlsruhe, Germany
debmalya.biswas@sap.com
\textsuperscript{2} IRISA/CNRS, Campus Universitaire de Beaulieu, Rennes, France
blaise.genest@irisa.fr

Abstract. Designing algorithms taking into account the hierarchical structure of systems is known to improve dramatically the complexity. We show here, through the analysis of compensation of Web services compositions, that it can also be really efficient at preserving privacy. Systems we consider are hierarchical finite state machines, each component corresponding to a (composite) service invocation. In the event of a failure, logs are used to recover the sequence of actions that was executed: each action is compensated, in reverse order of occurrence, leading to a stable state. However, for heterogeneous services, logging all the actions is often impracticable due to security issues and expensive in terms of both time and space. We thus design two algorithms. The first one computes a set of actions to log ensuring that the sequence of executed actions can be uniquely determined. This set should be as small as possible. The second algorithm shows how to perform compensation based on this log, in the event of a failure. Furthermore, the algorithms preserve privacy and handle heterogeneity by exchanging only a very limited amount of information between different components: logs are recorded and kept local to each component, and a black-box view of the implementation details of component services is sufficient.

1 Introduction

Hierarchical systems are often used to depict large systems with an intuitive, modular and compressed representation. Such a representation has been used for words [11], Finite State Machines [3], and even semi-structured data [9]. For words, for instance, hierarchical structures correspond to the LZ compression [11]. A major advantage of such structures, is that hierarchical algorithms can be designed to take advantage of the hierarchy in order to speed up the process [3, 6, 9, 11]. For instance, pattern matching can be performed very quickly in a LZ compressed word without uncompressing it [11]. We show here that other forms of improvement can be obtained from hierarchical structures as well. Indeed,
modularity allows to design algorithms that handle heterogeneous formalisms for each component, and which preserve privacy of each component. However, hierarchical algorithms are not straightforward to design, and not all hierarchical algorithms preserve privacy [6].

Web services are particularly interesting systems with strong requirements on privacy, security, reliability and heterogeneity. Web services [2] have become the de-facto standard for providing services on the Web. In order to provide new services, a composite service is composed of already existing (component) services, combining their capabilities. Business Process Execution Language for Web Services (BPEL) [7] is the current industry standard to model and execute Web services compositions. In order to provide reliability, each invoke operation can be compensated in BPEL, such that a previous stable state can be reached when a failure occurs. It follows existing Web services specifications such as WS-Coordination, WS-AtomicTransaction and WS-BusinessActivity. These are basically distributed agreement protocols that are based on the assumption that “all state transitions are reliably recorded” and can be compensated. Compensation consists of executing the compensating actions, corresponding to each executed action of the failed process, in reverse order of the original execution. In practice, each invoke operation implicitly creates a scope and compensating operations are associated with the scope. Taking a particular BPEL execution engine implementation, namely ActiveBPEL [1], a scope is implemented by the class \textit{AeActivityScopeImpl} and the information that might be needed to compensate it is stored in a class variable \textit{mCompInfo} of type \textit{AeCompInfo}. On successful compensation of a scope, its compensation information \textit{mCompInfo} is passed to its parent. In the event of a failure, the method \textit{getMatchingScopes()} of class \textit{AeActivityCompensateImpl} determines the list of scopes to be compensated. Of course, in case of ActiveBPEL, all executed scope references are visible, and the functionality of \textit{getMatchingScopes()} is simply to traverse the hierarchy of scope references to generate the list of executed scopes.

The main drawbacks of the methods used by ActiveBPEL and advocated by the Web services specifications are that they imply that the execution engine has full visibility over the log, that is, over every single executed action. It is not only inefficient with respect to time and space, but also does not preserve privacy at all. For a composite service, the providers of its component services are different. As such, their privacy/security constraints may prevent them from exposing (part of) the logs corresponding to the execution at their sites. Also, heterogeneity may lead to the logs being maintained in different formats, rendering some of them incomprehensible. Our approach is targeted towards a more heterogeneous and autonomous environment where it may not be possible to log every action, and even if possible, the log itself may be considered confidential to be exposed in its entirety. Basically both the composition schema and execution log of a service may need to be kept private as they contain sensitive design and run time information about that service. We thus consider two problems: (i) finding a set of actions to log as small as possible such that compensation can be performed (referred to as the minimal compensable set), and (ii) perform-
ing the necessary compensation when a failure occurs based on the partial log exchanging as little information as possible. In our testing, we found that only about one fifth of the actions need to be visible [6, 5, 4].

In our work, we differentiate between two types of logs: execution log and data log. In ActiveBPEL, for instance, an AeCompInfo object basically contains (i) a reference to the scope that completed, and (ii) a variable snapshot object to record the state of all of its variables. We are mainly interested in the execution log, as visibility over it is needed to determine the execution sequence. The list of executed scopes would need to be reconstructed from the visible scope references. And, the problem is then to determine a minimal set of scopes whose references need to be visible, based on which we can always determine the complete list of executed scopes. While the second type (that is, data log) is also important for compensation, we do not deal with this log directly in our work. We assume that any such required data is logged properly, is visible, and can be used by the compensating services as and when required.

Formally, we describe the problem in Sections 2 and 3 and model the composition service schema of a service that specifies the execution order of its components, as hierarchical graphs (see Section 4), performing actions on global variables. We do not tackle here the modelization of a service as a graph, which should be handled with care to yield a graph of reasonable size (see [15] and Example 1). Dealing with few visible actions and with a strict privacy preserving policy makes compensation harder to perform, as only a limited amount of information can be exchanged between a service and its components to reconstruct what happened without any ambiguity (see Section 5). The other problem of selecting a compensable set of actions of minimal size is related to sensor selection problems [16, 10]. It makes our problem NP-complete ([6, 4]) and Section 3). We nevertheless design a hierarchical algorithm that computes a minimal compensable set while preserving privacy in Section 6. We show that it is efficient performance wise and can be used with real life composite services in Section 7. Finally, we extend our algorithm to components with several inputs and outputs, allowing e.g. calling parameters and fault handling (see Section 8). Proofs are omitted due to lack of space, but can be found in [4].

2 Formalizing Compensability

Formally, we define a composite service as a 4-tuple Finite State Machine (FSM) \( M = (Q, s_0, s_f, T) \), where \((Q, T)\) is a graph (\(q \in Q\) is called a state and \(t \in T\) 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 \( \rho = \tau_1 \cdots \tau_n \in T^* \) is a path of \( M \) if there exists \( q_0, \cdots, q_n \in Q^{n+1} \) with \( \tau_i = (q_{i-1}, q_i) \) for all \( 1 \leq i \leq n \). A path is called initial if furthermore \( q_0 = s_0 \). We denote by \( P(q_1, q_2) \) the set of paths between a pair of states \( q_1 \neq q_2 \), and by \( P(M) \) 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 \( T_O \subseteq T \) visible to \( M \). For an execution sequence \( \rho \) of \( M \), we call visibility projection the execution visibility we have after \( \rho \) was executed. We say that a visibility projection \( \sigma \) 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 \( M = \{Q, s_0, s_f, T\} \), let \( T_O \subseteq T \) be the set of visible transitions. The visibility projection \( \text{Obs}_O : T^* \rightarrow T_O^* \) is the morphism with \( \text{Obs}_O(a_1 \cdots a_n) = o_1 \cdots o_n \) with \( o_i = a_i \) if \( a_i \in T_O \), and \( o_i = \epsilon \) if \( a_i \in T \setminus T_O \), with \( \epsilon \) the empty word.

That is, \( \text{Obs}_O(\rho) \) is the subsequence of \( \rho \) obtained by eliminating from \( \rho \) every occurrence of a transition that is not in \( T_O \). With such a visibility projection \( \text{Obs}_O \), 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 \( \tau \notin T_O \). Then, \( \text{Obs}_O(\rho \tau) = \text{Obs}_O(\rho) \). A usual way to overcome such a problem is to ask for certainty only up to the last few transitions of the sequence [12]. 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 \( M = \{Q, s_0, s_f, T\} \) be a service, \( T_O \subseteq T \). The visibility projection \( \text{Obs}_O^{\text{last}} : T^* \rightarrow (T_O^*, Q) \) is the function \( \text{Obs}_O^{\text{last}}(\rho) = (\text{Obs}_O(\rho), q) \) for all \( \rho \in P(M) \) ending in \( q \).

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

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 \( M = \{Q, s_0, s_f, T\} \), we would like to determine a minimal set of transitions \( T_O \subseteq T \) which needs to be visible such that the system is still compensable.
The cardinality of such a minimal compensable set $T_O$ of a service $M$ is referred to as its compensable size $M_0(M) = |T_O|$. 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.

**Example 1.** We consider in Fig. 1 a travel funds request service, inspired by the workflow in [13]. It involves different departments across organizations, and it is hierarchical in that the deliver cheque service is hierarchically described.

![Diagram of a travel funds service workflow](image)

**Fig. 1.** (Hierarchical) Travel Funds service

We model the service using the FSM $M = (S, s_0, s_f, T)$, as shown in Fig. 2. Note that the FSM representation is a simplification since for instance the choice between the Team Leader or Supervisor approvals is not represented. The reason is that they are both associated with an empty compensating service, hence knowing which was chosen is not necessary to be able to perform compensation. However, it is necessary to know which bank issued the cheque in order to be able to compensate it by a “Cancel American Express (Citibank) Cheque”. As mentioned earlier, we assume that each service logs any data which might be required for its compensation (e.g., cheque amount and account number), and that its compensating service has visibility over any such data logs. Now, let $T_O = \{e_2, e_3, e_9\}$ and a failure occurs while processing $e_8$, that is, the cheque is not issued or delivered correctly. Then, $\text{Obs}^{\text{last}}(e_1e_2e_5e_7) = (e_2, s_5) =
Thus, we do not know if an American Express or Citibank cheque was processed. With $T'_O = \{e_2, e_6, e_9\}$, we have $\text{Obs}_{O}^{last}(e_1e_2e_4e_6e_7) = (e_2, s_5) \neq \text{Obs}_{O}^{last}(e_1e_2e_4e_6e_7) = (e_2e_6, s_5) \neq \text{Obs}_{O}^{last}(e_1e_2e_4e_6) = (e_2e_6, s_4)$, and $T'_O$ is a compensable set of transitions.

3 A Compensation Algorithm

We first relate the problem of computing a minimal compensable set with other known problems. We state now that computing a minimal compensable set is equivalent to the uniconnected subgraph problem [8], also called the minimal marker placement problem [10].

Let $M = \{Q, s_0, s_f, T\}$ be a service and $T_O$ a subset of transitions of $M$. Denote by $M' = \{Q, s_0, s_f, T \setminus T_O\}$ the service $M$ obtained by deleting all transitions belonging to $T_O$. Then, $T_O$ 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.

First, observe that 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 $q_2$.

Then, 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 $(\sigma, q_{n+1})$, we can reconstruct in a unique way the path $\rho$ of $M$ with $\text{Obs}_{O}^{last}(\rho) = (\sigma, q_{n+1})$ using the algorithm in Fig. 3.

First, the compensation algorithm runs in quadratic time, but it requires full visibility over both the logs and the composition schemas of all the composite services. Furthermore, this compensation algorithm can be used only when a compensable set has already been defined. Finding the smallest such set is actually an NP-complete problem. Indeed, we know from [10] that the minimal marker placement problem is NP-complete even for acyclic graphs. We also
Fig. 3. Compensation Algorithm

**Input:** Visibility projection \((\sigma, q_{n+1})\).

**Output:** The unique path \(\rho\) of \(M\) with \(\text{Obs}^{\text{out}}(\rho) = (\sigma, q_{n+1})\).

**Initialization:** Set \(\rho := \epsilon\), current state \(s := s_0\).

if \((n = 0)\) (no transitions logged)
set \(\rho\) to the unique path connecting \(s\) to \(q_{n+1}\) and return \(\rho\);
else
    for \(i = 1 \ldots n\) do
        if \((\tau_i\) is an outgoing transition of \(s\))
            append \(\tau_i\) to \(\rho\);
        else
            determine the unique path \(\rho_1\) of \(M'\) connecting \(s\) to \(\tau_i\);
            append \(\rho_1\tau_i\) to \(\rho\);
            set \(s := \tau_i^*\);
        if \((s = \tau_n^* = q_{n+1})\)
            return \(\rho\);
    else
        determine the unique path \(\rho_1\) connecting \(s\) to \(q_{n+1}\);
        append \(\rho_1\) to \(\rho\) and return \(\rho\);

Note that the unique path connecting \(s\) to \(\tau_i\) can be determined in linear time by a simple DFS of \(M'\) starting from \(s\).

showed in [6] 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).

A simple algorithm to compute the minimal compensable set is to enumerate every subset of \(T\) and check the ones which are compensable. Such an algorithm would take time \(O(2^{|M|})\) and need to have visibility over the whole composition schema of the service to work.

We propose now to consider the hierarchical structure of the composite services to preserve privacy, handle heterogeneity, and to reduce the complexity. Compared to [6], the algorithm is different as it reduces complexity even further (see Section 7), and more importantly can be used to preserve privacy and handle heterogeneity. Moreover, we extend the algorithm to components with more than one entry and one exit points. The polynomial time (non hierarchical) algorithms we proposed in [5] are basically heuristical algorithms to find some overapproximation of the minimal compensable set. While they also needed visibility over the whole composition schema to work, they can be used in conjunction with the present methodology to preserve privacy.
4 Hierarchical Services

Hierarchical services provide an efficient way to model large and complex services by allowing a modular decomposition. We consider hierarchical services where two transitions (supertransitions) can be further refined into another service.

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

- $(Q^i, T^i)$ is a finite graph,
- $s_0^i$ and $s_f^i$ are the initial and final states respectively, and
- $\tau_1^i, \tau_2^i \in T^i \cup \{\epsilon\}$ are two supertransitions representing FSMs $M_{k_1}^i, M_{k_2}^i$ respectively, with $k_1^i, k_2^i > 1$.

For instance, the workflow in Fig. 1 can be described by a hierarchical service $\langle M_1, M_2 \rangle$, where $M_2$ is made of an initial and final state, and two transitions $e_8, e_9$ from the initial to the final state. The service $M_1$ is very similar to Fig. 2, except that there is a unique transition $e_{10}$ between $s_5$ and $s_f$ instead of two. This is a supertransition $(\tau_1^1, k_1^1)$, with $\tau_1^1 = e_{10}$ and $k_1^1 = 2$, meaning that $e_{10}$ represents $M_2$.

With each hierarchical FSM $H$, we associate an ordinary FSM $H$ obtained by taking $M^i$, and recursively substituting each supertransition $\tau_1^i$ by the FSM $M_{k_1}^i$ it represents. Let $\tau_1^i = (s_1, s_2)$ and $M_{k_1}^i = \{Q, s'_1, s'_2, T\}$, then on substituting $\tau_1^i$ of $M^i$ by $M_{k_1}^i$, we have $M^i = \{Q', s'_0, s'_f, T'\}$ 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 $s'_f = s_f^i$
- $T' = T^i \setminus \{(s_1, s_2)\} \cup T \cup \mathcal{I}$, where $\mathcal{I} = \{(q, s'_y) \mid (q, s_y) \in T^i \land y \in \{1, 2\}\} \cup \{(s'_y, q) \mid (s_y, q) \in T^i \land y \in \{1, 2\}\}$.

For Example 1, $H$ is depicted in Fig. 2. Given a hierarchical service $\langle H_n \rangle$, $H_j$ is a component of $H_0$, if there is a supertransition $(t, j)$ in $H_i$. 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 $H$. The diameter $||H||$ of $H$ can be exponential in the size of $H$, because 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).

Now, let us define a hierarchical service $H$ having two levels. The top level $H_1$ has two states, one initial and one final, with two transitions $\tau_1, \tau_2$ from the initial to the final state. Transition $\tau_2$ is a supertransition. It is not easy to determine a minimal compensable set for $H$. Consider first that $\tau_2$ describes a system $H_2$ similar to $H_1$, that is two transitions $\tau_3, \tau_4$ from the initial to the final state, but without supertransitions. The set $T_2 = \{\tau_3\}$ is a minimal compensable set of $H_2$. Now, looking at $H_1$ as a normal system (without supertransitions),
\[ T_1 = \{ \tau_1 \} \] is also a minimal compensable set of \( H_1 \). We have furthermore that \( T_1 \cup T_2 \) is a minimal compensable set of \( H \).

However, if we take \( H_2 \) to be the system described in Fig. 2 and the associated minimal compensable set \( T'_2 = T'_O = \{ e_2, e_6, e_9 \} \) of transitions described in Example 1, then \( T_1 \cup T'_2 \) is not a minimal compensable set of \( H \). The reason is that \( T'_2 \) is already a compensable set of transitions, because all paths that pass through \( H_2 \) use at least one transition in \( T'_O \), so they can be differentiated from the path \( \tau_1 \). That is, the fact that a subset of transitions is a minimal compensable set of transitions is global to the whole graph, not local.

In the following, we present a divide and conquer algorithm that uses the hierarchical structure to compute a minimal compensable set.

We define a component service as follows:

**Definition 5 (Component FSM).** An FSM \( C = (Q', s'_0, s'_f, T') \) is a component of \( M = (Q, s_0, s_f, T) \) if

- \( Q' \subseteq Q \) and \( T' \subseteq T \),
- \( \forall q \in Q \setminus Q', q' \in Q' \), we have \( (q, q') \in T \) or \( (q', q) \in T \) implies \( q' \in \{ s'_0, s'_f \} \).

For example, the FSM \( M \) in Fig. 4 has a component \( C = (\{ s'_0, s'_f \}, s'_0, s'_f, \{ c, d, e, f, g \}) \).

![Fig. 4. A component C of service M depicted inside the dotted oval](image)

We now explain how to design a hierarchical compensation algorithm, preserving the privacy of logs and of services.

## 5 Privacy Preserving Compensation

We discuss the interactions required to reconstruct the actual execution sequence from the log (more formally form the visibility projection) in the event of a failure, and performing the actual compensation. Without global visibility, the logs or visibility projections are maintained locally by each service, and are visible only to that service. Other services, including even their parents or children, do not have direct visibility over the logs. Given this, we need some mechanism to synchronize the logs of at least parent-child services. For example, let us consider the FSM \( M \) in Fig. 4 with component \( C = (\{ s'_0, s'_f \}, s'_0, s'_f, \{ c, d, e, f, g \}) \) and
a minimal compensable set \( \{c, d, e, j\} \) visible. Then, for the execution sequence \( abcdehibcghjbf ehibcghibf ehk \), the local visibility projections at \( M \) and \( C \) would be \( j \) and \( cdecece \) respectively. With these visibility projections, we know that \( C \) was invoked five times, but we need some synchronization mechanism to know how many times it was invoked before \( j \)'s execution, and after its execution. To overcome this, we assume the use of global timestamps to synchronize parent-child logs (alternate strategies are discussed later).

First, we give the hierarchical compensation algorithm in Fig. 5 for an FSM \( M \) containing component \( C \). Note that \( M \) itself may be a component of a higher level FSM. It queries the number of times a component \( C \) was called between two transitions (actually between their timestamps). Once the execution sequence \( \rho \) of \( M \) has been computed, \( M \) compensates its executed transitions in the reverse order. For each invocation of component \( C \) (transition \( s'_0, s'_f \) in \( \rho \)), 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.

If global timestamps are infeasible, an alternate strategy would be as follows: The FSM \( M \), in addition to logging the transitions in \( T_{O1} \), also inserts a special marker (say \( X \)) in its local log each time it invokes component \( C \). For example, for the FSM \( M \) in Fig. 4 with compensable set \( \{c, d, e, j\} \) visible, the execution sequence \( abcd ehibcghjbf ehibcghibf ehk \) would lead to logging \( XX jX X X \) at \( M \). Note that this is equivalent to having \( (s'_0, s'_f) \) visible in \( M' \), and clearly, if we delete \( (s'_0, s'_f) \) from \( M' \), then there does not exist greater than one path between any pair of states in \( M' \).

The use of markers leads to some redundancy, and as our goal is clearly to minimize logging, we give another strategy below 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 the example scenario (Fig. 4), it would lead to the log \( XjX \) at \( M \). With logs \( XjX \) at \( M \) and \( cdecece \) at \( C \), we still do not know how many times \( C \) was invoked before and after the execution of \( j \), and hence cannot reconstruct the execution sequence. To overcome this, 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). For our example scenario, this would lead to the logs \( XjX \) and \( Xdecece \). 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 2. We give a sample run of the hierarchical compensation algorithm on the hierarchical FSM in Fig. 6. The hierarchical FSM in Fig. 6 is basically a simple extension of the workflow logic in Fig. 2 (levels 2 and 3 correspond
while (s_{\text{curr}} \neq q_{n+1}) do
begin
  if (\mathcal{P}_{M'}(s_{\text{curr}}, s_{\text{next}}) = \{\rho_2\}) (exactly one path between s_{\text{curr}}, s_{\text{next}})
  set \rho_1 := \rho_2;
  if (\mathcal{P}_{M'}(s_{\text{curr}}, s_{\text{next}}) = \{\rho_2, \rho_3\}), where \rho_3 contains (s'_0, s'_1)
  if C was invoked between s_{\text{curr}}, s_{\text{next}} (ask C)
  set \rho_1 := \rho_3;
  else set \rho_1 := \rho_2;
  if (|\mathcal{P}_{M'}(s_{\text{curr}}, s_{\text{next}})| > 2) (implies cycle s_{\text{curr}} \xrightarrow{\rho_5} s_{\text{curr}} \xrightarrow{\rho_5} \ldots (s'_0, s'_1) \xrightarrow{\rho_5} s'_0 \rightarrow s_{\text{next}})
  Determine the alternate path \rho_7 connecting s'_j to s_{\text{next}};
  Ask the number m of times C was invoked between s_{\text{curr}}, s_{\text{next}};
  if (m = 0)
  set \rho_1 the unique path from s_{\text{curr}} to s_{\text{next}}
  which does not contain (s'_0, s'_j);
  if (m = 1)
  set \rho_1 := \rho_4\rho_5(s'_0, s'_j)\rho_7;
  if (m > 1)
  set \rho_1 := \rho_4\rho_5(s'_0, s'_j)\rho_6\ldots|\rho_m(s'_0, s'_j)\rho_6|_{m-1}\rho_5(s'_0, s'_j)\rho_7;
  Append \rho_1 to \rho;
  if (n \neq 0)
  append \tau_n to \rho;
  if (s_{\text{next}} = q_{n+1})
  set s_{\text{curr}} := q_{n+1} (to terminate);
  else set s_{\text{curr}} := \tau_1;
  if (i < n)
  increment i and set s_{\text{next}} := \tau_i;
  else set s_{\text{next}} := q_{n+1};
end;

Fig. 5. Compensation Algorithm under Limited Visibility

Input. Visibility projection (\sigma = \tau_1\tau_2\ldots\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 \rho to be compensated with Obs^{\mathcal{M}}(\rho) = (\sigma, q_{n+1}).

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

M' designates M where the visible transitions T_O were deleted and the component C with states Q' and transitions 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, T \setminus (T' \cup T_O) \cup \{(s'_0, s'_f)\}).
to the Travel Funds and Deliver Cheque services, respectively). We refer to the FSMs at levels 1, 2 and 3 as $M, M^1$ and $M^2$, respectively. Let the assigned minimal compensable sets of $M, M^1$ and $M^2$ be $T_O = \emptyset$, $T_{O1} = \{e_{11}, e_{15}\}$ and $T_{O2} = \{e_{23}\}$, respectively. The visible transitions are denoted in Fig. 6 by dashed arrows.

![Sample hierarchical system (visible transitions denoted by dashed arrows)](image)

**Fig. 6.** Sample hierarchical system (visible transitions denoted by dashed arrows)

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 $M, M^1$ and $M^2$ are $s_3$ (the state before failure), $e_{11}$ and empty, respectively. Given this, $M$ starts the execution sequence reconstruction. There exist 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 $M^1$ corresponding to $e_2$ is asked to check the number of times it was invoked between the start time and logging time of $s_3$. $M^1$ checks its log, finds $e_{11}$ logged during that time, and replies that it was invoked once. So, the execution sequence at $M$ is set to $e_1 e_2$, and $M$ starts compensation in the reverse order. As the first transition to be compensated $e_2$ is a supertransition, the corresponding component $M^1$ is asked to compensate its execution first. Before it can compensate, $M^1$ 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 \( \rho \). Then, it checks for paths between \( e_{11}^1 = s_{12} \) and its final state \( s_{16} \). Recall that it is checking for paths in the FSM \( M^1 \) 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 \( \rho \), leading to the execution sequence \( e_{11}e_{14}e_{16}e_{17} \). With the execution sequence determined, \( M^1 \) 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 \( M^2 \) to compensate its execution first. For \( M^2 \), 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 transitions in the determined execution sequences.

We now present a hierarchical algorithm to compute a minimal compensable set, while preserving privacy.

6 Privacy Preserving Computation of a Minimal Compensable Set

We first define properties of a set of transitions that will be important to know whether it is a minimal compensable set. We say that a path \( \rho = \tau_i|_{i=1\ldots n} \) passes through an FSM \( M = (Q, s_0, s_f, T) \) if \( \exists \tau_i, \tau_i \in T \). We say that a path \( \rho = \tau_i|_{i=1\ldots n} \) belongs to an FSM \( M = (Q, s_0, s_f, T) \) if \( \forall \tau_i, \tau_i \in T \). We say that a path \( \rho = \tau_i|_{i=1\ldots n} \) does not touch an FSM \( M = (Q, s_0, s_f, T) \) if \( \forall \tau_i, \tau_i \notin T \).

Further, for an FSM \( M = (Q, s_0, s_f, T) \) and subset of transitions \( T_O \subseteq T \), we define the following predicates: A path \( \rho \) is referred to as an invisible path if it does not use any transitions of \( T_O \).

- \( P_0(M, T_O) \) holds if there does not exist more than one invisible path between any two states \( s_1 \neq s_2 \in Q \) (\( T_O \) is a compensable set of transitions).
- \( P_1(M, T_O) \) holds if (i) \( P_0(M, T_O) \) 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 the enclosing \( M \), 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. 7(a).
- \( P_1'(M, T_O) \) holds if (i) \( P_0(M, T_O) \) holds, and (ii) there do not exist states \( s_1, s_2 \in Q \), such that:
  - 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 \( M \), if there exists a path from the final to initial state of \( C \) that does not touch \( C \) as shown in Fig. 7(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 $s'_0$).

![Diagram](image)

**Fig. 7.** Significance of $P_1(M, T_O)$ and $P_{1'}(M, T_O)$

By definition, $P_{1'}(M, T_O) \Rightarrow P_1(M, T_O) \Rightarrow P_0(M, T_O)$, since for all $s$, there always exists a path from $s$ to $s$. Let $\epsilon < 0 < 1 < 1'$, we define $\text{Best}(M, T_O) = x \in \{\epsilon, 0, 1, 1'\}$, such that $P_x(M, T_O)$ holds but not $P_y(M, T_O)$ with $y > x$, with the convention $P_\epsilon(M, T_O)$ is always true. Informally, Best refers to the best properties a given set of transitions can ensure, if visible. For instance, in Fig. 2 with $T_O = \{e_2, e_6, e_9\}$, $P_0(T_O)$ holds because $T_O$ is compensable, $P_1(T_O)$ holds because every path from $s_0$ to $s_f$ uses at least one transition of $T_O$, but $P_{1'}(T_O)$ does not hold as there exists three invisible paths: $e_4$ from $s_2$ to $s_4$; $e_5e_7e_8$ from $s_2$ to $s_f$. Thus, $\text{Best}(M, T_O = \{e_2, e_6, e_9\}) = 1$.

Based on the above properties, we can define the following hierarchical algorithm to compute a minimal compensable set. It is based on replacing a component $C$ (potentially big) by a component $D$ of fixed size. We formalize here the replacement.

Given a component $C = (Q', s'_0, s'_f, T')$ of $M = (Q, s_0, s_f, T)$ and $D = (Q'', s''_0, s''_f, T'')$, we denote by $M_C(D)$ the FSM obtained on substituting $C$ by $D$, that is, $M_C(D) = (\bar{Q}, \bar{s}_0, \bar{s}_f, \bar{T})$ 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 $\bar{s}_f = s''_f$,
- $\bar{T} = T \setminus T' \cup T'' \cup I$, where $I = \{(q, s''_y) \mid (q, s'_y) \in T \land y \in \{0, f\}\} \cup \{(s''_y, q) \mid (s'_y, q) \in T \land y \in \{0, f\}\}$.

14
For $y \in \{0,1,1'\}$ and a component $C$, we denote by $T_y(C)$ a set of size minimal such that $P(C, T_y(C)) = y$. We are now ready to give a hierarchical algorithm to compute a minimal compensable set.

**Algorithm CompensableSet($M$):**

1. Ask a component $C$ of $M$ to give a FSM $D$ such that $|T_1(C)| - |T_0(C)| = |T_1(D)| - |T_0(D)|$ and $\min(1, |T_1(C)| - |T_1(C)|) = |T_1(D)| - |T_1(D)|$;

2. Compute a minimal compensable set $T$ of $M_C(D)$ and let $y = \text{Best}(D, T \cap D)$;

3. If $y \neq 1'$ or $|T_1(C)| - |T_1(C)| \leq 1$, then set $T \setminus D$ as visible, and inform $C$ to choose a set $T'$ with $\text{Best}(C, T') = y$.

4. Else, set $T \cup \{x\} \setminus D$ as visible, and inform $C$ to choose a set $T'$ with $\text{Best}(C, T') = 1$, where $x$ is any action on the unique invisible path of $M$ from the final state of $C$ to the initial state of $C$.

**Remarks**

- It is easy to see that there are 4 possible $D$ that cover all the cases of Step 1. They need at most 6 transitions. For a detailed explanation, the interested reader is referred to [4].

- A set $T'$ with $\text{Best}(C, T') = y$ can be computed using the MinimalDecomposition algorithm presented in [6].

- The algorithm can handle as many components of $M$ as needed, since it just performs an induction on $M_C(D)$.

- More importantly, this algorithm is **privacy preserving** since $M$ does not need to know anything about $C$, except the type of $C$ among the four possibilities (the choice of $D$).

**Example 3.** We give a sample run of the CompensableSet algorithm on the hierarchical system in Fig. 6. As in Example. 2, we refer to the FSMs at levels 1,2 and 3 as $M, M^1$ and $M^2$, respectively. To start with (in a bottom-up fashion), $M^2$ computes $x \in \{0,1,1'\}$, $MO_x(M^2)$, leading to the differences $M_{O_1}(M^2) - MO(M^2) = MO_1(M^2) - MO(M^2) = 1$ to $M^1$. Then, it sends $D = \{(s_0', s_2'), s_0, s_1', \{(s_0', s_1')\}\}$ satisfying the characteristics in Step 1, to $M^1$. $M^1$ substitutes $e_{17}$ by $D$, and let the substituted FSM be $M'$. $M'$ then computes $x \in \{0,1,1'\}$, $MO_x(M')$, leading to the differences $M_{O_1}(M') - MO(M') = 1$ and $MO_1(M') - MO(M') = 0$. Based on the difference values, $M^2$ sends to $M$ the FSM $E = (Q'', s_0'', s_1'', s_2'', T'')$, where $Q'' = \{s_0'', s_1'', s_2'', s_3''\}$ and $T'' = \{(s_0'', s_1''), (s_2'', s_1''), (s_1'', s_3''), (s_2'', s_3'')\}$. On receiving this, $M$ substitutes $e_2$ by the FSM $E$, and let the substituted FSM be $M''$.

Once the substitutions have been performed, the next step is to assign the compensable sets, which proceeds in top-down order. $T_O = \{(s_0'', s_1''), (s_2'', s_3'')\}$ is a minimal compensable set of $M''$. Given this, $M$ does not need to assign
any of its transitions as visible. As $\text{Best}(E, \{(s_0'', s_1'''), (s_2'', s_3''')\}) = 1$, then $M^1$ needs to assign a minimal set $T_{O^1}$ as visible, such that $\text{Best}(M^1, T_{O^1}) = 1$, say $T_{O^1} = \{e_{11}, e_{15}\}$. On the same lines, with only the transitions in $T_{O^1}$ of $M^1$ visible, none of $D$'s transitions are visible, as such $\text{Best}(D, \emptyset) = 0$. Then, $M^2$ only needs to assign a minimal set $T_{O^2}$ as visible, such that $\text{Best}(M^2, T_{O^2}) = 0$, say $T_{O^2} = \{e_{23}\}$. The above computations lead to the visible transitions depicted by dashed arrows in Fig. 6.

7 Complexity

The next theorem states the complexity of the above algorithm in the general case of hierarchical systems.

**Theorem 1.** Let $H = (M_i)_{i=1}^n$ be a hierarchical service. It takes at most time $O(2^{\sum_{i}|M_i|})$ for each component to compute a set of compensable transitions $T_{O_i}$, such that $\bigcup_i T_{O_i}$ is a minimal compensable set of $H$.

It is important to notice that a small hierarchical graph can represent an exponentially bigger flat graph, hence the worse case complexity of applying the brute force method to a flattened version of $H = (M_i)_{i=1}^n$ is $2^{\sum_{i}|M_i|}$. We proved that using the hierarchical algorithm, the complexity is linear in the number of hierarchy level, 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. For instance, if components are of size 6 at most, then our method would take $64nC$ unit of time, with $C$ a constant, while it could be doubly exponential in $n$ using the brute force method (one exponential due to the reuse of components and another due to decomposition).

![Fig. 8. Execution time vs. Number of subcomponents/edges](image)
We tested the CompensableSet 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 [14]. 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 expand the hierarchical graphs as (flat) graphs, whose size is linear in the number of base components. We then run both a brute force algorithm and our hierarchical algorithms on these graphs. We do not input the hierarchical structure of the graph, instead it uses the polynomial time algorithm presented in [4] to determine the hierarchical structure.

Fig. 8 shows the times in logarithmic scale needed to compute a minimal compensable set using different algorithms. In Fig. 8, Decomposition 1 refers MinimalDecomposition algorithm presented in [6]. It is a hierarchical algorithm, but it does not preserve privacy as it needs to have full visibility over its components. Decomposition 2 refers to the CompensableSet algorithm presented in Section 6. The Decomposition 1 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 1 algorithm on an average takes 17s for 106 edges. However, it is still not polynomial time with respect to the number of base subcomponents/number of edges (on an average, 0.17s for 18 edges and 17.6 for 106 edges). Now, let us consider the test timings of the Decomposition 2 algorithm. The algorithm is indeed linear time with respect to the number of base subcomponents/number of edges (on an average, 0.14s for 18 edges and 0.73s for 108 edges).

In general, with respect to the ratio of edges required to be visible among all the edges, the ratio ranges from 1 out of 4 edges to 1 out of 15 edges [6]. The mean value seems to tend to 1 out of 6 edges. This implies that logging only (execution details of) the visible transitions can lead to significant space savings.

8 Complex Components

In this section, we extend the hierarchical algorithm to complex components, that is, FSMs having more than one initial and/or final states. This kind of components are very useful to model services with calling parameters, or fault/exception handling for instance. We refer to initial and final states as port states in general, in this section. For example, the service \( M = (Q, s_0, s_f, T) \) in Fig. 9 has a complex component \( C' = (Q', s'_0, s'_f, T') \) with two final states \( s'_1 \) and \( s'_2 \).

We consider services \( M = (Q, s_0, \ldots, s_0, s_{f_1}, \ldots, s_{f_2}, T) \) having a set \( P \) of \( b = b_1 + b_2 \) port states, consisting of \( b_1 \) initial and \( b_2 \) final states. Let \( T_O \) be a subset of transitions of \( M \). We define \( b^2 + 1 \) predicates \( P_0(M, T_O), P_{p_1, p_2}(M, T_O) \) for all \( p_1, p_2 \in P \).
\[ P_0(M, T_O) \] holds if there does not exist more than one invisible path between any two states \( s_1 \neq s_2 \in Q \) (\( T_O \) is a compensable set of transitions).

- if \( p_1 \) is an initial and \( p_2 \) is a final state: \( P_{p_1, p_2}(M, T_O) \) holds if (i) \( P_0(M, T_O) \) holds, and (ii) there does not exist an invisible path from \( p_1 \) to \( p_2 \). The significance of \( P_{p_1, p_2}(M, T_O) \) is analogous to that of predicate \( P_1(M, T_O) \) for simple components.

- if \( p_1 \) is a final and \( p_2 \) an initial state: \( P_{p_1, p_2}(M, T_O) \) holds if (i) \( P_0(M, T_O) \) holds, and (ii) there do not exist states \( s_1, s_2 \in Q \), such that (a) there is an invisible path from \( p_2 \) to \( s_2 \), (b) there is an invisible path from \( s_1 \) to \( p_1 \), and (c) there is an invisible path from \( s_1 \) to \( s_2 \). The significance of \( P_{p_1, p_2}(M, T_O) \) is analogous to that of predicate \( P_1(M, T_O) \) for simple components.

- if \( p_1, p_2 \) are two initial states: \( P_{p_1, p_2}(M, T_O) \) holds if (i) \( P_0(M, T_O) \) holds, and (ii) there does not exist a state \( s \) in \( M \) with invisible paths from both \( p_1 \) and \( p_2 \) to \( s \). If \( P_{p_1, p_2}(C, T_O) \) does not hold for a component \( C \) with visible set of transitions \( T_O \), then it might be a problem for the compensability of the enclosing \( M \) if there exists a state \( s_1 \) of \( M \) with invisible paths to both \( p_1 \) and \( p_2 \) as shown in Fig. 10(a).

- if \( p_1, p_2 \) are two final states, \( P_{p_1, p_2}(M, T_O) \) holds if (i) \( P_0(M, T_O) \) holds, and (ii) there does not exist a state \( s \) in \( M \) with invisible paths from \( s \) to both \( p_1 \) and \( p_2 \). If \( P_{p_1, p_2}(C, T_O) \) does not hold for a component \( C \) with visible set of transitions \( T_O \), then it might be a problem for the compensability of the enclosing \( M \) if there exists a state \( s_1 \) of \( M \) with invisible paths from both \( p_1 \) and \( p_2 \) to \( s_1 \) as shown in Fig. 10(b).

Note that some predicates imply others as \( P_{p_1, p_2} = P_{p_2, p_1} \) for two initial or final states, and \( P_{p_1, p_2} \Rightarrow P_{p_2, p_1} \) for \( p_1 \) and \( p_2 \) a final and initial state respectively. However, we do not have a total order between predicates. That is, we define \( \text{Best}(M, T_O) : P^2 \rightarrow \{0, 1\} \) as a function with \( \text{Best}(M, T_O)(p_1, p_2) = 1 \) iff \( P_{p_1, p_2}(M, T_O) \). We can then extend algorithm CompensableSet, denoting \( T_{p_1, p_2}(C) \) for a set of minimal size such that \( P_{p_1, p_2}(C, T_{p_1, p_2}(C)) \) holds.

\[ \text{Fig. 9. Service } M = (Q, s_0, s_f, T') \text{ having complex component } C' = (Q', s'_0, s'_f, T') \]
Algorithm CompensableSet($M$):

1. Ask component $C$ of $M$ to give FSM $D$ such that for all $p_1, p_2 \in P$, $|T_{p_1,p_2}(C)| - |T_0(C)| = |T_{p_1,p_2}(D)| - |T_0(D)|$;

2. Compute a minimal compensable set $T$ of $M_C(D)$ and let $y = \text{Best}(D, T \cap D)$;

3. Set $T \setminus D$ as visible, and inform $C$ to choose a set $T'$ with $\text{Best}(C, T') = y$.

Obviously, this hierarchical algorithm still preserves privacy as for simple components. However, it may be difficult to find a small replacement FSM $D$ when the number of ports (input and output) is very large.

9 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 a compensation framework that is secure as well, and 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 services are implemented). Furthermore, we obtained as a byproduct much faster complexity. Such algorithms require only a minimal number of transitions to be logged, roughly one fifth of the transitions that have been executed, and can handle complex components.
References