Minimal Observability for Transactional Hierarchical Services *

Debmalya Biswas and Blaise Genest
IRISA/INRIA & CNRS, Campus de Beaulieu, 35042 Rennes Cedex, France
{firstname.lastname}@irisa.fr

Abstract

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

1. Introduction

An interesting problem for complex systems is to determine a minimal set of actions that needs to be observable such that a given property holds. Some of the properties studied in literature of discrete event systems are normality [6], observability [5], state observability [9], diagnosability [13], etc. Our system corresponds to a composite (workflow) Web service. A Web service [1] refers to an online service accessible via Internet standard protocols. A composite service, composed of already existing (component) services, combines the capabilities of its components to provide a new service. A service schema which specifies the execution order of its components, can be modeled as a graph, 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).

Our long-term objective is to provide a transactional framework for (composite) Web services. A transaction can be considered as a group of actions encapsulated by the operations Begin and Commit/Abort, having the following properties: Atomicity (A), Consistency (C), Isolation (I) and Durability (D). Here, we focus on the atomicity aspect, that is, either all the actions of a transaction are executed or none. In the event of a failure, atomicity is preserved by compensation [3, 14]. Compensation consists of executing the compensating actions, corresponding to each executed action of the failed process, in reverse order of the original execution. Many advanced transactional models have also been proposed, e.g. "semantic compensation" [14] which do not require any knowledge of the execution sequence. However, their application to more autonomous systems like Web Services has been limited, where the default compensation mechanism of the widely used Business Process Execution Language (BPEL) is to "execute the completed actions in reverse order". Thus, for compensation to be feasible, we need to reconstruct each executed action or the complete history of any execution. To achieve that, we maintain a log of the observable actions. In addition to the obvious space overhead of logging, the complete log may not always be accessible. For a composite service, the providers of its component services are different. As such, their privacy/security constraints may prevent them from exposing (part of) the logs corresponding to the execution at their sites. Also, heterogeneity may lead to the logs being maintained in different formats, rendering some of them incomprehensible. Existing Web services specifications to provide transactional guarantees, such as WS-Coordination, WS-AtomicTransaction and WS-BusinessActivity [16], are basically distributed agreement protocols which are based on the assumption that "all state transitions are reliably recorded" and can be compensated. Our approach is targeted towards a more heterogeneous environment where all transitions may not be observable. Hence, we want from a partial log of the observable actions to know with certainty the actual sequence of executed actions, to be able to compensate it.

Section 2 introduces the required formal preliminaries including the precise problem statement. Clearly, we are interested in logging the smallest number of actions possi-

*This work is supported by la Region Bretagne (CREATE ACTIVE-DOC) and ANR-06-MDCA-005 DOCFLOW.

ble. However, determining the minimal number of actions to log, such that any execution of a system is compensable, is NP-Complete. This is not very surprising, since the closely related sensor selection problems [17, 8] are also NP-Complete (see section 3). Also, the problem cannot be approximated [11] in polynomial time, which means that polynomial time algorithms cannot give a minimal set for all graphs, and that for many graphs, the observable set produced would be much bigger than the minimal set.

A complex service is often constructed hierarchically (see section 4), with some services at a high level corresponding to many composite services at a lower level. Each hierarchical level potentially describes the interactions at a different level of abstraction, e.g., the top level may describe the interactions between several providers, then the next level between services of a provider, and so on. Moreover, components can be reused in a hierarchical system, giving a compressed way to represent big systems. Hierarchical systems are often used for words [10], Finite State Machines [2], and even trees [7]. For words, e.g., hierarchical structures correspond to the LZ compression [10]. We show in section 5 how to use the hierarchical representation to compute efficiently a minimal observable set of transitions. The algorithm is not straightforward since the log of both minimal sets of actions of different components is not necessarily enough to recover the actual sequence of executed actions of the whole graph. One solution could be to resort to function summarization, but then only an overapproximation of the minimal set of actions would be obtained. Nevertheless, *we show that it suffices to run the algorithm with slightly different parameters on each component*. We thus obtain a divide and conquer algorithm. We present a theoretical complexity analysis which illustrates the benefit of our method (up to two exponentials better when using the full hierarchical representation and one exponential better by using the hierarchical representation even if components are used only once, compared to flattening the hierarchical graph), that is verified experimentally (section 6). Proofs and details omitted for lack of space can be found in [4].

2. Preliminaries

Formally, we model a transactional service as a 4-tuple $M = (Q, s_0, s_f, \mathcal{T})$, where (Q, 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.

Our systems are thus graphs with a unique input and output point, each node and arc corresponds to a state and transition, but we ignore the alphabet. We assume that the service M does not have any unreachable states and that all states can reach the final state s_f. For convenience, we also assume that there are no outgoing edges from s_f and no incoming edges to s_0.[1] We say that an execution sequence $\rho = \tau_1 \cdots \tau_n \in \mathcal{T}^*$ 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. Finally, we denote by $|M|$ the size of M, that is, its number of transitions.

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

Definition 1 *Let $\mathcal{T}_O \subseteq \mathcal{T}$ be the set of observable transitions. The observation projection $Obs_O : \mathcal{T}^* \longrightarrow \mathcal{T}_O^*$ is the morphism with $Obs_O(a_1 \dots a_n) = o_1 \cdots o_n$ with $o_i = a_i$ if $a_i \in \mathcal{T}_O$, and $o_i = \epsilon$ if $a_i \in \mathcal{T} \setminus \mathcal{T}_O$, with ϵ the empty word.*

That is, $\text{Obs}_O(\rho)$ is the subsequence of ρ obtained by eliminating from ρ every occurrence of a tuple which is not in $\mathcal{T}_O$. With such an observation projection Obs_O, the only way of having execution sequence detectability is to have every transition observable. Indeed, as soon as there exists even one non-observable transition, the service is not execution sequence detectable. Else, let us take a path $\rho\tau$ with the last transition $\tau \notin \mathcal{T}_O$. Then, $\text{Obs}_O(\rho\tau) = \text{Obs}_O(\rho)$. An usual way to overcome such a problem is to ask for certainty only up to the last few events of the sequence [9]. However, this workaround does not make sense in our framework since if we cannot compensate the very last action, then we cannot compensate any action at all. As such, we design a new observation mechanism, where the last control point reached before failure is monitored, even if the last action is not logged. In practice, it means that every state that is reached is monitored, and overwrite the previous state in a special memory buffer.

Definition 2 *Let M be a service, $\mathcal{T}_O \subseteq \mathcal{T}$. The observation 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.*

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

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

The cardinality of such a minimal observable set $\mathcal{T}_O$ of a service M is referred to as its observable size $MO(M) =$

[1] Notice that we could deal with a service without these requirements, but the proof would be more technical.

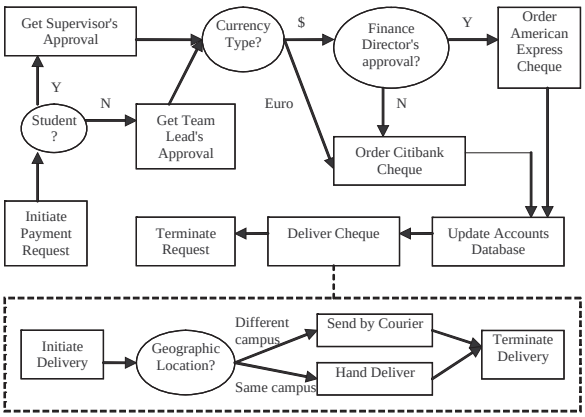

Figure 1. Travel funds request workflow.

$|\mathcal{T}_O|$. Notice that as is usual with decision and computation algorithms, it is sufficient to have an algorithm which from a service gives its observable size. That is, we can derive a minimal observable set of the service based on knowledge of its observable size in polynomial time.

Example 1 *We consider in Fig. 1 a travel funds request service, inspired by the workflow in [12]. It involves different departments across organizations, and it is hierarchical in that the deliver cheque service is hierarchically described.*

We model the service using the service $M = (S, s_0, s_f, \mathcal{T})$*, as shown in Fig. 2. Notice that this service 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 transition, hence knowing which path was taken here is not necessary to be able to perform recovery. However, it is necessary to know which bank issued the cheque in order to be able to compensate it, by a "Cancel Last American Express (Citibank) Cheque". Note that we do not exclude the logging of data values (in some persistent storage) required for compensation. For instance, if there wasn't any "Cancel Last Cheque" mechanism, then it would be needed to log the amount and account number associated with the ''Update Accounts Database" transition. Recovery would manually credit the amount of money written in the log to the corresponding account. Obviously, we cannot save on logging the data values, but we optimize the logging associated with the path visited. Our experiments performed on BPEL representations of some workflows reveal that one transition out of five is logged (which is confirmed in section 6) and that data values logs are small compared to logging the path.*

Now, let $\mathcal{T}_O = \{e_2, e_3, e_9\}$ *and a failure occurs while processing* e_8*, that is, the cheque is not issued or delivered correctly. Then,* $Obs_O^{last}(e_1e_2e_5e_7) = (e_2, s_5) = Obs_O^{last}(e_1e_2e_4e_6e_7)$*. Thus, we do not know if an American Express or Citibank cheque was processed. With* $\mathcal{T}'_O = \{e_2, e_6, e_9\}$*, we have* $Obs_O^{last}(e_1e_2e_5e_7) = (e_2, s_5) \neq Obs_O^{last}(e_1e_2e_4e_6e_7) = (e_2e_6, s_5) \neq Obs_O^{last}(e_1e_2e_4e_6) = (e_2e_6, s_4)$*, and* $\mathcal{T}'_O$ *is an observable set of transitions. Every path from* s_0 *to* s_f *uses at least one transition from* $\mathcal{T}'_O$*.*

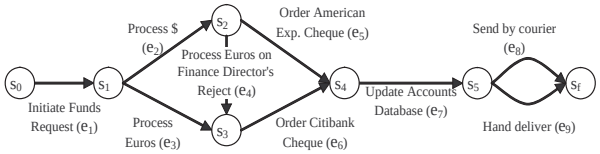

Figure 2. Modelization of Fig. 1.

3. Problem Hardness

We first relate the problem of computing $MO(M)$ using our definition of observable projections with other known problems. We state now that computing the minimal observable set is equivalent to the *uniconnected subgraph problem*, also called the *minimal marker placement problem* [8], in the meaning of the following proposition.

Proposition 1 *Let* M *be a service and* $\mathcal{T}_O$ *a subset of transitions of* M*. Denote by* M' *the service* M *obtained by deleting all transitions belonging to* $\mathcal{T}_O$*. Then,* $\mathcal{T}_O$ *is an observable set of* M *iff there does not exist a pair of paths* $\rho_1 \neq \rho_2$ *of* M' *with* ρ_1 *beginning and ending at the same states as* ρ_2*.*

To prove proposition 1, it suffices to prove that if there does not exist a pair of paths $\rho_1 \neq \rho_2$ of M' with ρ_1 beginning and ending at the same states as ρ_2, then from any observable projection (σ, q_{n+1}), we can reconstruct in a unique way a path with $\text{Obs}_O^{\text{last}}(\rho) = (\sigma, q_{n+1})$. The converse is trivial. Indeed, it suffices to define the only path ρ_i of M' between q'_i and q_{i+1} for $\sigma = (q_i, q'_i)_{i=1}^n$, and $i = 0 \cdots n$ (we fix $q'_0 = s_0$ the initial state of M', and recall that q_{n+1} is the last observed state). Then, the path $\rho = \rho_0(q_1, q'_1)\rho_1 \cdots (q_n, q'_n)\rho_n$ is the only path with $\pi_O^{\text{last}}(\rho) = (\sigma, q_{n+1})$. The search for each path ρ_i can be performed in linear time by a simple depth first search of M'.

The fact is that the marker placement problem is an NP-Complete problem. The question is then to know if there is a structural subclass of graphs which has a tractable algorithm to give the minimal observable size. We know from [8] 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 [8], but the encoding to get a unique starting and ending point is both easier to understand and allows a lower in and outdegree.

Theorem 1 *Let M be 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.*

This theorem does not mean that the problem is impossible to solve, but that it cannot be solved for all possible services. For instance, the complexity of the brute force method which generates every subset of transitions and tests whether it is observable, is $O(2^{|M|})$ for a service M with $|M|$ transitions. The question then is which structural property makes the problem easier to solve and often holds for (real life) composite services. We propose hierarchical services as a candidate property.

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. A hierarchical service 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,
- $\tau_1^i, \tau_2^i \in \mathcal{T}^i \cup \{\epsilon\}$ are two supertransitions representing services $M^{k_1^i}, M^{k_2^i}$ respectively, with $k_1^i, k_2^i > i$.

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 (τ_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 service H, we associate an ordinary service $\mathcal{H}$ obtained by taking M^i, and recursively substituting each supertransition by the service it represents. For example 1, $\mathcal{H}$ is depicted in Fig. 2. Given a hierarchical service $\langle H_n \rangle$, $\mathcal{H}_j$ is a component of $\mathcal{H}_i$, 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 $\mathcal{H}$. 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 system H with two levels. The top level H_1 has two states, one initial and one final, with two transitions τ_1, τ_2 from the initial to the final state. Transition τ_2 is a supertransition. It is not easy to determine a minimal set of transitions for H. Consider first that τ_2 describes a system H_2 similar to H_1, that is two transitions τ_3, τ_4 from the initial to the final state, but without supertransitions. The set $T_2 = \{\tau_3\}$ is a minimal observable set of transitions for H_2. Now, looking at H_1 as a normal system (without supertransitions), $T_1 = \{\tau_1\}$ is also a minimal observable set of transitions for H_1. We have furthermore that $T_1 \cup T_2$ is a minimal observable set of transitions for H.

However, if we take H_2' to be the system described in Fig. 2 and the associated minimal observable set $T_2' = \mathcal{T}_O' = \{e_2, e_6, e_9\}$ of transitions described in example 1, then $T_1 \cup T_2'$ is not minimal among the observable set of transitions for H. The reason is that T_2' is already an observable set of transitions, because all paths that pass through H_2 use at least one transition in $\mathcal{T}_O'$, so they can be differentiated from the path τ_1. That is, the fact that a subset of transitions is a minimal observable set of transitions is global to the whole graph, not local.

5. Algorithm for Minimal Observability

We turn now to defining an algorithm which uses the hierarchical structure of a complex service to compute the minimal observable set. First, we need the following notations. Given $\mathcal{T}_O$, a path ρ is said to be an unobserved path if it does not use any transition of $\mathcal{T}_O$. For a service M and a set of transitions $\mathcal{T}_O$ of M, we define the following predicates:

- $P_0(M, \mathcal{T}_O)$ holds if there does not exist more than one unobserved path between any two states $s_1 \neq s_2 \in Q$ ($\mathcal{T}_O$ is an observable 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 unobserved path from s_0 to s_f.
- $P_{1'}(M, \mathcal{T}_O)$ 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 (a) there is an unobserved path from s_0 to s_2, (b) there is an unobserved path from s_1 to s_f, and (c) there is an unobserved path from s_1 to s_2. We refer to such a combination of nodes and edges as an unobserved reverse cyclic pattern between s_1 and s_2 (within M) .

For instance, on Fig. 2 with $\mathcal{T}_O' = \{e_2, e_6, e_9\}$, $P_0(\mathcal{T}_O')$ holds because $\mathcal{T}_O'$ is observable, $P_1(\mathcal{T}_O')$ holds because every path from s_0 to s_f uses at least one transition of $\mathcal{T}_O'$,

but $P_1(\mathcal{T}'_O)$ does not hold since there exists three non observable paths: e_4 from s_2 to s_3/ e_1e_3 from s_0 to s_3/ $e_5e_7e_8$ from s_2 to s_f.

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 Best$(M, \mathcal{T}_O) = x \in \{\epsilon, 0, 1, 1'\}$ such that $P_x(M, \mathcal{T}_O)$ holds, but not $P_{xx}(M, \mathcal{T}_O)$ with $xx > x$, 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 observed.

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$(M, \mathcal{T}_O \cup \mathcal{T}_2)$.*

For $x \in \{0, 1, 1'\}$, we define $\mathcal{T}_x(M)$ as a smallest subset $\mathcal{T}_O$ of transitions of M such that $P_x(M, \mathcal{T}_O)$ holds. For a subset of transitions T of a component C of M, we also denote by $\mathcal{T}_x^{T,C}(M)$ a smallest set $\mathcal{T}_O$ such that $\mathcal{T}_O \cap C = T$ and $P_x(M, \mathcal{T}_O)$ holds. Every algorithm to compute the minimal observable set of transitions is recursive, taking the set of transitions considered observable as input. It is easy to modify them to input in the beginning not $\emptyset$ but T, and disallowing to select any new transitions in C, such that they compute $\mathcal{T}_x^{T,C}(M)$, and they do it faster than $\mathcal{T}_x(M)$ because they cannot choose among the transitions of C. As proved in proposition 2, the size of $\mathcal{T}_O$ is constant for several T such that $Best(C, T) = y$. If $|T'| > |T|$ with $Best(C, T) = Best(C, T')$, then $|\mathcal{T}_x^{T',C}(M)| > |\mathcal{T}_x^{T,C}(M)|$. We can use this idea to compute $\mathcal{T}_x(M)$ in a compositional manner, for a service M having component C:

MinimalDecomposition(M, C):

1. Compute a minimal set $\mathcal{T}_y(C)$ of transitions of C, $\forall y \in \{0, 1, 1'\}$.
2. Compute a minimal set $\mathcal{T}_0^{\mathcal{T}_y(C),C}(M)$ of transitions of M, $\forall y \in \{0, 1, 1'\}$.
3. Output a set of smallest size among $\mathcal{T}_0^{\mathcal{T}_y(C),C}(M)$.

For example, consider the service M having component C in Fig. 3.

1. A minimal set $\mathcal{T}_0(C) = \{(s'_0, s_2), (s_1, s'_f)\}$, $\mathcal{T}_1(C) = \{(s'_0, s_1), (s_2, s'_f)\}$, and $\mathcal{T}_{1'}(C) = \{(s'_0, s_2), (s_1, s'_f), (s_1, s_2)\}$.
2. The corresponding observable sets of M: $\mathcal{T}_0^{\mathcal{T}_0(C),C}(M) = \{(s'_0, s_2), (s_1, s'_f), (s_0, s_f)\}$ of size 3, $\mathcal{T}_0^{\mathcal{T}_1(C),C}(M) = \{(s'_0, s_1), (s_2, s'_f)\}$ of size 2, and $\mathcal{T}_0^{\mathcal{T}_{1'}(C),C}(M) = \{(s'_0, s_2), (s'_1, s_f), (s_1, s_2)\}$ of size 3.
3. $\mathcal{T}_0^{\mathcal{T}_1(C),C}(M)$ is a minimal observable set of M.

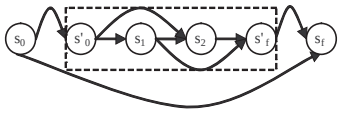

Figure 3. Service $M = (Q, s_0, s_f, \mathcal{T})$ having component $C = (Q', s'_0, s'_f, \mathcal{T}')$.

We can now state the main theorem of the paper.

Theorem 2 *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 $O(\sum_{i=1}^n 2^{|M_i|})$.*

It is important to notice that since a service is in particular 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 bigger flat graph. We prove that this is not the case. Moreover, we prove that the complexity is linear in the number of components, and exponential only in the size of each base component. That is, we prove that with a smart algorithm, one can compute efficiently the absolute minimal observable 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 base components of size 2 (each one being reused 2^{n-1} times). The brute force non-compositional method runs on $\mathcal{H}$ and 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).

6. Experimental Results

We tested our decomposition 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 by using the *Synthetic DAG generation* tool (http://www.loria.fr/~suter/dags.html). 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 runtime and

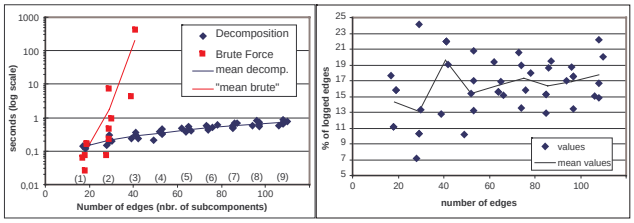

Figure 4. Execution time & observable size

observable size). We then unfold 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 algorithm on these graphs. We do not input the hierarchical shape of the graph, instead the algorithm finds the optimal decomposition with a polynomial time algorithm, see [4]. Fig. 4(left) shows the times (in logarithmic scale) needed to compute a minimal observable 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 subcomponents). Our decomposition algorithm is indeed linear time with respect to the number of base subcomponents/number of edges (0.14s for an average number of edges of 18 and 0.73s for an average number of edges of 108), while the brute force is exponential in the number of edges, already timing out at a little over 40 edges. For 1 subcomponent, the overhead of our method makes the decomposition slightly worse than the brute force method. Fig. 4(right) shows the percentage of edges needed to be logged among all the edges. Both algorithms answer the same number on the same graphs but there is a huge variation among graphs, from one edge needs to be logged out of 4 to one edge out of 15. The mean value seems to tend to one out of 6.

7. Discussion and Conclusion

We studied compensation under partial log visibility. To the best of our knowledge, this problem has never been considered in the context of transactional services. We proposed a framework which uses the hierarchical nature of composite services, and gives an efficient algorithm to compute the absolute minimum number of transitions to observe in order to get compensability.

The algorithm we proposed considers only a subset of the whole set of transitions. It is thus straightforward to add constraints, such as, a subset of transitions "can/cannot be observed". It is very useful since in practice, we have to take into account privacy/security issues. The algorithm would then answer the absolute minimal observable set among those satisfying the constraints. Also, the hierarchical decomposition allows to deal with dynamicity. Indeed, if a service gets transformed (e.g., after the discovery/death of a sub-service), obtaining a minimal observable set would need recomputation, only at its level of the hierarchy (not below), plus *few* levels above (until the properties of a level are unchanged). It also allows to describe more accurately the details of a service which was considered atomic until now, in order to have feedback on where a service failed exactly. We explain in [4] how to deal with distributed systems, and with systems which are not given in a hierarchical way (using a folding algorithm).

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architecture and Applications*. Springer Verlag, ISBN: 3540440089, 2004.

[2] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM TOPLAS, 23(3)*, pages 1–31, 2001.

[3] D. Biswas. Compensation in the world of web services composition. *In SWSWPC*, pages 69–80, 2004.

[4] D. Biswas and B. Genest. Minimal observability for transactional hierarchical services. *available at http://www.crans.org/ğenest/BG08.pdf*.

[5] R. Kumar and V. Garg. Modeling and control of logical discrete event systems. *Kluwer Academic*, 1995.

[6] F. Lin and W. Wonham. On observability of discrete-event systems. *Information Sciences, 44(3)*, pages 173–198, 1988.

[7] M. Lohrey and S. Manneth. The complexity of tree automata and xpath on grammar-compressed trees. *Theoretical Computer Science, 363(2)*, pages 196–210, 2006.

[8] S. Maheshwari. Traversal marker placement problems are np-complete. *Boulder Univ. Report CU-CS-092-76*, 1976.

[9] C. Ozveren and A. Wilsky. Observability of discrete event dynamical systems. *IEEE Trans. Auto. Control, 35(7)*, pages 797–806, 1990.

[10] W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. *In Jewels are Forever, Springer*, pages 262–272, 1999.

[11] K. Rohloff and J. Schuppen. Approximating the minimal cost sensor selection for discrete-event systems. *JDEDS, 16(1)*, pages 143–170, 2006.

[12] W. Sadiq and M. Orlowska. Analyzing process models using graph reduction techniques. *Inf. Syst., 25(2)*, pages 117–134, 2000.

[13] M. Sampath, R. Sengupta, S. Lafortune, K. Sinaamohideen, and D. Teneketzis. Diagnosability of discrete event systems. *IEEE Trans. Auto. Control, 40(9)*, pages 1555–1575, 1995.

[14] G. Weikum, A. Deacon, W. Schaad, and S. H. Open nested transactions in federated database systems. *IEEE Data Engg. Bulletin, 16(2)*, pages 4–7, 1993.

[15] A. Wombacher, P. Fankhauser, and E. Neuhold. Transforming bpel into annotated deterministic finite state automata for service discovery. *In ICWS*, pages 316–323, 2004.

[16] Web services transactions specifications. *http://msdn2.microsoft.com/en-us/library/ms951262.aspx*.

[17] T. Yoo and S. Lafortune. Np-completeness of sensor selection problems arising in partially observed discrete-event systems. *IEEE Trans. Auto. Control, 35(7)*, pages 797–806, 1990.