# On the Aggregation Problem for Synthesized Web Services

Ting Deng
Beihang University
dengting@act.buaa.edu.cn

Wenfei Fan
Leonid Libkin
Yinghui Wu
University of Edinburgh
{wenfei@inf, libkin@inf, Y.Wu-18@sms}.ed.ac.uk

## ABSTRACT

The paper formulates and investigates the aggregation problem for synthesized mediators of Web services (SWMs). An SWM is a finite-state transducer defined in terms of templates for component services. Upon receiving an artifact, an SWM selects a set of available services from a library to realize its templates, and invokes those services to operate on the artifact, in parallel; it produces a numeric value as output (e.g., the total price of a package) by applying synthesis rules. Given an SWM, a library and an input artifact, *the aggregation problem* is to find a mapping from the component templates of the SWM to available services in the library that maximizes (or minimizes) the output. As opposed to the composition syntheses of Web services, the aggregation problem aims to optimize the realization of a given mediator, to best serve the users' need. We analyze this problem, and show that its complexity depends on the underlying graph structure of the mediator: while it is undecidable when such graphs contain even very simple cycles, it is solvable in single-exponential time (in the size of the specification) for SWMs whose underlying graphs are acyclic. We prove several results of this kind, with matching lower bounds (NP and PSPACE), and analyze restrictions that lead to polynomial-time solutions.

## Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

## General Terms

Delphi theory

## Keywords

web services artifacts synthesis problem static analysis transducers


*ICDT 2010,* March 22-25, 2010, Lausanne, Switzerland.


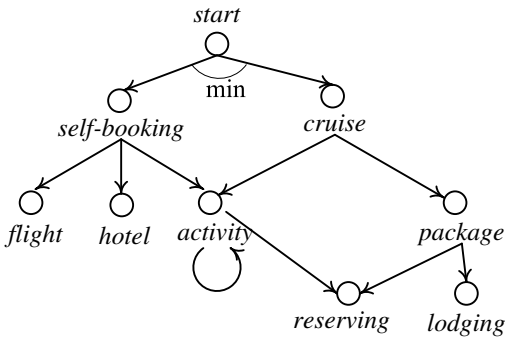


**Figure 1: An SWM specifying a travel planner**

## 1. Introduction

Fundamental research on Web services has mostly focused on service models, verification and composition. A variety of models have been proposed to specify the behaviors and interactions of Web services, based on finite-state automata [6, 16, 19], data-driven transducers [2, 5, 12, 13, 14, 31] or recently, artifacts [7, 11, 15]. A number of verification problems have been studied to decide, e.g., whether a transaction with certain properties can be generated by a service, or whether two services are equivalent [1, 2, 11, 12, 13, 14, 16, 17, 31]. The composition synthesis aims to determine whether available services can be coordinated to deliver a requested service, by automatically generating a mediator. Complexity bounds on the composition problem have been established for various service models [5, 6, 14, 19, 23, 24].

This paper studies a problem that has not yet received much attention, referred to as *the aggregation problem for Web services*. In practice a mediator is often predefined, in terms of templates for component services. Each template indicates a service of a certain functionality (e.g., for booking flight tickets or reserving hotel rooms), and is to be realized with an available service. Provided that a mediator and a library of available services are *already in place*, a natural question concerns how to find an optimal realization of the mediator that best serves the users' need. That is, given user's input, we want to generate a composite service *on the fly* by selecting a set of available services from the library and realizing templates in the mediator with these services, such that certain values representing the user's interest are maximized (e.g., benefits) or minimized (e.g., price). We illustrate the problem by an example.

**Example 1.1:** Consider a mediator $M_1$ for planning a trip to Disney World. Users are offered two options, as shown in Fig. 1. (1) They may book a flight, reserve a hotel room, and arrange activities separately, all by themselves. (2) Alternatively, they may opt for a cruise package, with which the choices of hotels are limited. In either option, the users may repeatedly make reservations for activities, e.g., Disney World, scuba, to fill out their free time slots.

The mediator $M_1$ is defined with component templates, e.g., *flight* and *activity*, which indicate services with a functionality for booking a flight and an activity, respectively. Such a template is to be realized with an available service having the required functionality. For example, *flight* can be realized with one of the online ticket booking systems launched by airlines or services such as Expedia and Priceline. Provided with travel dates, the available service that is chosen to realize *flight* returns the lowest airfare and reserves a ticket.

Provided travel dates and a list of free time slots, etc, the mediator is expected to explore both options.

(1) For the first option, it ranges over available services for checking flights, hotel rooms and activities. It picks the ones that lead to the minimum cost $C_1$, which is the sum of the airfare, accommodation cost and the costs of the activities chosen.

(2) For the second option, it ranges over cruise packages, and for each package, it inspects its lodging constraint and finds a hotel accordingly. It inspects activities as in the first option. The cost $C_2$ is the sum of the prices of the cruise, lodging and activities.

(3) After both $C_1$ and $C_2$ are found, the mediator returns the option with $\min(C_1, C_2)$. The option is reserved with the locked price [36], and recommended to the users. The users may then either decide to purchase the package, or cancel the reservation and repeat the process again. The actions are *not* committed *until* the users are ready to do so.

Observe that the templates in $M_1$ may be realized with possibly multiple available services. In this work we focus on how the mediator should realize its templates with the ones that lead to the lowest cost. □

The aggregation analysis is not only of theoretical interest. The need for it is also evident in practice. In response to practical demand, there have been service providers looking into service selection based on the quality of services, e.g., the Océano project at IBM [21]. However, the issue has not yet received a formal treatment, from models for specifying aggregation syntheses to the complexity of the problem.

The aggregation problem is, however, nontrivial. As we have seen from the example above, there are typically multiple choices of available services to realize a component template. Furthermore, there is *data flow* [23] among the components, i.e., the output of a component is passed as the input to another; as a result, the realization of a component is dependent on the choice of the services for the components that invoke it. In addition, the control flow of the mediator may be complex, e.g., represented as a tree, a DAG or a cyclic graph. These make this optimization problem rather challenging.

**Contributions**. We present a model to specify mediators with aggregation, formulate the aggregation problem, and establish complexity bounds on the problem for mediators of various structures.

*Mediators with aggregation.* We present a notion of *synthesized mediators for Web services* (SWMs), which extends mediators studied in [14] by incorporating aggregation synthesis. An SWM specifies a requested service that takes an artifact as input, and returns an aggregate value at the end. We consider artifacts that are updatable records representing the life-cycle of the processing of a requested service (see [7, 11, 15, 25] for detailed discussions about artifacts).

An SWM $M$ is a finite-state transducer. Each state has a *transition rule* and a *synthesis rule*. The transition rule is specified with a precondition, component templates and successor states. Upon receiving an artifact, it checks whether the precondition is satisfied; if so it realizes the templates with available services in a library, invokes the services to operate on the artifact in parallel, and passes the updated artifacts downward to its successor states. The synthesis rule is to compute aggregation value in the state. It is defined in terms of a polytime-computable function on the aggregate values of the successor states. That is, aggregate values are passed upward. The aggregate value generated in the start state of $M$ is returned as (part of) the output of the service.

*A formulation of the aggregation problem.* An SWM $M$ is realized with available services in a library $L$. A service in $L$ is a function that takes an artifact as input and returns an (updated) artifact. A realization of $M$ in $L$ is a mapping $\rho$ from the templates of $M$ to $L$. Substituting service $\rho(\tau)$ for each template $\tau$ of $M$ yields a *composite service* $M[\rho]$.

To ensure that the composite service generated by a realization $\rho$ is sensible, we also consider *realization constraints* on $\rho$ that specify what available services are allowed to realize a template.

Given an SWM $M$, an input artifact $t$, a library $L$, and a realization constraint $\lambda$, the *aggregation problem*, denoted by $\mathsf{AGP}(M, L, \lambda, t)$, is to find a realization $\rho$ of $M$ in $L$ that satisfies $\lambda$ and maximizes (or minimizes) the output of $M[\rho]$ on the input $t$.

*Complexity bounds.* The control flow of an SWM $M$ can be depicted as a graph $G[M]$ of a form similar to Fig. 1, in which nodes are states of $M$ and an edge $(s_1, s_2)$ indicates that $s_2$ is a successor state of $s_1$. We establish lower and upper bounds on $\mathsf{AGP}(M, L, \lambda, t)$, all matching, for $M$ of various structures. We show that $\mathsf{AGP}(M, L, \lambda, t)$ is undecidable when $G[M]$ is cyclic. In fact, for every cyclic graph $G$, the aggregation problem is undecidable over SWMs $M$ so that $G[M] = G$. But when $G[M]$ is not cyclic (i.e., is a DAG), the aggregation problem becomes decidable. Note that for many verification problems that ask questions about specifications, rather than data, single-exponential running time is viewed as acceptable (and in many cases unavoidable) [10]. We show that by forbidding cycles we get such acceptable algorithmic solutions: the problem is PSPACE-complete in the acyclic case, and the complexity drops further to NP-complete (but approximation-hard) when $G[M]$ is a tree.

We also study special cases of $\mathsf{AGP}(M, L, \lambda, t)$. In particular, we give the complexity bounds for the problem when

$M$ is fixed but $L$ varies, and when $L$ is fixed while $M$ may change. The former is to cope with a set of predefined mediators when the library $L$ may take new services or drop obsolete services, and the latter is to accommodate the practical setting where a relatively stable library $L$ serves various mediators. We show that the former simplifies the aggregation synthesis, e.g., $\mathsf{AGP}(M, L, \lambda, t)$ is in PTIME as opposed to PSPACE-complete for DAG-structured $M$. In contrast, the latter does not make our lives easier: the complexity bounds remain intact when $L$ is fixed.

To the best of our knowledge, this work is among the first formal treatments of aggregation syntheses of Web services. Our results provide a comprehensive picture of complexity bounds for the aggregation problem. In addition, the proofs provide algorithmic insight for developing effective methods to conduct the syntheses.

**Related work**. Several algorithms have been developed for selecting available services for service composition, based on the quality of services (QoS) [8, 20, 34, 35]. Previous work on QoS differs from this work in the following aspects. (a) The criteria for QoS focus on system issues such as service response time, cost, reliability, availability, trust and bandwidth. In contrast, the aggregation problem is to maximize (or minimize) certain values in an artifact representing users' interest, which are the data processed by the services and are returned as output. (b) The complexity of the aggregation problem largely comes from data flow among component services, i.e., the output of one component is treated as the input of another. In contrast, data flow is not a major issue for previous work on QoS. (c) Previous work on QoS does not address how aggregation syntheses are expressed in Web services. Furthermore, previous results mostly consist of heuristic algorithms for estimating QoS and selecting available services accordingly; complexity bounds for service selection are not studied, except [8]. An NP-complete bound was shown in [8] for optimal selection of available services, for pipelined (linear-structured) services based on a QoS model, in the presence of constraints on connecting a pair of services. The QoS model and the constraints of [8] are quite different from the aggregation syntheses and constraints studied in this work. Indeed, in the QoS settings the optimal selection problem remains in NP even for DAG-structured services [34], as opposed to the PSPACE-complete bound of this work.

A number of standards have been developed for specifying Web services, e.g.,WSDL [33], WSCL [32], OWL-S [26], SWFL [30] and BEPL [9]. A variety of models have also been proposed to characterize services supported by those standards, based on finite-state automata [6, 16, 19], data-driven transducers [2, 5, 12, 13, 14, 31] or artifacts [7, 11, 15]. The notion of SWMs is a refinement of synthesized mediators studied in [14], which shows that its mediators are able to express automaton and transducer abstractions of services. SWMs refine mediators of [14] by defining synthesis rules in terms of aggregation functions. They emphasize data flow among component services, along the same lines as [23]. Meanwhile SWMs specify control flow in terms of transitions of a transducer. To our knowledge, only [14] and the split-join operator of OWL-S [26] allow one to express synthesis operations, and moreover, no previous model supports aggregation syntheses.

As remarked earlier, several verification problems have been investigated for Web services [1, 2, 11, 12, 13, 14, 16, 17, 23, 31]. Complexity bounds have also been developed for the composition problem [5, 6, 14, 19, 23, 24]. To our knowledge, however, no previous work has studied the aggregation problem. In particular, the aggregation problem is quite different from the composition problem. The latter is a decision problem to determine whether there exists a mediator that coordinates available services to deliver a requested service; in contrast, the former is an optimization problem that aims to find a realization of a *given* mediator to maximize (or minimize) certain values in an artifact.

An artifact is an identifiable record in which attributes may be created, updated, or deleted [7, 11, 25]. It represents the life-cycle and business-relevant data of a business entity [15]. In this work we use artifacts to characterize input messages to a composite service, communications between components during a run of the service, and the output of the run of the service.

**Organization**. We present SWMs in Section 2, and formulate the aggregation problem in Section 3. We establish the undecidability of the aggregation problem in Section 4. We identify decidable special cases and provide their matching complexity bounds in Section 5. Finally, section 6 summarizes the main results and identifies open problems.

## 2. Synthesized Mediators for Web Services

We now define the syntax and the semantics of SWMs.

### 2.1 Synthesized Mediators

Before we formally define SWMs, we first describe artifacts and component templates.

**Artifacts and templates**. Following [15] we simply treat an artifact as a record specified by an *artifact schema*

$$R_A = (\mathsf{val} : \mathbb{Q}, A_1 : \theta_1, \ldots, A_n : \theta_n),$$

where each $A_i$ is an attribute and $\theta_i$ is its domain. We have a designated attribute $\mathsf{val}$ with the domain $\mathbb{Q}$ of rational numbers (for storing aggregate values). We assume that a special symbol $\perp$ is in each of the domains, denoting undefined as usual. We use $\mathcal{I}(R_A)$ to denote the set of all artifacts of schema $R_A$.

We assume a countably infinite set $\Gamma$ of *template names* for component services. Each template denotes a service of a certain functionality.

**Mediators**. A synthesized mediator (SWM) is a finite-state transducer defined in terms of component templates. When the templates are realized with available services, the SWM coordinates those services to deliver a requested composite service. More specifically, upon receiving an artifact, the SWM invokes the component services to operate on the artifact, and redirects artifacts by routing the output of one service to the input of another [9]. It generates the output of the requested service by synthesizing certain values in the artifacts updated by the component services.

$(q_1, \mathsf{true}) \rightarrow (q_s, \tau_{id}), (q_c, \tau_{id})$ $\quad \mathsf{val}(q_1) \leftarrow \min(\mathsf{val}(q_s), \mathsf{val}(q_c))$

$(q_s, \mathsf{true}) \rightarrow (q_f, \tau_f), (q_h, \tau_h), (q_a, \tau_a)$ $\quad \mathsf{val}(q_s) \leftarrow \mathsf{val}(q_f) + \mathsf{val}(q_h) + \mathsf{val}(q_a)$

$(q_a, \phi_a) \rightarrow (q_a, \tau_a), (q_r, \tau_{id})$ $\ (\phi_a \text{ is } t.T_l \neq \emptyset)$ $\quad \mathsf{val}(q_a) \leftarrow \mathsf{val}(q_a) + \mathsf{val}(q_r)$

$(q_c, \mathsf{true}) \rightarrow (q_a, \tau_a), (q_p, \tau_p)$ $\quad \mathsf{val}(q_c) \leftarrow \mathsf{val}(q_a) + \mathsf{val}(q_p)$

$(q_p, \mathsf{true}) \rightarrow (q_r, \tau_{id}), (q_l, \tau_l)$ $\quad \mathsf{val}(q_p) \leftarrow \mathsf{val}(q_r) + \mathsf{val}(q_l)$

$(q_f, \mathsf{true}) \rightarrow .$ $\quad \mathsf{val}(q_f) \leftarrow .$ $\quad$ /* similarly for $q_h, q_l, q_r$ */

**Figure 2: The transition rules and synthesis rules of mediator $M_1$**

**Definition 2.1:** A *synthesized mediator* (for web services, referred to as an SWM) over an artifact schema $R_A$ is defined as $M = (Q, \delta, \sigma, q_0)$, where $Q$ is a finite set of *states*, $q_0$ is the *start state*, $\delta$ is a set of *transition rules*, and $\sigma$ is a set of *synthesis rules*, such that for each $q \in Q$, there exist a unique transition rule $\delta(q)$ and a unique synthesis rule $\sigma(q)$:

$$\begin{array}{lrcl} \delta(q): & (q, \phi) & \rightarrow & (q_1, \tau_1), \ldots, (q_k, \tau_k). \\ \sigma(q): & \mathsf{val}(q) & \leftarrow & F_q(\mathsf{val}(q_1), \ldots, \mathsf{val}(q_k)). \end{array}$$

Here $q, q_1, \ldots, q_k$ refer to states in $Q$, and

- all the $\tau_i$'s are template names from $\Gamma$ (referred to as *component templates* of $M$; the set of these templates in $M$ is denoted by $\Gamma(M)$);
- $\phi$, called the *precondition* of $q$, is a PTIME-computable predicate over artifacts of schema $R_A$;
- $k \geq 0$; in particular, when $k = 0$, the right-hand side (RHS) of the rules $\delta(q)$ and $\sigma(q)$ are empty; and
- $F_q : \mathbb{Q}^k \rightarrow \mathbb{Q}$ is a PTIME-computable function.

For a transition $(q, \phi) \rightarrow (q_1, \tau_1), \ldots, (q_k, \tau_k)$, we refer to $q_1, \ldots, q_k$ as the *successor* states of $q$ *carrying templates* $\tau_1, \ldots, \tau_k$, respectively. □

**Example 2.1:** The mediator $M_1$ described in Example 1.1 can be expressed as an SWM. The artifact schema for $M_1$ consists of attributes specifying (1) departure city, travel dates, and the number of tickets, (2) a list $T_l$ of free time slots to be filled, (3) a list $A_l$ of activities, initially empty, and (4) $\mathsf{val}$ indicating the total cost of a trip, initially $\perp$. We define mediator $M_1 = (Q_1, \delta_1, \sigma_1, q_1)$, where $Q_1 = \{q_1, q_s, q_c, q_f, q_h, q_a, q_r, q_p, q_l\}$, and the transition rules $\delta_1$ and synthesis rules $\sigma_1$ are shown in Fig. 2.

In the mediator $M_1$, the set $\Gamma(M_1)$ includes templates $\tau_f, \tau_h, \tau_a, \tau_p$ and $\tau_l$. As shown in Fig. 1, these templates are to be realized with available services for checking *flight*, *hotel*, *activity*, *cruise package* and *lodging*, respectively. Each of these services updates certain attribute values of the artifact. For example, $\tau_a$ updates attributes $A_l$ and $T_l$ by filling a time slot with an activity. In addition, $\Gamma(M_1)$ contains a dummy template $\tau_{id}$, which simply passes artifact to its successor state without incurring any changes.

Note that the synthesis rule for $q_1$ is defined with aggregation operator min, while the synthesis rule for $q_s$ is defined with the sum aggregate. We shall explain the semantics of $M_1$ in Example 2.2. □

**Remark.** We focus on SWMs in which the transitions are deterministic. The reason for this is twofold. First, SWMs can already encode nondeterminism to a certain degree. Indeed, nondeterminism in this scenario is encoded not by transitions, but by the choice of library functions that represent possible implementations of templates. An SWM thus encodes a variety of actual realizations, which will be defined shortly. Second, allowing nondeterminism in transitions leads to significant technical problems. To start with, one needs an ad-hoc method for defining the aggregate value of the mediator even when all library functions have been fixed, as many possible runs may exist. Even more unpleasantly, there are no meaningful structural restrictions that ensure decidability of the problem of choosing the best realization of an SWM. In light of these, we consider deterministic SWMs to focus on the main theme of the aggregation analysis, and defer a full treatment of nondeterminism to future work.

## 2.2 Semantics of Mediators

The semantics is defined via *realizations* of SWMs, which substitute available library services for template names. Once this is done, we give two ways of presenting the semantics of SWMs: a traditional, purely operational one, as well as an equivalent semantics that describe the run at once, rather than via a sequence of steps.

**Realizing SWMs**. We view available services as function on artifacts, i.e., functions $f : \mathcal{I}(R_A) \rightarrow \mathcal{I}(R_A)$. We shall only impose a condition that such functions are tractable, i.e., PTIME-computable. We assume that we have a *library* $L$ of available services to choose from. The library can be built by leveraging techniques for Web service discovery (e.g., [4, 29]).

In a nutshell, the output of an available service is used to update attribute values of the input artifact. The service conducts the computation based on data in its local database and the input artifact. While in practice it may take additional input from the users, to simplify the discussion we assume that all the input parameters are encompassed in the input artifact as attributes. This assumption does not change the complexity bounds for the aggregation problem to be investigated.

To make a composite service, an SWM needs to be realized by substituting available library services for its templates. Thus, we define a *realization* of an SWM $M$ in library $L$ as a mapping $\rho$ from the set $\Gamma(M)$ of templates of $M$ to $L$. We denote the result of substituting a library service $\rho(\tau)$ for each occurrence of $\tau$ in $M$ by $M[\rho]$, referred to as the *composite service of $M$ realized by $\rho$*.

To ensure that the services realized make sense, we need to impose constraints on realizations. For instance, it is not sensible if one realizes a template intended for airfare with a service for hotel. Thus, we define a *realization constraint*

as a mapping $\lambda$ from $\Gamma(M)$ to the powerset $\mathcal{P}(L)$ of $L$. A template $\tau$ is restricted to a set $\lambda(\tau)$ of available services that have the required functionality, such that $\tau$ is only allowed to be realized with a service in the subset $\lambda(\tau)$ of $L$. Realization constraints classify services in the library based on their functionality, and can be automatically found by capitalizing on Web service discovery methods [4, 29].

A realization $\rho$ of $M$ is said to be *valid w.r.t.* constraint $\lambda$ if for each $\tau$ in $\Gamma(M)$, we have $\rho(\tau) \in \lambda(\tau)$. We also say that a realization constraint $\lambda$ is *deterministic* if it uniquely determines the library service for each template, i.e. $|\lambda(\tau)| = 1$ for all $\tau$.

**Runs of composite service – operational semantics**. A composite service $M[\rho]$, where $M$ is defined over an artifact schema $R_A$, runs on artifacts of $R_A$. We present two equivalent notions of a run: one of purely operational, and the other of more denotational flavor.

For the operational notion, we define a step relation $\Rightarrow_{(M[\rho],t_0)}$, where $t_0$ is an artifact. The relation is between *execution trees* [5, 6]. One starts with a single-node execution tree labeled by the triple $(q_0, t_0, \perp)$, and proceeds until a terminal execution tree is reached, on which the step relation is not applicable. Then the value of the third attribute of the root's label in that execution tree is the result of running the composite service, i.e., $M[\rho](t_0)$.

More precisely, in an execution tree, each node $v$ is labeled by a triple $(q, t, w)$, where $q \in Q$, $t$ is an artifact of schema $R_A$, and $w \in \mathbb{Q} \cup \{\perp\}$. We refer to the value $w$ as $\mathsf{val}(v)$. For two execution trees $\xi$ and $\xi'$, we write $\xi \Rightarrow_{(M[\rho],t_0)} \xi'$ if one of the following conditions holds.

**Spawning.** If there is a leaf node $v$ of $\xi$ labeled with $(q, t, \perp)$ (where the transition rule for $q$ is $(q, \phi) \rightarrow (q_1, \tau_1), \ldots, (q_k, \tau_k)$) and $\xi'$ is obtained from $\xi$ as follows.

- If $k = 0$ or $\phi$ evaluates to $\mathsf{false}$ on $t$ (i.e., either $q$ has no successor state, or the precondition for $q$ does not hold), then $\xi'$ is obtained from $\xi$ by setting $\mathsf{val}(v)$ to the value of the $\mathsf{val}$ attribute of $t$.
- Otherwise $\xi'$ is obtained from $\xi$ by spawning $k$ children $u_1, \ldots, u_k$ of $v$, in parallel. For each $i \in [1, k]$, a distinct node $u_i$ is created as the $i$-th child of $v$. The node $u_i$ is labeled with $(q_i, \rho(\tau_i)(t), \perp)$, i.e., it invokes available service $\rho(\tau_i)$ and labels $u_i$ with the updated artifact $\rho(\tau_i)(t)$.

**Synthesizing.** If there is no leaf node to which a transition rule applies, then $\xi'$ is obtained from $\xi$ by picking a node $v$ labeled by $(q, t, \perp)$ so that none of its successors $u_1, \ldots, u_k$ has $\mathsf{val}(u_i) = \perp$, and updating $\mathsf{val}(v)$ according to the synthesis rule: $\mathsf{val}(v)$ gets the value $F_q(\mathsf{val}(u_1), \ldots, \mathsf{val}(u_k))$, where $F_q$ is the aggregate from the synthesis rule for $q$.

In other words, the synthesis rule is applied if $\mathsf{val}(v) = \perp$ as soon as $\mathsf{val}(u_i)$ is available for all $i \in [1, k]$.

The run starts from an execution tree $\xi_0$ consisting of a single root node $r$, labeled with $q_0$, the input artifact $t_0$ and carrying $\mathsf{val}(r) = \perp$. Then an execution tree is generated top-down; spawning new nodes stops at a node reached if either the node is in a "final state" $q$ indicated by the transition rule of $q$ (with an empty RHS), or at the node the precondition $\phi$ is not satisfied. In both cases $\mathsf{val}$ at such a node carries a non-$\perp$ value. The synthesis rule for state $q$ is applied bottom-up to a node $v$ labeled with $(q, t, \perp)$ as soon as $\mathsf{val}(u_i)$'s are available for all its children.

If the process stops, $\mathsf{val}(r)$ is the output. More precisely, the *result* of the run of $M[\rho]$ on artifact $t_0$ is an execution tree $\xi$ such that $\xi_0 \Rightarrow^* \xi$ and there is no distinct $\xi'$ satisfying $\xi \Rightarrow \xi'$ (of course $\Rightarrow^*$ is the reflexive-transitive closure of $\Rightarrow$). The *output* $M[\rho](t_0)$ is the content of $\mathsf{val}(r)$ at the root $r$ of the result of the run.

The process may not necessarily stop when a mediator $M$ is "recursively defined", i.e., when a state in $M$ can reach itself after one or more transitions. In other words, there may not exist a finite execution tree $\xi$ such that $\xi_0 \Rightarrow^* \xi$ and $\xi$ cannot be further expanded via spawning. When this happens, $M[\rho](t_0)$ is undefined.

**Denotational semantics**. Note that while there may be multiple runs of a composite service, their results coincide, and thus the output is uniquely defined. In fact, one can compactly represent the output of such runs by a single tree, as shown in the easily verified proposition below. The proposition suggests a semantics of denotational flavor, which is equivalent to its operational counterpart given above.

**Proposition 2.1:** *For a composite service $M[\rho]$ and an artifact $t_0$ of schema $R_A$, the result of a run and the output of $M[\rho]$ on $t_0$ are either a $(Q \times I(R_A) \times \mathbb{Q})$-labeled tree $\xi$ and a number $w_0 \in \mathbb{Q}$ satisfying the following conditions:*

1. *the root of $\xi$ is labeled with $(q_0, t_0, w_0)$;*
2. *consider a node $v$ of $\xi$ labeled with $(q, t, w)$, where $(q, \phi) \rightarrow (q_1, \tau_1), \ldots, (q_k, \tau_k)$ is the transition rule for $q$;*
   (a) *$v$ is a leaf iff $w = t.\mathsf{val}$ and either $\phi(t) = false$ or $k = 0$;*
   (b) *$v$ is a non-leaf node iff it has $k$ children labeled with $(q_i, \rho(\tau_i)(t), w_i)$ for $i = 1, \ldots, k$ so that $w = F_q(w_1, \ldots, w_k)$, where $F_q$ is the aggregate in the synthesis rule for $q$.*

*or are undefined.*

**Example 2.2:** Recall the mediator $M_1$ from Example 1.1. Given an artifact $t_1$ of schema $R_1$ and a realization $\rho_1$, the execution tree specifying the run of $M_1[\rho_1]$ on $t_1$ is constructed as follows, as depicted in Fig. 3.

**(1)** It starts with a tree $\xi_0$ consisting of only the root node $r$, labeled with $(q_1, t_1, \mathsf{val}(r) = \perp)$.

**(2)** Since the preconditions for $q_s$ and $q_c$ are $\mathsf{true}$, the tree $\xi_0$ is expanded to $\xi_1$ by creating two children $v_s$ and $v_c$ for root $r$, labeled with $(q_s, t_1, \perp)$ and $(q_c, t_1, \perp)$, respectively. Note the dummy service $\tau_{id}$ simply passes the input artifact $t_1$ to $v_s$ and $v_c$.

**(3)** At node $v_s$, the available services $\rho_1(\tau_f)$, $\rho_1(\tau_h)$ and $\rho_1(\tau_a)$ are invoked unconditionally, in parallel with parameter $t_1$ associated with $v_s$. The tree $\xi_1$ is expanded by creating three children $v_f, v_h, v_a$.

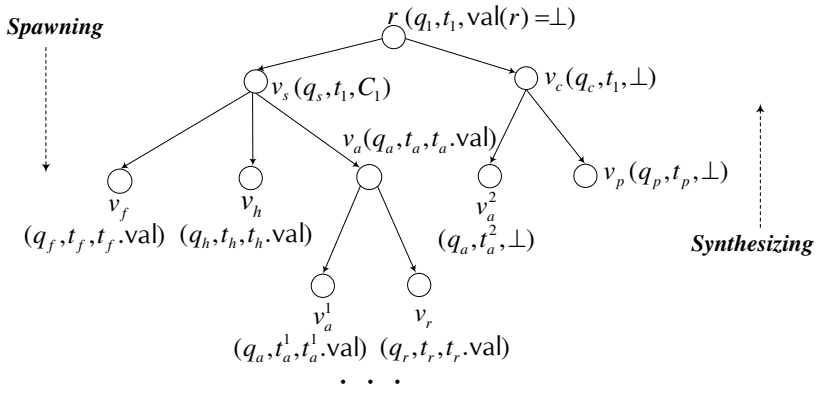


**Figure 3: An example execution tree**

At node $v_f$, assume that $t_f$ is the output artifact of $\rho_1(\tau_f)$, and $t_f.\mathsf{val}$ is the airfare found by $\rho_1(\tau_f)$ based on the data in the input $t_1$ and the local database of $\rho_1(\tau_f)$. Since state $q_f$ does not have any successor state, $v_f$ does not spawn any new node, and $\mathsf{val}(v_f)$ is simply set to be $t_f.\mathsf{val}$; similarly for $v_h$.

On the other hand, at node $v_a$, if the precondition $\phi_a(t_1)$ is satisfied, $\rho_1(\tau_a)$ is triggered to find an activity. It returns an artifact $t_a$, which updates $t_1$ by filling a free time slot with an activity, i.e., adding the newly chosen activity to $t_1.A_l$ (treated as $t_a.A_l$), and removing the corresponding slot from $t_1.T_l$ (denoted as $t_a.T_l$). It spawns two children $v_a^1$ and $v_r$, and passes $t_a$ to them. While the node $v_r$ simply retains $t_a.\mathsf{val}$ for synthesizing (denoted as $t_r.\mathsf{val}$), the process repeats at node $v_a^1$, which invokes $\rho_1(\tau_a)$ to select activities for the remaining time slots in $t_a.T_l$. The tree expands until all the free time slots are filled, i.e., when the precondition $\phi_a$ no longer holds.

**(4)** As soon as the spawning process terminates for the subtree of $v_a$, the synthesizing phase starts for the subtree of $v_s$. Synthesizing val values upwards, $\mathsf{val}(v_a)$ is set to be the sum of the costs for all the chosen activities. When $\mathsf{val}(v_a)$ is available, $\mathsf{val}(v_f)+\mathsf{val}(v_h)+\mathsf{val}(v_a)$ is computed and assigned to $\mathsf{val}(v_s)$, which is the cost $C_1$ shown in Fig. 3.

**(5)** Similarly, at node $v_c$ two children $v_a^2$ and $v_p$ are created. In particular, at the node $v_p$ the service $\rho_1(\tau_p)$ is triggered to select a cruise package, which yields artifact $t_p$. Based on the package selected and its constraint on lodging, a hotel is chosen by invoking $\rho_1(\tau_l)$, taking $t_p$ as the input parameter.

Along the same lines as described above, the subtree rooted at $v_c$ is completed and $\mathsf{val}(v_c)$ is computed. At this point $\mathsf{val}(r)$ can be computed, as $\min(\mathsf{val}(v_s),\mathsf{val}(v_c))$. This yields the result of the run, an execution tree in which no node $v$ has $\mathsf{val}(v)=\perp$. The output $M_1[\rho_1](t_1)$ of the run is $\mathsf{val}(r)$. □

To sum up, a transition rule indicates a business rule, and the precondition for each state determines whether its associated business rule should be carried out or not. An SWM specifies the control flow in terms of its transition rules, and the data flow with artifacts. There exist dependencies on the artifacts, e.g., the output artifact of $\rho_1(\tau_p)$ is the input of $\rho_1(\tau_l)$ in the example above; that is, the choice of hotel depends on what cruise package is selected in the previous state, as various cruise packages impose different lodging constraints. Also, to simplify the discussion, we only take a single artifact as input and produce a single value val as output. However, the definition of SWMs can be readily extended such that a composite service may take multiple artifacts as input and return multiple artifacts as output (including but not limited to val), and this does not change the results in the paper.

## 3. The Aggregation Problem

We now present the aggregation problem. Given an SWM $M$ over artifact schema $R_A$, an artifact $t$ of $R_A$, a library $L$ of available services and a realization constraint $\lambda$, *the aggregation problem* is to find a realization $\rho$ of $M$ in $L$ that is valid *w.r.t.* $\lambda$ and maximizes (or minimizes) $M[\rho](t)$.

Intuitively, given an input $t$ and a mediator $M$, the aggregation synthesis is to generate a composite service "on-the-fly" [28] that is "optimal" for user's request, by realizing templates of $M$ with available services *w.r.t.* the user's input. For instance, the aggregation synthesis for SWM $M_1$ of Example 1.1 is an instance of the aggregation (minimization) problem.

As usual, to study the complexity, we turn to a decision version of the aggregation problem. In such a version, we are interested in a valid realization $\rho$ of $M$ in $L$ so that $M[\rho](t) \geq B$, for a predefined bound $B$.

| PROBLEM: | $\mathsf{AGP}(M, L, \lambda, t)$ |
|---|---|
| INPUT: | 1. An SWM $M$ over an artifact schema $R_A$;<br>2. an artifact $t$ of $R_A$;<br>3. a library $L$ of available services,<br>4. a realization constraint $\lambda$. |
| QUESTION: | Does there exist a realization $\rho$ of $M$ in $L$ valid *w.r.t.* $\lambda$ so that $M[\rho](t) \geq B$? |

Of course one can change the sign of all the values and

aggregate functions and arrive at an equivalent minimization problem which asks whether $M[\rho](t) \leq B$. If we want to emphasize whether we refer to the maximization ($M[\rho](t) \geq B$) or minimization ($M[\rho](t) \leq B$) version, we shall write $\mathsf{AGP}_{\max}$ or $\mathsf{AGP}_{\min}$, resp.

Our goal is to investigate the complexity bounds of $\mathsf{AGP}(M, L, \lambda, t)$ for SWMs $M$ of various structures. More precisely, we define the *mediator graph* of an SWM $M = (Q, \delta, \sigma, q_0)$, denoted by $G[M]$, as a directed edge-labeled graph $G[M] = (Q, E, L)$ in which there is an edge $(q, q')$ in $E$ labeled with $\tau$ if $q'$ is a successor state of $q$ carrying template $\tau$, i.e., $(q', \tau)$ is in the RHS of the transition rule for $q$ in $M$. In the sequel we simply write $G[M]$ as $(Q, E)$ when $L$ is clear from the context. Apparently an SWM $M$ is recursively defined if $G[M]$ is cyclic.

We start by showing that the general problem is undecidable even for very simple SWMs that have a single state and whose underlying graph is a self-loop. In fact, we show that for *every* graph containing a cycle, the aggregation problem for mediators with that underlying graph is undecidable (even if some of the parameters are fixed). As an example, Figure 1 depicts an SWM with a cyclic graph structure.

So this suggests a restriction to SWMs whose mediator graph is a DAG. We show that for such DAG-structures SWMs the problem is decidable in PSPACE, and the further restriction to tree-structured SWMs puts the problem in NP.

## 4. Aggregation Synthesis: undecidability

In this section we show that the general aggregation synthesis problem $\mathsf{AGP}(M, L, \lambda, t)$ is undecidable, and identify restrictions that need to be put on the parameters of the problem to achieve decidability.

Recall that the mediator graph for an SWM $M$ is the graph $G[M]$ whose nodes are reachable states of $M$, and which has an edge from $q$ to $q'$ if $q'$ appears in the right-hand side of the unique transition rule for $q$ in $M$.

We then have the following undecidability result. Recall that a realization constraint $\lambda$ is *deterministic* if $|\lambda(\tau)| = 1$ for all $\tau$, i.e., for each template, the library service realizing it is uniquely determined.

**Theorem 4.1:** *Let $G$ be an arbitrary connected graph with a cycle. Then there exists an SWM $M_0$ whose mediator graph is $G$, a fixed library $L_0$ and a deterministic realization constraint $\lambda_0$ such that the problem $\mathsf{AGP}(M_0, L_0, \lambda_0, t)$ (whose only input is $t$) is undecidable.*

**Proof sketch:** The reduction is from the existence of solutions to Diophantine equations with a fixed number of variables and of fixed degree (e.g., degree 16 with 29 variables [22]). The artifact codes the coefficients of such a polynomial (note that the schema is fixed) as well as a code for a tuple of variables (with the initial value 0) and the value of the Diophantine polynomial on the decoded tuple. We use a fixed library consisting of a single function $f$ that increases the value of the code by 1. The SWM has one state $q$, with transition and synthesis rules as follows:

$$(q, P \neq 0) \rightarrow (q, \tau), \quad \mathsf{val}(q) \leftarrow \mathsf{val}(q).$$

Here $P$ decodes the code value and computes the value of the polynomial, in PTIME. If the value is 0, the val attribute is set to 0 and propagated up. Otherwise the function $f$ is invoked to increase the code by 1, and the process proceeds. A run does not terminate if there is no solution; otherwise it terminates with output 0[1].

The graph of this SWM has one node with a self-loop. For graphs with larger cycles just add dummy states. □

A slight modification of the proof shows the following undecidability result.

**Corollary 4.2:** *The aggregation problem is undecidable even if the library, the (deterministic) realization constraint, the artifact, and the cyclic mediator graph are fixed. That is, for an arbitrary connected graph $G$ with a cycle, there exist a fixed library $L_0$, a deterministic realization constraint $\lambda_0$ and an artifact $t_0$ so that the problem $\mathsf{AGP}(M, L_0, \lambda_0, t_0)$, whose only input is an SWM $M$ with the mediator graph $G$, is undecidable.*

**Proof sketch:** We follow the previous proof and remove the coefficients of the polynomial from the artifact and instead put attributes for the values of the decoded tuple of variables. The SWM produces the tuple from a code and computes the polynomial in the precondition, i.e., we have a transition rule $(q_0, p(\bar{n}) \neq 0) \rightarrow (q_1, \tau)$, where $p$ is the Diophantine polynomial and $\bar{n}$ refers to the $k$-tuple holding the decoded values. The library consists of a single function $f$ as in the previous proof. □

Analyzing the proof, we see that there are two main reasons for undecidability:

1. cyclicity of the mediator graph (even a single cycle leads to undecidability), and
2. the infinite domain of attribute values of the artifact.

The second constraint is essential for many applications as artifacts store numbers, dates, strings, etc. So we need to impose restrictions on the mediator graph. As no cycles are allowed, we shall look at mediator graphs which are DAGs and trees. But now, for completeness only, we present a simple result for the case of fixed-size domain.

**Proposition 4.3:** *Assume that the size of the domain of each attribute of the artifact schema is fixed. Then $\mathsf{AGP}(M, L, \lambda, t)$ can be solved in single-exponential time. If $M$ and the artifact schema are fixed as well, it is solvable in polynomial time.*

**Proof sketch:** When the size of the domain is fixed, there are at most exponentially many artifacts, and then runs of each realization can be accepted by exponential-size tree automata. Since the number of realizations with a fixed domain is exponential too, we need to check nonemptiness of exponentially many tree automata, each of exponential size, which can be done in single-exponential time. When $M$ and the artifact schema are fixed too, we have polynomially many realizations and artifacts, and the problem becomes polynomial. □

[1]We thank an anonymous referee for suggesting an improvement of our initial proof.

## 5. Decidable Cases

In this section we identify special decidable cases of the aggregation problem. We investigate $\mathsf{AGP}(M, L, \lambda, t)$ for SWMs $M$ that are not recursively defined, i.e., when the mediator graph $G[M]$ of $M$ is acyclic. As a result, one does not have to worry about the termination of runs of composite services realized with these SWMs.

### 5.1 Tree-Structured Mediators

We start with $\mathsf{AGP}(M, L, \lambda, t)$ for *tree-structured* SWMs $M$, i.e., when $G[M]$ is a tree.

**Complexity**. Our first result shows that the aggregation problem indeed becomes decidable when $G[M]$ has a tree structure. In fact, it can be solved in single-exponential time, which is acceptable for static analysis of specifications such as SWMs.

The problem is, however, intractable even for simple "pipelined" SWMs, i.e., when $G[M]$ has a linear (chain) structure. More specifically, $M$ has a *pipelined structure* if every transition rule either has an empty right-hand side, or is of the form $(q, \phi) \rightarrow (q', \tau)$. Moreover, the intractability is rather robust: it holds even if we fix the library $L$ (which is a reasonable assumption, as in practice, a library of available services may be relatively stable: it is only updated periodically).

**Theorem 5.1:** *$\mathsf{AGP}(M, L, \lambda, t)$ is NP-complete for tree-structured SWMs. The problem remains NP-hard when the library $L$ is fixed, and when the mediator $M$ has a pipelined structure.*

**Proof sketch:** There is an NP algorithm that, given a tree-structured SWM $M$, a library $L$, an artifact $t$, a constraint $\lambda$ and a number $B$, checks whether there exists a realization $\rho$ valid *w.r.t.* $\lambda$ such that $M[\rho](t)$ is smaller than (or greater than) $B$. The algorithm first guesses a realization $\rho$, and then checks whether $\rho$ is valid *w.r.t.* $\lambda$ and $M[\rho](t) \leq B$ (or $M[\rho](t) \geq B$), in PTIME.

We verify the NP lower bounds of $\mathsf{AGP}_{\max}(M, L, \lambda, t)$ and $\mathsf{AGP}_{\min}(M, L, \lambda, t)$ by reductions from 3SAT and non-tautology, respectively, which are known NP-complete problems (cf. [18]). The reductions are based on $M$ with a pipelined structure and a fixed $L$. □

In light of this intractability result one might be tempted to develop a PTIME approximation algorithm for the aggregation problem such that one can still efficiently find a solution with certain performance guarantee. However, this is also infeasible. The result below shows that the aggregation problem is not even in APX, the class of problems that allow PTIME approximation algorithms with approximation ratio bounded by a constant (see, e.g., [3] for APX).

Below we show a stronger result: $\mathsf{AGP}(M, L, \lambda, t)$ does not even allow PTIME approximation algorithms with approximation ratio bounded by any polynomial. An algorithm is said to *achieve an approximation ratio* $n^l$ for a maximization (resp. minimization) problem if for every instance of the problem, it produces a solution of value at least $\frac{1}{1+n^l}\mathsf{OPT}$, i.e., in the range $[\frac{1}{1+n^l}\mathsf{OPT}, \mathsf{OPT}]$ (resp. at most $(1 + n^l)\ \mathsf{OPT}$), where $l$ is fixed and $\mathsf{OPT}$ is the value of the optimal solution. We refer to such an algorithm as a *$n^l$-approximation algorithm*. The result below tells us that no matter what $n^l$ is used, it is impossible to find a PTIME $n^l$-approximation algorithm for $\mathsf{AGP}(M, L, \lambda, t)$ unless P = NP, even when $L$ and $M$ are fairly restricted.

**Theorem 5.2:** *Unless P = NP, there does not exist any PTIME $n^l$-approximation algorithm for $\mathsf{AGP}(M, L, \lambda, t)$, even when $M$ has a pipelined structure and when $L$ is fixed.*

**Proof sketch:** For $\mathsf{AGP}_{\max}(M, L, \lambda, t)$, we show that given an instance $\varphi$ of 3SAT, one can construct in PTIME a pipelined SWM $M$, a realization constraint $\lambda$ and an initial artifact $t$, with a fixed library $L$ of services, such that (a) if $\varphi$ is satisfiable, then there exists a realization $\rho$ such that $M[\rho](t) = 2^{|X|+1}$, where $X$ is the set of variables in $\varphi$ ($2^{|X|+1}$ is expressed in binary, in $O(|X|)$ space); and (b) otherwise for all realizations $\rho$, $M[\rho](t) = 0$. This suffices, for if there exists a PTIME $n^l$-approximation algorithm for the aggregation problem, then one can decide 3SAT in PTIME. Indeed, given any instance $\varphi$ of 3SAT, we can construct $M$, $\lambda, t$, and execute the algorithm on $M$, $\lambda, t$ and the fixed $L$, all in PTIME. We could then conclude that $\varphi$ is satisfiable iff the algorithm returns a value no less than $\frac{1}{1+|X|^l}2^{|X|+1}$. Hence we would have had a PTIME algorithm for 3SAT, which is impossible unless P = NP.

The proof for $\mathsf{AGP}_{\min}(M, L, \lambda, t)$ is similar, by reduction from non-tautology. □

### 5.2 DAG-Structured Mediators

We next investigate $\mathsf{AGP}(M, L, \lambda, t)$ for SWMs with a DAG structure. Like tree-structured SWMs, DAG-structured SWMs simplify the aggregation analysis: $\mathsf{AGP}(M, L, \lambda, t)$ is also decidable in this setting.

Given Theorem 5.1, the best one can hope for is that $\mathsf{AGP}(M, L, \lambda, t)$ remains in NP for DAG-structured SWMs. It turns out that for these SWMs, the complexity goes up, but the aggregation problem is still solvable in single exponential time (in PSPACE). Moreover, the PSPACE hardness bound remains intact even when the library $L$ is fixed and the realization constraint $\lambda$ is deterministic.

**Theorem 5.3:** *$\mathsf{AGP}(M, L, \lambda, t)$ is PSPACE-complete for DAG-structured SWMs. The problem remains PSPACE-hard when the library $L$ is fixed and the realization constraint $\lambda$ is deterministic.*

**Proof sketch:** To show the upper bound, we provide a nondeterministic algorithm that, given $M, L, \lambda, t$ and $B$, guesses a realization $\rho$, and checks in PSPACE whether (a) $\rho$ is valid *w.r.t.* $\lambda$, and (b) $M[\rho](t) \geq B$. Hence the problem is in PSPACE, since PSPACE = NPSPACE.

The PSPACE lower bound is verified by reduction from Q3SAT, which is known to be PSPACE-complete (cf. [27]). The reduction is such constructed that the library $L$ is fixed and there exists a unique realization $\rho$ valid *w.r.t.* realization

| Mediators $M$ | $\mathsf{AGP}(M, L, \lambda, t)$ | with fixed $L$ | with fixed $M$ |
|---|---|---|---|
| tree-structured | NP-complete (Th 5.1)<br>approximation-hard (Th 5.2) | NP-complete (Th 5.1)<br>approximation-hard (Th 5.2) | PTIME (Prop 5.4) |
| DAG-structured | PSPACE-complete (Th 5.3) | PSPACE-complete (Th 5.3) | PTIME (Prop 5.4) |
| graph-structured | undecidable (Th 4.1) | undecidable (Th 4.1) | undecidable (Th 4.1) |

**Table 1: Complexity bounds on the aggregation problem $\mathsf{AGP}(M, L, \lambda, t)$**

constraint $\lambda$. □

**When the mediator is predefined**. We have seen from Theorems 4.1, 5.1, 5.2 and 5.3 that fixing library does not make our lives easier: the lower bounds remain unchanged when the library of available services is predefined and fixed.

Another practical setting is that a service provider often maintains a set of predefined mediators. That is, the SWMs can be considered fixed, while the library $L$ is periodically updated by adding newly found available services to it, or removing obsolete services from it.

Below we show that fixing SWMs simplifies the aggregation synthesis: the problem is in PTIME for a fixed SWM $M$, when $M$ has a tree or a DAG structure. Contrast this to Theorem 4.1, which tells us that when the mediators are recursively defined, fixing both mediators and library does not help.

**Proposition 5.4:** $\mathsf{AGP}(M, L, \lambda, t)$ *is in PTIME when* $M$ *is a fixed* DAG*-structured SWM.*

**Proof sketch:** When $M$ is fixed, given any $L, \lambda, t$ and for any realization $\rho$ valid *w.r.t.* $\lambda$, it takes PTIME to compute $M[\rho](t)$. Indeed, the size of $M$ is a constant in this setting. From this it follows that it is in PTIME to find a realization $\rho$ that maximizes (or minimizes) $M[\rho](t)$. □

## 6. Conclusion

We have provided a formal treatment of the aggregation synthesis of Web service mediators, a problem that is of practical importance but has not been adequately addressed theoretically. We have developed a model for specifying mediators with aggregation synthesis, and formulated the aggregation problem. We have also established matching upper and lower bounds on the problem for mediators of various structures. The main results of the paper are summarized in Table 1. We showed that the problem is beyond reach in practice for recursively defined mediators, even when the mediators and the library of available services are predefined and fixed. Nevertheless, for mediators with a DAG or a tree structure, the problem becomes decidable in single-exponential time, which is an acceptable complexity for static analysis problems. More specifically, the problem is PSPACE-complete for DAG-shaped mediators, and is NP-complete for tree-shaped ones; it is even in PTIME when a set of predefined mediators are considered, a common setting in practice. These make our lives easier, but only to an extent: the NP-lower bound remains intact when the mediator has a pipelined structure and the library is fixed. Worse still, the problem does not allow any PTIME approximation algorithms with a polynomial ratio. The PSPACE lower bound is also robust when the library is fixed.

This work is a first step toward understanding the aggregation synthesis of Web services. There is naturally much more to be done. First, nondeterminism deserves a full treatment. While SWMs support a simple form of nondeterminism by means of the instantiations of templates, it is interesting and practical to extend SWMs by allowing a transition rule to be associated with multiple pre-conditions. As remarked earlier, however, this introduces several challenges. Second, the run of a composite service generated from an SWM may not terminate. We are currently investigating practical restrictions on SWMs such that every run is guaranteed to terminate and yield a solution. Third, while the aggregation problem is undecidable in general and is intractable for non-recursive SWMs, we expect that practical PTIME cases can be identified in certain specific application domains. Fourth, it is interesting to revisit the composition problem when aggregation synthesis is brought into the play. That is, we want to automatically generate mediators that coordinate available services and deliver a requested service, with aggregation analysis that aims to best serve the users' need. Finally, we would like to develop efficient heuristic algorithms to realize mediators with aggregation synthesis for specific applications. The operational semantics given in Section 2.2 provides a conceptual-level strategy for delivering a requested service. The strategy can certainly be improved by capitalizing on practical pruning techniques. For instance, one may employ a lazy evaluation strategy such that if some branch already yields a larger (or smaller) value than a given bound, realizations of the templates in other branches as well as their computation can be entirely avoided.

**Acknowledgments**. Fan is supported in part by EPSRC E029213/1 and a Yangtze River Scholarship. Fan and Libkin acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599. Deng is with the National Laboratory of Software Development Environment at Beihang University, and is supported in part by the 973 Program (2005CB321803) and the 863 Program of China (2007AA010301).

## 7. References

[1] S. Abiteboul, L. Segoufin, and V. Vianu. Static analysis

of active XML systems. In *PODS*, 2008.

[2] S. Abiteboul, V. Vianu, B. S. Fordham, and Y. Yesha. Relational transducers for electronic commerce. *JCSS*, 61(2):236–269, 2000.

[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial optimization problems and their approximability properties*. Springer Verlag, 1999.

[4] B. Benatallah, M.-S. Hacid, A. Leger, C. Rey, and F. Toumani. On automating Web services discovery. *VLDB J.*, 14(1), 2004.

[5] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *VLDB*, 2005.

[6] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioral descriptions. *Int. J. Cooperative Inf. Syst.*, 14(4):333–376, 2005.

[7] K. Bhattacharya, C. E. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *BPM*, 2007.

[8] P. A. Bonatti and P. Festa. On optimal service selection. In *WWW*, 2005.

[9] Business Process Execution Language for Web Services version 1.1 (BEPL4WS), 2004. *http://www.ibm.com/developerworks/library/specification/ws-bpel/*.

[10] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.

[11] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT*, 2009.

[12] A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven Web applications. *JCSS*, 73(3):442–474, 2007.

[13] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven Web services. In *PODS*, 2006.

[14] W. Fan, F. Geerts, W. Gelade, F. Neven, and A. Poggi. Complexity and composition of synthesized Web services. In *PODS*, 2008.

[15] C. Fritz, R. Hull, and J. Su. Automatic construction of simple artifact-based business processes. In *ICDT*, 2009.

[16] X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL Web services. In *WWW*, 2004.

[17] X. Fu, T. Bultan, and J. Su. Conversation protocols: a formalism for specification and verification of reactive electronic services. *TCS*, 328(1-2):19–37, 2004.

[18] M. Garey and D. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman and Company, 1979.

[19] C. E. Gerede, R. Hull, O. H. Ibarra, and J. Su. Automated composition of e-services: lookaheads. In *ICSOC*, 2004.

[20] S.-Y. Hwang, H. Wang, J. Tang, and J. Srivastava. A probabilistic approach to modeling and estimating the QoS of web-services-based workflows. *Inf. Sci.*, 177(23):5484–5503, 2007.

[21] IBM. The océano project. *http://researchweb.watson.ibm.com/oceanoproject/*.

[22] J. P. Jones. Undecidable Diophantine equations. *Bull. Amer. Math. Soc.*, 3(2):859–862, 1980.

[23] Y. Lustig and M. Y. Vardi. Synthesis from component libraries. In *FOSSACS*, 2009.

[24] A. Muscholl and I. Walukiewicz. A lower bound on Web services composition. In *FoSSaCS*, 2007.

[25] A. Nigram and N. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.

[26] OWL-S: Semantic Markup for Web Services, 2004. *http://www.w3.org/Submission/OWL-S/*.

[27] C. H. Papadimitriou. *Computational Complexity*. AW, 1994.

[28] M. Pistore, P. Roberti, and P. Traverso. Process-level composition of executable web services: "on-the-fly" versus "once-for-all" composition. In *ESWC*, 2005.

[29] C. Schmidt and M. Parashar. A peer-to-peer approach to web service discovery. In *WWW*, 2004.

[30] Semantic Web Services Framework (SWSF) Version 1.1, 2005. *http://www.daml.org/services/swsf/1.1/*.

[31] M. Spielmann. Verification of relational transducers for electronic commerce. *JCSS*, 66(1):40–65, 2003.

[32] Web Services Conversation Language (WSCL) 1.0, 2002. *http://www.w3.org/TR/wscl10/*.

[33] Web Services Description Language (WSDL) 1.1, 2001. *http://www.w3.org/TR/wsdl*.

[34] T. Yu and K.-J. Lin. Service selection algorithms for Web services with end-to-end QoS constraints. *Inf. Syst. E-Business Management*, 3(2):103–126, 2005.

[35] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for Web services composition. *IEEE Trans. Software Eng.*, 30(5):311–327, 2004.

[36] Y. Zhang and W. Fan et al. Extending online travel agency with adaptive reservations. In *Coopis*, 2007.