Managing Top-down Changes in Service-Oriented Enterprises *

Xumin Liu † Athman Bouguettaya †‡
† Department of Computer Science, Virginia Tech, USA
‡ CSIRO ICT Center, Canberra, ACT, Australia
{xuminl,athman}@vt.edu

Abstract

A Service Oriented Enterprise (SOE) provides an efficient and flexible platform where multiple Web services can cooperate together to provide a value-added service. Change management is one of the fundamental issues in enabling SOEs. In this paper, we propose a framework that facilitates in automatically managing top-down changes in SOEs. We start with formalizing a SOE’s schema since it is a central concept for specifying and managing top-down changes in SOEs. We then propose a change model as a guide to react to changes. Algorithms are proposed to implement changes by refining a SOE’s behavior.

1 Introduction

The last few years have witnessed an explosion of activities involving Web services. Significant efforts are invested in developing Web services. The Web is poised to take a new giant step to be a central repository for the ever increasing number of Web services [7]. This trend is motivating a paradigm shift in the enterprise structure from the traditional entity to a collaboration of Web services. Such Service Oriented Enterprises (SOEs) would potentially open the door of entrepreneurship to all Web users. Simply put, a SOE is a virtual, adaptive, extensible, market-driven, and on demand, Web service based enterprise.

1.1 Problem Statement

SOEs outsource their functionalities via third-party Web services. This triggers a need for a systematic approach to manage and maintain the proper functioning and cooperation of these services. This is of significant importance and very difficult because a SOE has to perform its functions in an extremely dynamic environment (i.e., on the Web). Market requirements and business regulations may change and individual services may come and go at will. In SOEs, changes are the rule, and are not the exception, as it is the case in traditional enterprises. Therefore, providing a framework for change management in SOEs is important.

There are two types of changes that happen to a SOE: top-down changes and bottom-up changes. Top-down changes refer to the changes that are initiated by SOEs’ owners. Bottom-up changes refer to the changes that are initiated by the outsourced Web service providers. In this paper, we focus on top-down changes that are always triggered by either new business strategies or new regulations. A SOE may frequently make top-down changes to improve business processes, enhance market competitiveness, and comply with new regulations.

1.2 Challenges

A top-down change is expected to occur frequently during the life-cycle of a SOE due to the dynamic environment (e.g., user’s requirement, marketing, laws). It is always affiliated with a new requirement on a SOE’s member services and the way they cooperate with each other. Efficiently managing a top-down change poses several new challenges.

First, top-down changes in a SOE may exhibit a great variety because SOEs may differ widely in their characteristics with respect to their functionality and quality. Thus, changes are usually specified in an ad-hoc manner and only for human consumption. Dealing changes based on such a change specification will naturally involve the tedious work that requires intensive human interferences.

Second, a top-down change represents a new requirement on a SOE’s behavior. Hence the process of reacting a top-down change may require a re-engineering of a SOE, such as adding (or removing ) certain functionality, or changing the way that the Web services cooperate with each other in a SOE. Manually performing the re-engineering process for each top-down change would introduce high costs.

Third, how a top-down change has been responded by the SOE should also be validated to make sure the new requirement is satisfied.

1.3 Solution

In this paper, we propose a solution to automate change management in a SOE. We define a SOE schema by lever-
agning the formalisms of the proposed Web service ontology, such as OWL-S and WSMO [1, 12]. This allows us to specify top-down changes in a formal way with sufficient reasoning capacity. The reasoning capacity provides foundational support to automate the change management process. We design a framework for change management in a SOE, which consists of two main components: change model, and change reaction. Change model is to formally specify changes. We classify top-down changes into different categories and investigate their features. Given a change specification resulted from the change model, change reaction is to adjust a SOE’s behavior to make it adapt to the change. The correctness of a SOE is maintained during change reaction. We summarize our contributions in this paper as follows:

- **SOE schema** – We formalize a SOE’s schema since it is a central concept for specifying and managing changes in a SOE. A SOE’s schema describes the business logic of a SOE by specifying a SOE’s member services and their cooperation patterns. Moreover, we define the correctness of a SOE. The process of change reaction should transfer a SOE’s schema from a correct state to another.

- **Change model** – We propose a formal specification for top-down changes of a SOE. We model a top-down change with respect to its motivation that can be translated as conditional modification of a SOE’s functionality and its quality.

- **Change reaction** – We propose a solution to react to changes, given their specification. Change reaction is performed at two levels: SOE schema level and concrete service level. At the first level, a SOE schema is modified. At the second level, concrete services are integrated based on the updated SOE schema and specified quality requirement.

**Example 1.1.** As a running example, we use an application from the travel domain. The business goal of this SOE is to provide a comprehensive travel package for users, including purchasing airline tickets, booking hotels, and renting cars. To achieve this business goal, the SOE collects the collaborative efforts from the corresponding Web services in the service space. Changes in a SOE may be very common and frequent due to the updates on the business requirement (i.e., business-centric changes) or organization regulations (i.e., regulation-centric changes). Suppose the travel agency SOE intends to add a taxi reservation service as an alternative to its car rental service for an international travel. It also wants to the airport and the hotel to provide more convenient network accessing. Fulfilling the new requirement demands intensive manual work. For example, to add a taxi service, the owner of the SOE needs to first select appropriate taxi reservation services from the service space. He needs to investigate the specification of the selected services to understand how the services can be used, such as the signatures of the service operations. He then needs to integrate the new service with the existing services by manually designing the data flow and control flow. Since the taxi service and the car rental service both provide ground transportation, the owner may also need to specify a selection mechanism for accessing these services. To address this issue, we propose a solution to automate the process of managing changes in this paper.

The remainder of this paper is organized as follows. In Section 2, we discuss some related work. In Section 3, we depict a SOE’s architecture and formalize a SOE’s schema that enables to specify and manage changes. In Section 4, we classify top-down changes into categories and investigate their features. In Section 5, we propose a model for specifying a top-down change. In Section 6, we propose a solution for change reaction. In Section 7, we conclude our paper.

## 2 Related Work

Change management is an active research area. Although it is relatively new in the Web service community, some frameworks are proposed for managing changes for process-oriented systems, such as workflows. In this section, we describe some related work in this area.

In [2], a framework is proposed for detecting, propagating, and reacting bottom-up changes in a SOE. A bottom-up change is initiated by an individual service without the consent of the enterprises that utilize the service. The changes may affect the invocation of other services in the SOE, which in turn affects the entire performance of the SOE. Petri-net is employed to formally model bottom-up changes. Ontologies are leveraged to facilitate dynamic service discovery from the service space. Agents help detect and manage bottom-up changes on the enterprise.

Our work focuses on a different type of changes on a SOE: top-down changes, which is initiated by the owner of a SOE and triggered by new business requirements.

Many frameworks for realizing adaptive workflows have been proposed [11, 3, 8, 10, 4]. They mainly focus on adjusting the process instances to a changed process schema. An important issue addressed by these frameworks is how to define correctness criteria, which evaluate the compliance of process instance with a changed process schema. Once such criteria are determined, mechanisms are proposed to propagate a process type change to an instance.

The focus of change management in a SOE is different from that of a workflow system. In a SOE, the key issue is to automate the process of reacting a top-down change by leveraging the Web service technologies. In contrast, change management is usually carried out in a manual way...
Figure 1. Architecture of a SOE

in workflow systems.

3 Preliminaries

In this section, we elaborate on several fundamental concepts that layout the ground of this paper. We first depict the architecture of a SOE. We then propose a SOE schema and identify its key features for specifying and managing changes in a SOE.

3.1 A SOE Architecture

Figure 1 depicts the architecture of a SOE. There are three key components in this architecture: SOE schema, ontology provider, and Web service space. The SOE schema defines business logics of a SOE. A SOE schema is created and maintained by a SOE owner. It specifies what kinds of services should be chosen as the members and how these members should work together. The service space provides the instantiation of the member services. The SOE schema also relies on the service ontology for getting semantic support. A SOE schema is a central concept for a SOE to specify and manage changes. We elaborate on the SOE schema in the following section.

3.2 SOE Schema

In this section, we propose a SOE schema to model the business logics of a SOE. A SOE is treated as a middleware between users and its member services. It consumes the input from users, mediates among its member services, and returns the result to users. A SOE schema consists of a set of service ontologies (i.e. its members) and the cooperation patterns among these members.

Generally, a SOE schema E is a pair \( \{ M, C \} \), where \( M = \{ S_1, S_2, \ldots, S_n \} \), represents the set of service ontologies, and \( C \) is the cooperation pattern between member services, which can be defined by its control flow (\( CF \)) and data flow (\( DF \)).

3.2.1 Control Flow

The control flow defines the execution order of the member services. We use the notions in dynamic logic to express the control flow of a SOE. In general, a control flow contains a set of control ontology notions and a set of predefined constructs: ‘+’ for non-deterministic choice, ‘;’ for sequence, ‘!’ for iteration, and ‘|’ for parallel. The service process expression can be recursively defined. For example, the control flow of ET is expressed as:

\[ CF_{ET} = S_A; S_R; S_H, \]

where \( S_A \) is an airline service ontology, \( S_R \) is a car rental service ontology, and \( S_H \) is a hotel service ontology. This expression shows that \( A \) is invoked first, which is followed by \( R \). When \( R \) is finished, \( H \) is then invoked.

From a control follow, we can derive a set of process constraints that can be used to specify a top-down change. A process constraint specifies the relationships between the member services. These relationships include existence (‘[ ’), sequence (‘\( \gg \)’), concurrence (‘\( || \)’), iteration (‘\( * \)’), and commutability (‘\( \leftrightarrow \)’). \( [S_1] \) means that \( S_1 \) must be invoked. \( S_1 \gg S_2 \) means that \( S_1 \) should be invoked before \( S_2 \). \( S_1 || S_2 \) means that \( S_1 \) and \( S_2 \) should be invoked in parallel. \( S_1 * \) means that \( S_1 \) can be invoked repeatedly. \( S_1 \leftrightarrow S_2 \) means that \( S_1 \) and \( S_2 \) provide the similar functionality and they can exchange with each other.

3.2.2 Data Flow

The data flow of a SOE defines how data is transferred from the user and among its member services. The output of a service might be used as the input of another service. Web services are heterogeneous with respect to the data format, terminologies, and granularity. A SOE is supposed to resolve the heterogeneity and enable the seamless data transfer.

We use a set of data transferrers to denote the data flow of a SOE. A data transferrer \( \gamma \) specifies the relationship between two sets of data items. It is denoted as: \( \gamma : D_f \rightsquigarrow D_i \), which means that \( D_i \) can be derived from \( D_f \) regardless the syntactical differences.

A SOE data flow is specified as a triplet: \( \{ In, Out, \gamma \} \), where \( In \) is the SOE’s input that is received from its users, \( Out \) is the SOE’s output that will be sent to its user, and \( \gamma = \{ \gamma_1, \gamma_2, \ldots, \gamma_m \} \) is a set of data transferrers.

For example, the data flow of a travel SOE \( ET \) is expressed as:

\[ In_{ET} = \{ accountInfo_{ET}, journeyInfo_{ET} \}, \]
\[ Out_{ET} = \{ travelPackage_{ET} \}. \]

\( \gamma \) includes:

\( \gamma_1 : \{ accountInfo_{ET} \} \rightsquigarrow \{ loginInfo_A \} \),
\( \gamma_2 : \{ accountInfo_{ET} \} \rightsquigarrow \{ loginInfo_H \} \),
\( \gamma_3 : \{ accountInfo_{ET} \} \rightsquigarrow \{ loginInfo_R \} \),
\( \gamma_4 : \{ ticket_A \} \rightsquigarrow \{ flight_H \} \),
\( \gamma_5 : \{ ticket_A \} \rightsquigarrow \{ flight_R \} \),
\( \gamma_6 : \{ ticket_A, rental_R, bookH \} \rightsquigarrow \{ travelPackage_{ET} \} \).

3.2.3 SOE Schema Correctness

A SOE schema is considered to be correct only if it meets two requirements: (1) It should be consistent between the
data flow and control flow. (2) The data flow should guarantee that all the member services receive their required inputs. The correctness of a SOE’s schema should be guaranteed after the process of change reaction. There are two resources for a member service to get its input: the user’s input and the output from other member services. Therefore, we can check the correctness of a SOE schema by following the control flow and starting with the user’s input as the available information. The SOE schema is correct if all the member services can get required input according to the execution order specified by the control flow.

4 Top-down Change Classification

Top-down changes are rules for an enterprise in the highly dynamic business environment. In order to manage top-down changes, we need to identify some fundamental characteristics of these changes. In this section, we first give a high-level overview of top-down changes in a SOE. We then classify these changes into different categories using the three important features as dimensions: motivation, effects, and behavior.

4.1 An Overview of Top-down Changes

Figure 2 shows changes in a SOE in three layers: business requirement changes, SOE schema changes, and SOE instance changes. The uppermost layer reflects the dynamic environment that a SOE is exposed to. A business requirement change could be brought by new technology, new business strategies, new market requirement, or new regulations and laws. It will be interpreted to SOE schema changes at the lower layer. A SOE schema gives a high-level abstraction of a SOE’s functionality and invocation. A business requirement change can be interpreted as modification of SOE’s functionality, invocation, and the updated performance requirement at the SOE schema level. The SOE schema change will then be propagated to SOE instance changes at the lowest layer to implement the change in practice. A SOE instance consists of a set of concrete Web services, which work together to achieve the business goal of a SOE. A SOE instance change can be specified as the modification of the list of these services and the way they cooperate with one another.

To specify a top-down change, it is essential to know: (1) when it will occur; (2) what effect it is supposed to make to a SOE; (3) how it behaves. The answers to these three questions actually deliver the information about the three important features of a change: motivation, effect, and behavior. The motivation defines the factors that will affect the occurrence of a change. The effect specifies the impact that a change is expected to make to a SOE. The behavior specifies how a change is implemented in a SOE, i.e., how a SOE adjusts its internal structure to adapt to the change. In the following sections, we classify changes by using these three features as dimensions.

4.2 Motivation of Changes

There are various factors that affect the occurrence of a top-down change in a SOE, such as business model, new technology, market requirement, and business regulations. Based on these factors, we classify top-down changes into two categories: business-centric and regulation-centric.

- **Business-centric changes** are initiated by a commercial requirement, such as business development and business improvement. For example, a travel agency SOE (e.g., priceline.com) may add a travel insurance service to make the package more comprehensive to attract more market interests.

- **Regulation-centric changes** are initiated to comply with new regulations, e.g., new legislations or business regulations. For example, a travel agency SOE may be required by a law to provide accidental insurance services for users.

4.3 Effects of Changes

Top-down changes are expected to bring various effects to a SOE. For example, the structure may be modified, the owner may change, the ways that access to a SOE may change, etc. Top-down changes can be classified into two categories based on their effects: External changes and Internal changes.

- **External changes** are the changes on a SOE that can be noticed by the outside. A SOE exposes itself to its outside with three features: functionality, invocation and performance. An external change is supposed to modify one or more of these three features. For example, a travel SOE may add a new cruise service (i.e., modify functionality). Users will notice the change when interacting with the SOE.

- **Internal changes** are the changes on a SOE that are transparent to the outside. The effects of these changes are hidden from the SOE’s users. For example, a SOE owner may add a new audit mechanism to monitor the running of the SOE. We will not address this type of changes in this paper.

4.4 Behavior of Changes

Change behavior specifies how to modify a SOE to implement a change. It can be simple (e.g. add a new functionality) or complicated (e.g. re-construct the whole SOE’s

![Figure 2. A layer of changes in a SOE](image-url)
architecture). We identify three basic types of top-down change behavior: adding/removing functionality, improving quality, and updating business logic. A top-down change behavior can be specified by one or the combination of these three basic types. We elaborate on these three types as follows:

- **Adding/Removing Functionality**: It is initiated with the purpose of business development. For example, suppose the owner wants to enhance the SOE by adding a cruise service in the provided travel package. As a result of this change, a cruise service should be added and combined with other member services. For another example, suppose the owner wants to remove a car rental service from the provided travel package since the service does not attract much interests from users. As a result of this change, the car rental service should be deleted. The other services should be re-integrated together.

- **Improving Quality**: It is initiated with the purpose of business improvement. Suppose the owner wants to offer a more economic travel package by decreasing the lodging expense. This type of changes does not modify the structure of the SOE. The Web service that offers the lower rate of hotel service will be selected to replace the expensive one.

- **Updating Business Logic**: It is initiated with the purpose of reconstructing the cooperation pattern between member services. For example, the owner wants to parallelize the invocations of the car rental service and the hotel service.

### 5 Change Model

Changes in a SOE are various and unpredictable. As a result, dealing with changes in an ad hoc manner will naturally involve the tedious and time-consuming work that demands intensive human interference. In this section, we use modeling as a means to manage the changes in a systematic and efficient way.

We formalize the change model that facilitates automatic change management. Specifically, \( C = \{ C, F M, Q M \} \), where \( C \) specifies the change condition, \( F M \) specifies the functionality modification, and \( Q M \) specifies the quality modification. The change model is depicted in Table 1.

#### 5.1 Condition

The condition \( C \) of a top-down change is a logical expression over a set of propositions in the domain of a SOE. Each proposition specifies a constraint on the occurrence of a change, either context-centric or application-centric. A context-centric constraint is related to an environment parameter, such as time and location. An example of such a proposition is that \( \text{it is during the Christmas week} \). The application-centric constraint is related to the domain parameters of the SOE, such as travel type and market interests in the travel domain. Examples of such propositions include: the travel type is international and there are less than five percent of the customers that take interests in this service.

#### 5.2 Functionality Modification

If a SOE’s owner wants to change the functionality of a SOE, his requests can be specified in terms of the functionality he would like to add, remove, or reconstruct. Therefore, we can describe functionality modification as: (1) add new functionalities, (2) remove some existing functionalities, (3) change the execution order of the member services, (4) the combination of the first three. As depicted in Table 1, \( F M \) consists of \( F^+ \), \( F^- \), and \( M_P \). \( F^+ \) is the set of functionalities that are intended to be added. \( F^- \) is the set of functionalities that are intended to be removed. \( F^+ \) and \( F^- \) are specified as target services in the format of a service ontology. A SOE owner can choose whether to use an existing service ontology to specify the target service or not. \( M_P \) is the set of constraints that are intended to be enforced on the member service’s execution order, as we defined in Section 4.

#### 5.3 Quality Modification

\( Q M \) consists of a set of quality constraints. It can be either the requirement on the quality of one or more member services. We specify a quality constraint as \( \{ S, i, r \} \), where \( S \) is a set of component services of a SOE, \( i \) specifies the quality parameter, and \( r \) indicates the requirement on these services with respect to \( i \).

**Example 5.1.** The top-down change specified in the running example can be modeled as:

- **Condition**: \( C := \{ \text{travel type=“international”} \} \)
- **Functionality modification**: \( F M := \{ \{ S_f \}, \phi, M \} \)
  - where, \( S_f \) is the ontology for the target taxi service,
  - \( M := \{ S_A \gg S_f; S_f \parallel S_H; S_A \leftarrow S_R \} \)
- **Quality modification**: \( QM := \{ \delta_1 \} \)
  - where \( \delta_1 := \{ S_A, S_H \} \), convenience Network, “\( \geq 5 \)”, meaning that the airline service and the hotel service should both provide the convenient internet service. The degree of the convenience should be rated above 5.

---

**Table 1. Top-down Changes Model**

<table>
<thead>
<tr>
<th>Top-down Changes</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>( C := { C_1 \theta_2 \ldots \theta_n, \theta \in { \vee, \wedge } } )</td>
</tr>
<tr>
<td>Functional Modification</td>
<td>( FM := { F^+, F^-, M_P } )</td>
</tr>
<tr>
<td>Quality Modification</td>
<td>( QM := { \delta_1, \delta_2, \ldots, \delta_n } )</td>
</tr>
</tbody>
</table>
6 Change Reaction

In this section, we propose a framework for change reaction. Considering the Web is expected to be populated by a large number of Web services, we assume that we can always find a solution to achieve the purpose of a top-down change in a SOE architecture. As depicted in Figure 3, change reaction takes place at two levels: schema level and concrete service level. Change reaction at schema level is to update a SOE’s schema based on F M in a change specification. It includes service matching, data redistribution, and process modification. The SOE’s schema is updated with aspects to the member service list, data flow, and control flow. Change reaction at concrete level is to select and compose concrete Web services. The selection is based on the updated SOE schema and the Q M in a change specification.

6.1 Service Matching

The first step of change reaction is to target the services that the SOE’s owner wants to add to or remove from the SOE. To specify a request, the SOE’s owner can choose either importing a service ontology from an ontology provider or creating a new one on his own. For the latter case, we need to make matching between the requested service and the provided one. By doing this, we can update the member service list of a SOE’s schema to follow the change specification.

We use “matching” to denote the relationship between two services, where a service can provide the similar functionality that another service offers. We make matching between two services by comparing two aspects: their observable behavior and the data they operate on. A service S1 matches another service S2 if two conditions can be satisfied: (1) S1’s process can simulate S2’s process. (2) S1 can generate S2’s output by using S2’s input. The first condition is to check the process simulation between two services. The second condition is to check the matching between two services’ data.

We use the simulation theory from \( \pi \)-calculus to compare the observable behavior between two services [6, 9]. Process Q simulates P if there is a binary relation B, such that whenever \( P \xrightarrow{\alpha(a)} P' \) and \( PBQ \) then there exists \( Q' \) such that \( Q \xrightarrow{\alpha(b)} Q' \) and \( P' B Q' \) and \( \text{type(a)} \leq \text{type(b)} \). \( \alpha \) is an observable action (either send or receive) with the data item \( (a, b) \). The definition says that Q simulates P if P’s behavior structure is a subset of Q’s, i.e., Q can mimic P’s transitions. For example, if \( P = a.\overline{b}.P \) and \( P' = a.\overline{b}.P' + a.\pi.P'|Q \), \( Q = \overline{b}.Q \), then \( P' \) simulates \( P \).

We then compare the ability of transferring data between two services. A service S1’s data matches that of a service S2’s if S1 can generate the output of S2 by using S2’s input. That is, \( OS_1 \supseteq OS_2 \) and \( IS_1 \supseteq IS_2 \).

Through the process of service matching, we can update the member service list as specified in a change specification. First, the service ontology that matches the target service in \( F^+ \) will be added to the member service list (We assume that there is only one service ontology that matches a targeted service ontology). We use \( S_m \) to denote this set of service ontologies. Second, the service ontology that matches the target service in \( F^- \) will be removed from the component service list. We use \( S_m \) to denote this set of service ontologies. We then get an updated list of the member services.

Modifying the member service list may compromise the correctness of the SOE’s schema. First, if a service has been removed, the services that need this service’s output will not get enough input to be invoked. Second, a new service has been added. It may not get enough input from its peers or the users. To address this issue, the next step is to fix the SOE’s schema by refining the data flow of a SOE.

6.2 Data Redistribution

The purpose of data redistribution is to achieve the correctness of a SOE’s schema by guaranteeing that each member service can receive the required input. To achieve this purpose, we need to update (i.e., remove or add) the data transferrers that involve the service \( s \), where \( s \in (S_m \cup S_a) \).

First, for a service \( s \) in \( S_m \), it should not be included in the data flow. Therefore, the data transferrer which involves the data from \( s \) will be removed. A new data transferrer will be generated to substitute the removed one. In our running example, the schedule for the hotel reservation service is derived from the flight information generated by an airline service. If the airline service is removed, the journey information from the user can be used as a substitute to provide the information for the hotel reservation service’s input. The substitute data transferrer can be predefined.

Second, for a service \( s \) in \( S_a \), the related data transferrers should be added to a SOE’s schema. They specify what information the SOE or other member services will provide for \( s \) and where the output of \( s \) will go to. These new data
transferrers can be predefined.

In our running example, when a taxi service is added, the following data transferrers will be added:

\[
\begin{align*}
\gamma_1 &: \{\text{accountInfoET}\} \rightarrow \{\text{loginInfo1}\}; \\
\gamma_2 &: \{\text{ticket_A}\} \rightarrow \{\text{flight}\}; \\
\gamma_3 &: \{\text{hotel_H}\} \rightarrow \{\text{hotel}\}; \\
\gamma_4 &: \{\text{pickup}\} \rightarrow \{\text{schedule_H}\}; \\
\gamma_5 &: \{\text{ticket}, \text{rental_H}, \text{pickup}\} \rightarrow \{\text{travelPackageET}\}; \\
\end{align*}
\]

A SOE may not always be able to find substitute information for the output of the removed services. It also may not always guarantee the input for the newly added services. This is because that a SOE can only transform information, not create information. Therefore, it is possible that a SOE cannot provide enough input for its component services. In this case, other new services should be automatically added to generate the required information.

Algorithm Data Redistribution

Input: The original member service list S, A set of added service ontologies Sa; A set of removed service ontologies Sr; A set of new data transferrers \( T' = \{\gamma_1, \gamma_2, \ldots, \gamma_n\} \).
Output: The updated member service list \( S' \); A new set of data transferrers \( T' = \{\gamma_1', \gamma_2', \ldots, \gamma_m\} \); A set of process constraints \( PC' = \{c_1, c_2, \ldots, c_n\} \).

1. \( S' = S; T' = T; PC' = \phi \).
2. For each \( s_i \in S \).
3. \( S_i' = S_i - s_i \).
4. \( T' = T' - \text{dataTransferers}(s_i) \).
5. For each \( s_j \in S \).
6. \( T' = T' + \text{dataTransferers}(s_j) \).
7. For each pair of services \( (s_i, s_j) \), where \( s_i, s_j \in S' \).
8. If \( s_i \) provides input for \( s_j \).
9. \( PC = PC + \{s_i \rightarrow s_j\} \).
10. If \( s_j \) provides input for \( s_i \).
11. \( PC = PC + \{s_j \rightarrow s_i\} \).

Figure 4. Algorithm of Data Redistribution

We present the Data Redistribution algorithm in Figure 4 to generate a complete list of member services and the data flow of a SOE. The constraints on the control flow are also produced. From Line 02 to 05, it deletes the transferrers that involve removed services. For the transferrers that have \( s \)'s output on the left side, a substitute will be picked up to form a new transferrer if such a substitute is available. From line 06 to 08, it adds the transferrers for each newly added service. If not, the service that can provide the required input will be located and added. From line 17 to 21, we add process constraints derived from the data flow to enforce the correctness of a SOE's. That is, if \( s_{i'} \) provides input for \( s_i \), \( s_{i'} \) should be invoked before \( s_i \). If \( s_i \) provides input for \( s_{i'} \), then \( s_{i'} \) and \( s_i \) should be invoked in parallel. In this algorithm, if a new service is added, a new service may be added to feed its input. Adding such a service may cause adding other new service to provide its input. However, such a ripple effect is not common in practice.

By applying the algorithm in Figure 4, we can get a complete list \( S' \) for component services of a SOE. The generated transferrers specify the data flow of the SOE, including where the input of users goes, how the data is transferred among the component services, and how the final output is generated. It also generates a set of process constraints which eliminate the conflicts between the control flow and the data flow.

6.3 Process Modification

The purpose of process modification is to react to the following situations: (1) the member service list of the SOE has been updated; (2) there are process constraints \( (MP) \) specified by a change specification; (3) there are process constraints generated during the data redistribution process.

We follow two steps to modify the process of a SOE. First, we generate the new process constraint set \( C \) by following the result of service matching and data redistribution. Let \( C_1 \) be the set of constraints deduced from the previous process expression. Let \( C_2 \) be the set of process constraints generated from data redistribution. Let \( C_m \) be the set of process constraints that involve the services from \( S_m \). Then the generated process constraint is \( C \), where \( C = (C_1 \uplus (MP \cup C_2)) - C_m \). Here, \( \uplus \) is a conflict-clear union, where if there is a conflict, the latter will be followed. Second, we generate the new process expression based on the new process constraints. There are five types of relationships between two services in a process constraint: "\(\rightarrow\)", "\(\rightarrow\)", "\(\leftarrow\)", "\(\rightarrow\)" and "\(\rightarrow\)", as defined in Section 5. The process expression can be generated based on the semantics of these constraints. This step is depicted in Figure 5.

As depicted in Figure 5, we generate a process expression based on a set of process constraints. From line 03-09, we integrate the related constraints. If a service \( s \) must be invoked (i.e., there is a "\(\rightarrow\)" in \( C \)), the services that are compatible with \( s \) should not appear in the process. From line 10-30, we then generate a process expression. We first pick up a service that has no antecedents in the rest service list. The services that can commute with it will then be picked up. After that, the service that are parallel with it will be picked up and their relationship will be added to the process expression.

By applying the algorithm ProcessGeneration, we can generate the new process expression for the running example as:

If \( \text{travelType} = \text{"international"} \), \( P_{\text{new}} \stackrel{\text{def}}{=} (S_A); (S_T | S_H) \). Otherwise, \( P_{\text{new}} \stackrel{\text{def}}{=} (S_A); (S_R + S_T | S_H) \).

Till now, we have shown how the SOE schema has been updated to react to the changes. The remaining work deals
with concrete services.

6.4 Reaction at the Concrete Level

Change reaction at the concrete level is to react to a top-down change in practice. The outcome is the SOE instantiation that will conform to the new SOE schema and the change model.

The first step of the change reaction at this level is to select the appropriate services based on their functionalities and performances. Let W be the set of concrete services of a SOE before a change occurs. A service w ∈ W will be deleted from W if its functionality has been removed from the updated SOE schema. A new service w′ will be added to W if its functionality has been added to the updated SOE schema. A change specification may also deliver the requirement on performance of member services. In this case, the services that fail to provide the desirable performance will be replaced with the ones that provide the better performance. This will continue till the performance requirement is fulfilled. After this step, W′, the new set of concrete service of the SOE, will be generated.

The second step is to follow the updated SOE schema to compose the participants of a SOE. The SOE schema defines the control flow and data flow between the participants. The control flow specifies the invocation order of the concrete services. The data flow specifies how data is transferred between the services. It can be achieved by applying the existing Web service composition techniques [5].

7 Conclusion

We have proposed a framework for managing top-down changes in a SOE. Changes are specified with respect to the functionality and quality of a SOE. The specification will be used as the basis for change reaction. Changes are reacted at the schema level and the concrete level. The correctness of a SOE is maintained during the change reaction process.

References