Abstract

A service model is the key work product of the service-oriented solution life cycle, in which localizing the effects of business changes could reduce the cost of evolution and maintenance. Practitioners in service-oriented computing strongly desire automated methods, tools, and environments that accommodate changing business requirements both cost-effectively and in an agile way. To achieve this goal, this paper proposes an automated approach for service model evolution named Automated Service Model Evolution Method (ASMEM), which enjoys two salient characteristics: a) it localizes the business model changes to the service model in a traceable manner; and b) it propagates incremental changes instead of re-transforming the entire changed business model. To achieve these characteristics, two steps are proposed: first, several types of change scenarios to the input business model are explored; and second, evolution management steps are provided for each scenario. In fact, these steps represent how these changes should be propagated to the service model, based on quantitative criteria. The paper also discusses the preliminary implementation of Automated Service-Oriented Modeling Tool (ASOM-Tool), which is developed to supply automation and traceability features for modeling and maintaining service-oriented solutions.

4.1 Introduction

Evolution is recognized as an extremely labor-intensive activity in the software life cycle (Boehm and Papaccio 1988), particularly in today’s competitive businesses. Because any system is exposed to many different change conditions, most of which emerge from the business itself (Bennett and Rajlich 2000, Lehman and Ramil 2003), they have to be flexible enough to encounter these changes, and dynamically adapting (Shaw and Clements 2006) to them would help to maximize the benefit to each user. Since service-oriented solutions have a cohesive relation with business (Arsanjani, et al. 2008, Wahl, et al. 2007), they have to be capable of accommodating unanticipated business changes cost-effectively and in an agile way. Therefore, the theme of maintenance and evolution of service-oriented systems is in the focus of today’s research (MESOA 2009).

Service-oriented modeling is the discipline of modeling business and systems for the purpose of designing and specifying service-oriented solutions within a service-oriented architecture (Jamshidi, Sharifi and Mansour, To Establish Enterprise Service Model from Enterprise Business Model 2008). According to IBM SOMA (Arsanjani, et al. 2008), the service-oriented modeling life cycle comprises three main phases: service identification, service specification, and service realization. In order to effectively and efficiently produce a service model, service modeling activities should be supplied with automation features in a systematic and model-driven fashion (Wahl, et al. 2007). In this regard, the authors proposed Automated Service Identification Method (ASIM) (Jamshidi, Khoshnevis, et al. 2009) and Automated Service Specification Method

The service model work product is the core artifact of service-based solutions and is used as an essential input to implementation and test activities. Therefore, localizing and applying the effects of business changes in the service model work product by leveraging design and dependency information (Hearnden 2007) could reduce the cost of maintenance. Due to this issue, there is truly a high motivation for developing an automated method for service model evolution. Therefore, the main objective of this research work is to investigate incremental (vs. re-transformation) change propagation through service modeling activities in the service-oriented solution life cycle, as an enabling mechanism towards automated service-oriented system maintenance and evolution.

In this regard, we propose an automated method for service model evolution, which is based on the notions and concepts of an automated service-oriented modeling life cycle through ASIM and ASSM. In other words, realizing a fully-automated service-oriented modeling life cycle requires another building block to support model evolution features, namely Automated Service Model Evolution Method (ASMEM). ASMEM enjoys two salient characteristics: a) it localizes the business model changes to the service model in a traceable manner; and b) it propagates incremental changes instead of re-transforming the entire changed business model.

Because any possible approach for fully-automated service-oriented modeling must be supported by a tool, we introduce the Automated Service-Oriented Modeling Tool (ASOM-Tool) (Nikravesh, et al. 2009) which provides a model-driven environment for capturing design and dependency information through the related models and corresponding transformations. Moreover, ASOM-Tool supports model evolution features, which can efficiently redesign the service model in case of business model changes. This means that by adopting the tool, consequential changes induced by the maintenance activities in the systems can be accommodated automatically, incrementally, and efficiently, reducing the cost of software maintenance and evolution.

Because limited research has occurred on the model-driven software evolution in the context of service-oriented solutions, we briefly introduce most relevant work in Section 4.2. Regarding our previous research work, several concepts which have been reused in this work are presented in Section 4.3. The position of model evolution in the service-oriented modeling framework, several change scenarios in service-oriented solution maintenance, our proposed solution to accommodate them, and its implementation in a tool are described in Section 4.4. The evaluation of the contribution has been conducted through a case study described in Section 4.5. Finally, the most well-known challenges of software evolution that we have tackled in this work are discussed in Section 4.6, and concluding remarks are presented in Section 4.7.

4.2 Related Work

While substantial research from both industry and academia has been devoted to software evolution and maintenance, there has been limited exploration of these concepts in service-oriented computing. In addition, although some techniques such as service identification, concept location, and service testing were introduced to support maintenance and evolution of service-oriented systems (MESOA 2009), work on automated model-driven software evolution in service-oriented systems is nonexistent in the literature, to the best of our knowledge. However, this section briefly reviews the closest work accomplished over decades in (automated) software evolution.
There are several techniques to accomplish (semi-) automated software evolution, ranging from heavyweight, formal methods (Yang and Ward 2003, Wiels and Easterbrook 1998), to lightweight, ad hoc approaches (Rajlich 1997). Some of the more well-established techniques include reengineering (Yang and Ward 2003), change impact analysis (Arnold 1996), techniques based on category theory (Barr and Wells 1990), and techniques based on meta-programming (Mens and Tourwe 2001, Tourwe and Mens 2003). The underlying foundation of these techniques is the proper exploitation of metadata. Reengineering is dominated by the recovery of design information, whereas change impact analysis depends heavily on dependency information. Category theoretic techniques rely on implicit categorical metadata, while meta-programming directly exploits metadata.

4.3 Basic Concepts

In this section, definitions of relevant terminology and its implications are presented. For more details, see (Jamshidi, Sharifi and Mansour 2008, Jamshidi, Khoshnevis, et al. 2009).

In order to utilize the business model in this work, the lowest level of its dimensions that are applicable in a conceptual view should be adopted. Referring to the literature of software engineering, an Elementary Business Process (EBP) is defined as “a process performed by one person in one place at one time which adds significant value and which leaves data in a consistent state” as the lowest level of enterprise business process. In addition, “an enterprise domain model should be decomposed to the extent that each entity is created only in one business process and used in the other needed processes”, and in literature, it is called business entity.

Definition 1: A Business Entity (BE) can be defined as $BE = \{n, A, R\}$, where $n$ is the name of the business object, $A$ is the set of attributes, and $R$ is the set of relationships between $BE$ and other business entities.

Definition 2: An Elementary EBP can be defined as $EBP = \{n, (BE_j, sr)\}$, where $n$ is the name of the elementary business process, $BE_j$ is the $j^{th}$ business entity which semantically relates to the corresponding EBP. $sr \in \{“C”, “R”, “U”, “D”\}$ is the type of semantic relationship between EBP and $BE_j$.

In the present work, in order to have an appropriate business model as an input of the method, we use the EBPs as rows and BEs as columns of the matrix, which is called a CRUD matrix\(^2\) (Jamshidi, Sharifi and Mansour 2008). Therefore, a cluster of the matrix could be representative of an abstraction level (Business Capability), in which EBPs act as their behavioral elements and its BEs act as structural elements. These clusters are business-aligned entities, and therefore are at a much higher level of abstraction than are objects and components.

Definition 3: A CRUD matrix can be defined as $M = \{(EBP_i, BE_j) i=1...\#row, j=1...\#column\}$, where $EBP_i$ is the $i^{th}$ EBP and $BE_j$ is the $j^{th}$ BE. $\#row$ is the number of EBP$s$ and $\#column$ is the number of BE$s$ in the model.

Many definitions have been proposed for services, but we have found the ones below to be proper. A service is a well-defined, encapsulated, reusable, business-aligned capability. A service op-
eration is the elementary part of a service and specifies the associated inputs, purpose (function, duty or obligations), and outputs (artifacts, products, outcomes, or deliverables) (Arsanjani, et al. 2008).

**Definition 4:** A software service can be defined as \( S = \{n, I, Msg, RS\} \), where \( n \) is the name of the service, \( I \) is the set of operations associated with \( S \), \( Msg \) is the set of messages corresponding to \( S \), and \( RS \) is the set of relationships between \( S \) and the other services in the service model. Referring to the CRUD matrix, \( I \subseteq EBP \) is the set of corresponding EBPs, and \( Msg \subseteq BE \) is the set of corresponding BEs (Wiels and Easterbrook 1998).

An enterprise service model is a model of the core elements of a SOA and is used as an essential input to activities in implementation and testing of software solutions. The service model is an abstraction of the software services implemented within an enterprise and supporting the development of one or more service-oriented solutions. It is a comprehensive and composite artifact encompassing all services, providers, specifications, partitions, messages, collaborations, and the relationships between them (Arsanjani, et al. 2008).

**Definition 5:** An Enterprise Service Model can be defined as \( ESM = \{S, Pr, Spec, Prt, Msg, R\} \), where \( S \) is the set of services, \( Pr \) is the set of providers of the services, \( Spec \) is the set of specifications in the form of contract between what consumers need and what providers provide. In addition, \( Prt \) is set of partitions in order to manage the solution, \( Msg \) is the set of messages forming the underlying data model of the solution, and \( R \) is the set of relationship between services.

**Definition 6:** Suppose we have identified \( n \) services \( \{S_1, S_2, \ldots, S_n\} \) from a business model and form service set \( S \). We can use the average of each technical metric \( TM_j \) as an objective function of the problem as follows:

\[
\frac{1}{n} \left( \text{Max} \left( \sum_{i=1}^{n} TM_j(S_i) \right) \right), \quad \frac{1}{n} \left( \text{Min} \left( \sum_{i=1}^{n} TM_j(S_i) \right) \right)
\]

The consolidated objective function of the problem can be the multiplication of the individual objective functions. Therefore, the service identification problem (SI) can be mathematically delineated as follows:

\[
SI: EBM \rightarrow ESM
\]

\[
S = \{S_1, S_2, \ldots, S_n\}
\]

\[
\text{Max } Z(S),
\]

\[
Z = \prod_j \sum_{i=1}^{n} TM_j(S_i) / \prod_j \sum_{i=1}^{n} TM_j(S_i)
\]

We have adopted three technical metrics to derive the appropriate objective function represented as follows. The comprehensive description of how to derive the \( Z \) is presented in (Jamshidi, Khoshnevis, et al. 2009).
where \( G(S), \text{Cohesion}(S), \text{Coupling}(S) \) stand for granularity, cohesion, and coupling of the service \( S \) respectively.

**Definition 7:** In software engineering literature, the terms *evolution* and *maintenance* are virtually synonymous. Occasionally, maintenance refers only to particular kinds of evolution, but, in general, the two terms are used interchangeably (Bennett and Rajlich 2000, Hearnden 2007). We shall use maintenance to refer to general post-delivery activities, and evolution to refer to consequential change. A change to a model in a model-driven system may require other changes to be made to other models.

**Definition 8:** The obvious approach for effecting consequential change induced by a transformation relationship \( t = T(s) \) is simply to execute the transformation again, regenerating the output models, as shown in Figure 4.

![Figure 4  Re-Transformation (Hearnden 2007)](image)

A fundamental limitation of *re-transformation* strategies is that they require a complete re-execution of the transformation. In contrast, *incremental* strategies address this problem by handling the consequential change directly. Where the focus of merged re-transformation is the difference of the transformation outputs, the focus of an incremental strategy is the transformation of the difference of the inputs, as shown in Figure 5.

![Figure 5  Incremental Change Propagation (Hearnden 2007)](image)

### 4.4 Evolution Management in our Service-Oriented Modeling Framework

In this section, we will provide our solution for automating management of service model evolution in our service-oriented framework. For this purpose, we will briefly discuss evolution management issues in model-driven systems and the position of ASMEM in our framework. Thereaf-
ter, we will provide main evolution scenarios and steps on how to manage each scenario in ASMEM, and the tool support (ASOMT) will be explained.

4.4.1 Evolution in Model-Driven Systems

A change to a model in a model-driven system may propagate to other models, which is referred to as consequential change (Hearnden 2007). If the reason for a consequential change can be explained objectively (that is, without domain or semantic knowledge), then there is the potential to automate its derivation and application. The only objective reasons for consequential change in a model-driven system are the preservation of meta-model and transformation relationships. Any other kind of consequential change requires specialized domain knowledge, and thus cannot be automated without that knowledge.

In general, meta-model relationships cannot be usefully maintained automatically; however, their maintenance may be assisted by automation. But the maintenance of transformation relationships is completely automatable because they have machine-interpretable semantics.

As mentioned in Definition 8, there are non-incremental maintenance strategies such as retransformation, as well as incremental strategies such as delta-transformation (Hearnden 2007). The latter was used as a basis for model evolution management in ASMEM.

4.4.2 Model Evolution in the ASOM Framework

Using traditional solution development environments to develop service-oriented solutions usually causes that architects who work directly with the business models to get involved in deriving the solution models by utilizing prescriptive guidelines such as SOMA (Arsanjani, et al. 2008) as depicted in Figure 6. In this regard, the authors proposed Automated Service-Oriented Modeling Framework (ASOMF) (Jamshidi, Khoshnevis, et al. 2009), which consists of a cohesive assembly of methods, techniques, tools, and content in order to support service-modeling activities, from business modeling to the generation of implementation artifacts. ASIM, ASSM, and ASOM-Tool are parts of this framework.

![Figure 6 Service-Oriented Solution Life Cycle without Using the ASOM-Tool](image)

ASIM is used to automatically identify the candidate services set as the main architectural elements of the service model work product, with respect to specified technical metrics. In this me-
method, service abstractions with acceptable technical metrics can be derived automatically from high-level business requirements and business process models. We define the CRUD matrix as the desired abstraction level. A cluster (Jamshidi, Sharifi and Mansour 2008) of the matrix would represent a business-aligned service. In this method, each candidate service would be identified based on an objective function named \( Z \). (For more details, see (Jamshidi, Khoshnevis, et al. 2009).)

The ASSM method is intended for specifying the service model by deriving service model elements from the candidate services set.

In order to fully automate the life cycle, there is a need to introduce another method that is responsible for model evolution management. For this purpose, we propose the Automatic Service Model Evolution Method (ASMEM) in this paper.

ASMEM provides steps for handling every evolution scenario. The evolution scenarios are based on six categories of changes that may occur in the input model. This method and its implementation will be fully discussed in Sections 4.4.3 and 4.4.4 respectively.

ASOM-Tool provides an integrated environment that supports service modeling activities by providing automated features throughout the solution lifecycle. Its architectural model is depicted in Figure 7.

![Figure 7 ASOM-Tool Architecture](image-url)
4.4.3 Managing Evolution Scenarios in ASMEM

In order to enable automated software evolution, it is essential to understand what types of modifications are carried out in evolutionary tasks, rather than why those changes have been performed (Hearnden 2007).

Since ASIM and ASSM are based on the CRUD matrix, changes affecting its rows and columns lead to changes in services. Therefore, if EBPs or BEs of an enterprise are changed, the enterprise candidate service set should be identified again. The set of candidate services is crucial for generating and dealing with the service model in the ASSM stages (Jamshidi, Khoshnevis, et al. 2009). One solution to this problem is applying ASIM to the modified CRUD matrix; however, this approach takes much time and is costly. Another approach to solve the problem is categorizing possible changes in the CRUD matrix and proposing steps to reconfigure the candidate service set based on the type of the occurred change. Using the CRUD matrix, changes are located in the source model, and then using model transformations in the ASSM, the changes are propagated to the service model, as shown in Figure 8.

![Figure 8](image)

Figure 8 Changes are Located in the CRUD Matrix and Then Propagated Into the Service Model Through Transformation

There are six possible categories of changes in the CRUD matrix rows and columns. We have proposed steps to face these changes with proper reconfiguration of the candidate service set for each category as follows:

1. **Adding a new EBP to the CRUD matrix.** When a change to the business processes occurs in terms of adding a new EBP, we can handle it by considering it as a new operation for one of the identified enterprise services. For this purpose, we consider a new row added to the CRUD matrix and based on that, we check how many BEs are common between the new EBP and each of the services—which are clusters in the CRUD matrix. The service with the most common BEs will be the one to which the new EBP is added as a new operation, only if the objective function of the service is greater than the specified threshold.
2. **Adding a new BE.** A change to the business processes may occur in terms of adding a new BE. In this case, a new column is added to the CRUD matrix and should be positioned in a proper place somewhere between the other columns, where it maximizes the objective function (Z). Then based on the objective function one of the following actions may be taken:
   - The new BE is added to the service which is placed as a cluster at its northwest.
   - The new BE is added to the service which is placed as a cluster at its southeast.
   - A new service is defined to cover the new BE.

3. **Adding a new EBP and a new BE simultaneously.** This change can be applied to the CRUD matrix in two steps: the first step is adding a new BE to the CRUD matrix, and the second is adding a new EBP to it. Adding a new EBP can be influenced by the result of adding a new BE—that is why we first add the BE and then the EBP.

4. **Deleting a BE.** To delete a BE from the CRUD matrix, the first step is to verify the service which contains the BE. After deleting the BE, if there are still "C" actions in the service that uses the BE, we can simply delete the BE from the CRUD matrix. But if deleting the BE from matrix causes deleting all "C" actions of the service, the service no longer exists and based on Z, one of the three following actions would be taken:
   - Cells that remain after deleting the service would be added to the service placed in the northwest area of the matrix.
   - Cells that remain after deleting the service would be added to the service placed in the southeast area of the matrix.
   - Cells that remain after deleting the service would be considered as service channels between remaining services only.

5. **Deleting an EBP.** If an EBP includes a “C” action on a BE, then deleting the EBP causes deletion of that BE. Therefore, to delete an EBP from the CRUD matrix, the first step is to delete the BEs that would be deleted as a result of deleting the EBP. Then, the steps for the fourth category mentioned above can be applied to reconfigure the matrix. Finally, the EBP can be deleting resulting in the deletion of an operation of a service.

6. **Deleting a BE and an EBP simultaneously.** This change would be applied to the CRUD matrix as first deleting the EBP and then deleting the BE from the matrix.

### 4.4.4 Evolution Management in the Tool Architecture

We developed the ASOM-Tool (Nikravesh, et al. 2009) to implement the methods and guide service-oriented practitioners to design service-oriented solutions based on the mentioned service modeling methods in a traceable, reusable, configurable, and systematic way. The ASOM-Tool is a product, which is developed on top of the Eclipse Modeling Framework (EMF) to provide automation features for modeling service-oriented solutions. This product helps system architects to automatically accomplish the service-oriented modeling activities.

We used the ADD method (Bass, Clements and Kazman 2003) for creating the architecture of the ASOM-Tool. The architectural blueprint of the ASOM-Tool is shown in Figure 7.

There are four key modules located around a common artifact repository for storing intermediate and final results. This tactic helps us to keep track of the outputs of each module and facilitates
the integration of different modules, which increases the modifiability of the architecture. Responsibilities of the key modules are as follows:

- **User Management** manages user interaction with the system. There are two sub-modules, which are “user interface” and “report generator.”
- **Validation Management** is responsible for quantitative analysis of produced models, such as the service model and its elements.
- **Evolution Management** enables links between the solution artifacts (such as services and service components) and changing business models.
- **Method Management** implements the algorithms for ASIM, ASSM, and ASRM [10] according to the solution life cycle depicted in Figure 9.

**Figure 9 Using ASOM-Tool in the Service-Oriented Solution Life Cycle**

The evolution management module deals with the common artifact repository and the method management module. The dependency between the evolution management module and the common artifacts repository refers to the need to maintain intermediate and final model evolution artifacts in a common environment that enables the tool to share them with other modules. The dependency between the evolution management module and the method management module is defined in terms of dependencies between the Model Evolution (ME) and the sub-modules inside the Method Management module. ASIM and ASSM use the CRUD matrix as one of their inputs; therefore any change in the CRUD matrix directly affects them.

The ASOM-Tool’s strategy for the incremental change process is an extension of a transformation engine (Method Management Module) called live transformation. Once the initial transformation has been completed, the engine listens for changes to the input models (business model), and maps them directly to changes in the output models (service model). Live transformation propagates the updates incrementally and only re-computes the affected parts of the transformation.

Using the model evolution capability adds several new functionalities to the basic ASOM-Tool:

- Adding new services to the service model.
- Removing an available service from the service model.
- Adding to or removing operations from services.
• Adding to or removing parameters (messages) from operations.

It is clear that these new functionalities increase flexibility and maintainability in reference to the service model base elements, which are mostly services and their operations and messages. Other service model elements are mostly derived once the services and their operations and messages are determined.

4.5 A Case Study

Verification of the proposed steps was done by studying examples and performing case studies, one of which is described below.

The selected enterprise for the study is the sales department of a sales and goods distribution company. In order to simplify the problem description and its scope, we limited the business scenario to main and general (and yet real) issues, and ignored very detailed exceptions.

The department receives orders in a daily manner from mostly regular and some casual customers. The orders are placed by a role: "salesperson." The salesperson delivers the orders to the sales-in-charge, who then passes the orders on to the sales clerk for processing. Processing orders includes checking customer's credit information, checking discounts for the goods ordered (if any), checking inventory, and calculating the final order price. When the customer approves the final price, the order is ready to be shipped. The storekeeper issues a receipt, having delivered the goods to the customer. The financial department representative issues a financial “notes receivable” when a payment is made.

Our business model is represented through a CRUD matrix and is clustered by means of ASIM (Jamshidi, Khoshnevis, et al. 2009), as shown in Table 6. We are going to apply changes in terms of adding and removing EBPs and BEs to and from the matrix.

**Table 6 Original CRUD Matrix**

<table>
<thead>
<tr>
<th>Action</th>
<th>Customer</th>
<th>Credit</th>
<th>Account's Receivable Note</th>
<th>Discounts</th>
<th>Inventory</th>
<th>Warehouse Voucher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a customer</td>
<td>C</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add an accounts receivable note</td>
<td>R</td>
<td>U</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive order</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive order</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calculate discounts</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Check inventory</td>
<td></td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add discounts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>Add an item</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>Add a warehouse voucher</td>
<td>R</td>
<td></td>
<td></td>
<td>U</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>
In order to explain how the steps are applied, two examples for the first and the second change categories are provided. Suppose that we add a new EBP to this matrix as “Calculate price” (a change that fits in the first category mentioned in Section 4.4.3). The new matrix is shown in Table 7 based on the first category of steps in Section 4.4.3. The first step to handle this change is to place the new EBP in the right row of the CRUD cluster. As seen in Table 7, the new EBP has only one BE in common with Service 2. Therefore Service 2 is the first candidate for adding the new EBP. Also, the value of $Z$ (represented in the second row of Table 8) for the new service is greater than the threshold (in order to avoid complicated computations, we have omitted the quantitative value of the parameters); therefore, this EBP will be added to Service 2.

**Table 7 Adding an EBP to the CRUD Matrix**

<table>
<thead>
<tr>
<th>Add a customer</th>
<th>Customer</th>
<th>Credit</th>
<th>Accounts Receivable Note</th>
<th>Discounts</th>
<th>Inventory</th>
<th>Warehouse Voucher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add an accounts receivable note</td>
<td>R</td>
<td>U</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive order</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calculate discounts</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add discounts</td>
<td></td>
<td></td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calculate price</td>
<td></td>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add an item</td>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add a warehouse voucher</td>
<td>R</td>
<td></td>
<td></td>
<td>U</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

**Table 8 Values of $Z$ for the Categories of Change Scenarios**

<table>
<thead>
<tr>
<th>Change Category</th>
<th>Z for Service 1</th>
<th>Z for Service 2</th>
<th>Z for Service 3</th>
<th>Total Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>89.55</td>
<td>19.56</td>
<td>23.0</td>
<td>1222.39</td>
</tr>
<tr>
<td>1</td>
<td>89.55</td>
<td>1.30</td>
<td>1.0</td>
<td>196.51</td>
</tr>
<tr>
<td>2a</td>
<td>121.5</td>
<td>2.48</td>
<td>19.6</td>
<td>547.3</td>
</tr>
<tr>
<td>2b</td>
<td>43.74</td>
<td>30.375</td>
<td>19.6</td>
<td>832.94</td>
</tr>
<tr>
<td>3</td>
<td>43.74</td>
<td>30.375</td>
<td>19.6</td>
<td>832.94</td>
</tr>
<tr>
<td>4</td>
<td>43.74</td>
<td>2.41</td>
<td>0.53</td>
<td>277.97</td>
</tr>
<tr>
<td>5</td>
<td>43.74</td>
<td>45.32</td>
<td>omitted</td>
<td>362.06</td>
</tr>
<tr>
<td>6</td>
<td>43.74</td>
<td>45.32</td>
<td>omitted</td>
<td>362.06</td>
</tr>
</tbody>
</table>

Now suppose we add a new BE, “Order,” to the matrix in Table 7 (a change that fits in the second category). The new matrix will be the one shown in Table 9. The new BE should be placed in the right column of the CRUD matrix based on its “C” semantic relationship. As you can see, “creation” of the new BE is done by the “Receive order” elementary business process. Thus, this BE will be placed on the left side of Service 2.
As mentioned in Section 4.4.3, based on value of Z, the new BE would be added to Service 1 or Service 2. The third and fourth rows of Table 8 represent Z values, in the case of adding the new BE to Service 1 or Service 2. Therefore, by considering Z values, the new BE is associated with Service 2 which is depicted in Table 9 in green.

Suppose other changes include adding a Calculate Price EBP and an Order BE to our original matrix simultaneously (category 3), deleting the Add an item EBP (category 4), deleting the Warehouse Voucher BE from the matrix (category 5), and deleting the Add an item EBP and Warehouse Voucher BP simultaneously (category 6). Z values of these steps are depicted in the fifth to eighth rows of Table 8. Also, the first row of Table 8 illustrates the initial objective function values.

### Table 9  Adding a BE to the CRUD Matrix

<table>
<thead>
<tr>
<th></th>
<th>Customer</th>
<th>Credit</th>
<th>Accounts Receivable Note</th>
<th>Order</th>
<th>Discounts</th>
<th>Inventory</th>
<th>Warehouse Voucher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a customer</td>
<td>C</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add an accounts receivable note</td>
<td>R</td>
<td>U</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive order</td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive order</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calculate discounts</td>
<td></td>
<td></td>
<td></td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Check inventory</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>Add discounts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>Calculate price</td>
<td></td>
<td></td>
<td></td>
<td>R</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add an item</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add a warehouse voucher</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>U</td>
<td>C</td>
</tr>
</tbody>
</table>

### 4.6 Discussion of Research Challenges

Software evolution processes have a number of well-known challenges, which have been identified and classified by researchers during the last two decades. Every new method in software evolution must contribute to alleviating some of these challenges to decrease the cost of evolution and maintenance. Thus, this section enumerates some of the well-known challenges and clarifies how ASMEM contributes to handle them.

The enumerated challenges in this section are derived from previous research. In Mens, Wermelinger, et al. 2005, Bass, Clements and Kazman 2003) Mens et al. classify some of the important challenges in software evolution that were identified during a workshop on Challenges on Software Evolution (ChaSE 2005).
• Common software evolution platform
  The challenge is to develop and support a common application framework for doing joint software evolution research. Because Eclipse has the advantage of visibility, industrial acceptance and also reusability of certain components, and the ASOM-Tool was built using the Eclipse platform, this challenge is thoroughly met.

• Support for model evolution
  Evolution techniques should be raised to a higher level of abstraction than higher level artifacts such as analysis and design models, software architectures, requirement specifications, and so on. In this regard, ASMEM localizes the business model changes to the service model, which is a high-level artifact. In fact, business changes mostly and directly affect business process models. Because the service model is based on the business process model, any possible business process model modifications immediately influence the service model. Therefore, the challenge of supporting model evolution is supported by ASMEM.

• Support for co-evolution
  This challenge refers to the need to achieve co-evolution between various types of software artifacts or diverse representations of them. ASMEM propagates changes from the design model (i.e., service model) down to the lower level phases of the life cycle (e.g., specification or realization, among others). Therefore, co-evolution between different types of artifacts or different representations of them is achieved.

• Need for improved predictive models
  This challenge refers to the point that some models are necessary for predicting a variety of things, including where the software evolves, how it will evolve, the effort and time that is required to make a change, etc. In ASMEM, every possible change scenario is identified and handled. Moreover, in ASMEM, every change in the business model can be traced down to the other artifacts, such as the service model.

4.7 Conclusions and Future Work

We have provided a novel method, called ASMEM, which consists of several steps based on quantitative criteria to support automated evolution management in service-based solution life cycles. Because we have adopted a well-defined matrix as the business model, we are able to automatically localize the several types of change scenarios and evaluate incremental changes. In addition, because we have developed automated methods for service modeling, we can automatically propagate the incremental changes to the service model. Moreover, we have developed a tool to implement the service modeling and evolution management methods to supply automation and traceability features to the provided framework. By adopting this framework, changes induced by the maintenance activities in service-oriented systems can be treated automatically, incrementally, and efficiently, thus reducing the cost of maintenance. The method has been verified via a case study.

Future work for the ASOM-Tool can be divided into two main categories: improving the techniques used within the ASOM-Tool, and increasing the ASOM-Tool’s utility by expanding its scope.
References


