Migrating Legacy Information Systems to Web Services Architecture

Shing-Han Li, Tatung University, Taiwan
Shi-Ming Huang, National Chung Cheng University, Taiwan
David C. Yen, Miami University, USA
Cheng-Chun Chang, National Chung Cheng University, Taiwan

ABSTRACT

150 words or less

Keywords: legacy system; information system reengineering (ISR); service component; service-oriented architecture (SOA); Web services

INTRODUCTION

Due to the dynamic advancement of Information Technology (IT), the life cycle of the information system (IS) is greatly reduced to a certain extent. Generally speaking, the traditional legacy information systems possess such undesirable characteristics as latency of information, poor reach, inflexibility, and higher cost of maintenance. Furthermore, the traditional system architectures such as centralized and client/server are frequently incompatible with the requirements and specifications which exist in today’s business environment. To be more specific, the legacy information systems have these aforementioned shortcomings,
which have prevented the businesses and/or organizations to react/respond dynamically to the rapid challenges as they should. Consequently, enterprises have a strong need to utilize the technology of information system reengineering (ISR) to preserve the value of their legacy systems.

In this situation, enterprises or software companies are always in a dilemma of redeveloping/redesigning their legacy systems to include the newer Web services components (Bouguettaya, Malik, Rezgui, & Korff, 2006; Chen, Zhou, & Zhang, 2006; Kim, Sengupta, Fox, & Dalkilic, 2007). Discarding and redeveloping the existing systems not only wastes the money allocated for software investments, but also causes organizations to lose competitive advantages to meet numerous unanticipated contingencies and/or uncertainties.

Based on prior study (Ommering, 2005), the system migration will be one of the best ways to reengineer a legacy system. Traditionally, there are two approaches available to migrate the legacy system to the Web services architecture (Vanston, 2005). The first approach is the legacy externalization approach. This approach is usually the main alternative available on the current market. It generally uses strategic or pointed forms, along with new types of interface display, to develop the integrated products (such as “Web Scraping”). The other approach is the component encapsulation approach. This is another viable alternative to utilize the component standard technology like Common Object Request Broker Architecture (CORBA) (OMG, 1995; Vinoski, 1997), Component Object Model (COM) (Microsoft, 2007), or Enterprise Java Beans (EJB) (Sun, 2007) to encapsulate the legacy system into the components, and then translate them into a Web Services standard. Ultimately, this second approach is migrated to the component-based and transaction-oriented framework (such as IBM WebSphere and BEA WebLogic) (Liu, Fekete, & Gorton, 2005; Waguespack & Schiano, 2004). Both of the aforementioned approaches may not be a bad way for the legacy system to migrate into the equivalent Web services standards. However, they normally utilize the hard-cордин technique to implement the interface with the corresponding standard (Brereton & Budgen, 2000; Kwan & Li, 1999; McArthur, Saiedian, & Zand, 2002). Being a traditional structure program, the system normally has a shorter life cycle and lacks scalability, feasibility, and reusability. Further, it would be much more difficult to maintain in the future. On the other hand, if a company is applying the component encapsulation approach without incorporating appropriate component migrating methods, the system still has these aforementioned shortcomings (Rahayu, Chang, Dillon, & Taniar, 2000). Unfortunately, most alternatives adopted now by enterprises and/or businesses do not use the proper component migrating method.

Many related studies (Erickson, Lyytinen, K., & Siau, 2005; Fong, Karlapalem, Li, & Kwan, 1999; Gall, Klosch, & Mittermeir, 1995; Kwan & Li, 1999; Sang, Follen, Kim, & Lopez, 2002) have presented methods that can be utilized to systematically reengineer the legacy system into the Object-Oriented (OO) or the distributed system. However, the Web Services architecture by nature is different from a general distributed system. The core concept of Web Service is a Service-Oriented Architecture (SOA) (Huang, Hung, Yen, Li, & Wu, 2006; Stal, 2002). In the SOA environment, resources in a network are made available as an independent service
that can be accessed without any knowledge of the underlying platform implementation (Erl, 2005). Web services can certainly rely on a Web-services composition language (WSCL) such as the Business Process Execution Language for Web Services (BPEL4WS) (IBM, BEA Systems, Microsoft, SAP AG, & Siebel Systems, 2002) to transform an existing Web service into a new type of Web service by employing well-defined process modeling constructs (Chen, Hsu, & Mehta, 2003; Curbera, Khalef, Mukhi, Tai & Weerawarana, 2003), enterprise architects believe that SOA can help businesses respond more quickly and cost-effectively to fast-changing market conditions (Sutor, 2006). This style of architecture, in fact, promotes reusability at the macro level (service) rather than the micro levels (objects). By doing so, it can greatly simplify interconnection to and usage of existing IT (legacy) assets (Carroll & Calvo, 2004).

The purpose of this study is to propose a methodology which utilizes the existing data design of an information system to migrate the legacy system to an SOA system. The benefits include the following items: First, this approach has the advantage of reengineering the legacy system to various system components from a technical aspect, and use the Web services composition language (WSCL) to translate the existing model into the new Web services architecture. Unlike traditional object-oriented approaches, this proposed methodology migrates the old systems to services-oriented or functional-oriented ones. Additionally, this approach can be employed to develop a system which will be more flexible and adaptable to fit better to the constantly-changing business environment. Furthermore, this proposed approach will no doubt make the conversion process easier to integrate with other additional applications.

The remainder of this article is organized as follows: The second section provides a brief overview of some legacy systems’ reengineering and Web services approaches. The proposed legacy system’s migrating approach and the implementation of a prototype will be discussed in the third section. The fourth section contains the system implementation using simulation and a real case study. A comparison of the proposed approach with others is provided in the fifth section. Finally, the sixth section concludes this manuscript.

LITERATURE REVIEW

Information Flow for Business Process (IFBP)

Business Process Management (BPM) is one of the basic elements of Web services architecture (Basu & Kumar, 2002). It can be decomposed of two major phases—process design and diagnosis. The process diagnosis phase will discover an entire picture of the business process for an enterprise, also known as the AS-IS Model. The exploration of AS-IS Model is a very time-consuming and experience-oriented task. As a result, an enterprise has to spend a lot of time and pay huge labor costs during the process diagnosis phase. Besides, it is difficult for the process designers to transform one process model to another equivalent one. There are certain gaps among different process-designing methods. Thus, the study of Shi-Ming Huang and Fu-Yun Yu (2004) investigates a novel methodology for business process discovery based on information flow, called IFBP. This IFBP methodology includes the following three
phases: transformation, integration, and conversion phases. The input of IFBP is actually dataflow diagrams (DFD) for the existing information systems. The output is an Event-Driven Process Chain (EPC) diagram for enterprise process flow.

Information System Reengineering
There are several direct and indirect system migrating approaches.

Direct Migration
Sang et al. (2002) presented an approach to integrate legacy applications written in FORTRAN into a distributed object-based framework. FORTRAN codes are modified as little as possible when being decomposed into modules and wrapped as objects. They used object-oriented technique such as C++ objects to encapsulate individual engine components as well as the CORBA, and implement a wrapper generator, which takes the FORTRAN applications as input and generates the C++ wrapper files and interface definition language file.

Serrano, Montes, and Carver (1999) presented a semiautomatic, evolutionary migration methodology that produces an object-based distributed system for legacy systems. They first used ISA (Identification of Subsystems based on Associations), a design recovery and subsystem classification technique to produce a data-cohesive hierarchical subsystem decomposition of the subject system. Second, they adapted the subsystems to develop the object-oriented paradigm. Third, they wrapped up and defined interfaces of the subsystems in order to define components. Finally, middleware technologies for distributed systems were used to implement the communication between components.

Indirect Migration
Gall et al. (1995) proposed an approach, which re-architect the old procedural software to an object-oriented architecture. The transformation process was developed to identify potential objects in the procedural source codes in order to enable the utilization of object-oriented concepts for future, related software maintenance activities. This approach was not directly performed on the source-code level, but instead, different representations were developed out of the procedural program on higher levels of abstraction (e.g., structure charts, data flow diagrams, entity-relationship diagrams, and application models), which represent different kinds of program information. Additional application-domain knowledge was introduced by human experts to support the program transformation to enable several higher-level decisions during the development process.

Kwan and Li (1999) proposed a methodology to reengineer those previously-developed relational database systems to OO database systems. Their approach is based on the input of: (1) Extended Entity Relationship model (EER) that provides rules for structuring data; (2) Data Dictionary that provides static data semantics; and (3) DFD that provides dynamic data semantics. This approach captures the existing database design, uses knowledge in OO modeling, and then represents them by means of production rules, to build the pattern extraction algorithm that is applied to perform the data mining process to identify the “data dependency of a process to an object”. The existing Data Dictionary, DFD, and EER model are all useful and hence, are needed for capturing the existing database design to recover the hidden dynamic semantics.
Huang et al. (2006) proposed a methodology that focused on how to migrate legacy systems with a well-structured OO analysis to ensure the quality of a reengineered component-based system. This research adopted the well-structured object-oriented analysis to improve the quality of reengineered systems. The result of the reengineered system will be a Web-enabled system. The proposed migration approach discusses how to process a well-structured object-oriented analysis. The research considered the following four factors—(1) multi-value attributes, (2) inheritance relationships, (3) functional dependency, and (4) object behavior to ensure the quality of a reengineered component-based system. Further, their study considered the migration from three aspects—data, process, and user interface to make the applied reengineering process more complete. The comparison of all aforementioned reengineering approaches is shown in Table 1, which is shown below. In this comparison table, it is noted that the approach proposed by Huang et al. (2006) could present a more semantic legacy system with a higher quality of components.

**Business Component and Service Component Definition**

Business components typically emulate a specific business entity, such as a customer, an order, or an account. When sets of coherent components are linked together, they can inter-operate to accomplish an even higher level of business function (Herzum & Sims, 1998; Jin, Urban, & Dietrich, 2006; Lee, Pipino, Strong, & Wang, 2004; Vitharana & Jain; 2000; Zhao & Siau, 2007). The scope of a component discussed in this article will include the class and its related interfaces together as a component. By doing so, it can preserve the simple execution function of a component from getting over-complicated. Services are self-describing, open

---

**Table 1. Comparison of the reengineering approaches**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Approach</td>
<td>Direct Migration</td>
<td>Direct Migration</td>
<td>Indirect Migration</td>
<td>Indirect Migration</td>
<td>Indirect Migration</td>
<td></td>
</tr>
<tr>
<td>How to find objects</td>
<td>codes</td>
<td>codes</td>
<td>ERD</td>
<td>ERD</td>
<td>ERD</td>
<td></td>
</tr>
<tr>
<td>How to find methods</td>
<td>codes</td>
<td>Codes</td>
<td>Codes</td>
<td>DFD</td>
<td>DFD</td>
<td></td>
</tr>
<tr>
<td>Is component-based</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>User Interface Consideration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quality of Object-Oriented Model</td>
<td>Aggregation Consideration</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Inheritance Consideration</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Object Behavior Consideration</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Multi-value attributes consideration</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
components that can support rapid, low-cost composition of distributed applications (Papazoglou & Georgakopoulos, 2003). Service-Oriented components attempt to fulfill users’ requirement. Consequently, the service providers’ responsibility is mainly to design/develop the most adaptable service processes and components for different users. The users just enjoy the content and quality of the provided service, but do not care about who the true service provider is and how to acquire the service.

Some definitions of service argue that they can be implemented by components (Sprott, 2002). However, in a complex environment, a service can actually include several components (Perrey & Lycett, 2003). Therefore, this article considers that the service is comprised of a number of system components, which could simultaneously interact and integrate with each other. Enterprise has a tendency to keep distance from the composition of too many small and trivial system components and hence has a strong reservation about the operation and management of each other while applying the services. For this reason, this research defines the service to be composed of system components from the perspective of users’ requirements. A service-oriented feature should be an extension of the component-oriented one, and should be aimed at satisfying the user’s situation.

MIGRATING METHODOLOGY

By nature, Web services are different from legacy systems in terms of architecture and the degree of coupling. To migrate the system from a tightly-coupled one or from a loosely-coupled one, the best strategy is to apply OO reengineering. It is noted that the complete Web services architecture generally utilizes the Web Services Composition Language (WSCL) to construct the service components. Most available WSCL are designed based on business process. From the business perspective, most approaches available today merely apply business process to represent the legacy system. To this end, this article will perceive that the business process is one of the main elements of Web services. From the above discussion, this research analyzes the legacy system from two aspects: technical and business. This approach, as visualized in Figure 1, is summarized in five subsequent steps. Traditionally the system designers use the tools and techniques such as ERD, DFD, and user interfaces to describe the entire architecture and then design/develop the information system. These aforementioned models have been applied to analyze the business process of the legacy system and then have been reengineered to the corresponding system components. This research reengineers the legacy system to various system components from the technical aspect, analyzes the model of the legacy system to attain the business processes from the business aspect, integrates the system components and the business processes together, and then analyzes the service based on processes. Finally, according to this proposed integrated model, it translates the model to the Web services composition language to build a Web services architecture. The following sections discussed below describe each of the five steps of this proposed approach.

Step 1: Reengineer System Components

To migrate the system to a loosely-coupled architecture, the OO reengineering approach could be utilized. In the comparison table (Table 1) discussed earlier, the Huang
et al. (2006) approach could be utilized to represent more semantics of the legacy system with higher-quality components. The Huang et al. (2006) approach, as visualized in Figure 2, is summarized in the following five steps. This study extends Huang’s methodology to reengineering system components.

The first activity is to employ the model diagrams of a legacy system such as DFD, ERD, and system interface as different inputs. There are two extended rules associated with this step and can be discussed as follows:

- **Extended rule 1:** In step 1 of Huang’s approach, the fact represented from the data stores and external entities of a DFD can be translated to classes and attributes. In some cases, the DFD itself may also imply a comprehensive system. For this reason, the DFD can consequently be translated to be a system class.

- **Extended rule 2:** In step 3 of Huang’s approach, this article performs a further analysis between the functions and external entities (or data store). When a dataflow is connected between two functions, it actually indicates the directional flow of the arrow from one function to another one. It is very similar to the control flow used in EPC. In this step, the authors regard every dataflow as a method, especially the dataflow between one function and another. Since the aforementioned data flow implies...
that the IS controls the flow of data in the system, there should be a corresponding method of a system class. An illustrative example is provided in Figure 3.

After the completion of this step, we can get the output—component specifications.

**Step 2:**

**Analyze the Business Process**

Due to tight integration with the system components in step 1, this study analyzes business process from a system perspective. This step applies and extends the “Information Flow for Business Process” (IFBP) (Huang & Yu, 2004) method to analyze the business process. Table 2 illustrates the idea of this conversion methodology for IFBP.

Again, the first activity is to use DFD of the legacy system as an input. There are three extended rules in this step; they
Figure 3. The DFD is translated into several classes

![Diagram showing the translation of DFD into classes](image-url)

are discussed below and are also shown in Table 3:

- **Extended rule 1**: The dataflow between one function and another should be translated to the event element and the associated control flow of an EPC. Further, the function can be connected with various events by using the control flow.

- **Extended rule 2**: The dataflow connected from a function to a Data Store or an external entity should be translated to be an event and a control flow. The control flow is used to connect the function and the event, and the information flow is employed to connect the function and the external entity.

- **Extended rule 3**: This rule is similar to the extended rule 2. The dataflow connected from a data store or an external entity to a function should be translated to an event and a control flow. It differs from the aforementioned extended rule 2 in the direction of the control flow and the information flow.

By applying these three rules, we can then get the output—business process (EPC).

**Step 3: Integrate System Components and Business Process**

In this step, this study integrates the system component and business process. The first activity is to use component specification and EPC as inputs. In this step, the authors will integrate processes from their original sources-DFD. The dataflow in DFD is translated to be a method of a class in step 1 and an event in step 2, so that it can be integrated later based on the same source. Afterwards, these aforementioned DFD can be shown in an EPC model, which is noted as the methods of classes. Figure
4 demonstrates the situation to integrate the system components and business processes. Finally, in this step, we can get the output—the integrated EPC model.

**Step 4: Analyze Services**

This study considers that a service is composed of system components from the perspective of the users who are employing the service. First of all, using the integrated EPC model as inputs, this research utilized the integrated EPC model in step 3 to analyze system services from the users’ perspective. Enterprises using services to apply the interaction between the external entity and other elements in the EPC model. For simplicity, this study analyzes all possible interactive situations between two entities in the EPC model, and then classifies them into seven possible situations introduced as follows.

**Table 2. The conversion methodology for IFBP**

<table>
<thead>
<tr>
<th>Steps</th>
<th>Description</th>
<th>DFD</th>
<th>EPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1</td>
<td>DFD(Function) map to EPC(Function)</td>
<td><img src="DFD" alt="Function" /></td>
<td>Function</td>
</tr>
<tr>
<td>Step 2</td>
<td>DFD(Data Flow) map to EPC(Event)</td>
<td><img src="DFD" alt="Data Flow" /></td>
<td>Event</td>
</tr>
<tr>
<td>Step 3</td>
<td>Using EPC(Control Flow) connect between EPC(Function) and EPC(Event)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Step 4</td>
<td>Using logical symbol (AND, OR, XOR) combine more than one EPC(Event) with EPC(Function)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Step 5</td>
<td>DFD(External Entity) map to EPC(Information Object)</td>
<td><img src="DFD" alt="External Entity" /></td>
<td>Information Object</td>
</tr>
<tr>
<td>Step 6</td>
<td>DFD(Data store) map to EPC(File)</td>
<td><img src="DFD" alt="Data store" /></td>
<td>File</td>
</tr>
<tr>
<td>Step 7</td>
<td>DFD(‘Data Flow’ connect with ‘Data Store’) map to EPC(Information flow)</td>
<td><img src="DFD" alt="Data store" /></td>
<td>File</td>
</tr>
</tbody>
</table>
Table 3. The extended rules for IFBP

<table>
<thead>
<tr>
<th>Present</th>
<th>DFD</th>
<th>EPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended rule 1</td>
<td><img src="#" alt="DFD Diagram" /></td>
<td><img src="#" alt="EPC Diagram" /></td>
</tr>
<tr>
<td>Function</td>
<td>Data Flow</td>
<td>Function</td>
</tr>
<tr>
<td>Function</td>
<td>Data Flow</td>
<td>Event</td>
</tr>
<tr>
<td>Data store</td>
<td></td>
<td>Function</td>
</tr>
</tbody>
</table>

Extended rule 2

- **Situation 1:** There is no entity in the process. Analyzer should use low-level DFD until it locates the interaction between the entities. The analyzer can translate the whole process into a service.
- **Situation 2:** The external entity only inputs information into one function. The analyzer can translate the whole process into a service.
- **Situation 3:** Only one function inputs information to the external entity. Again, the analyzer can translate the whole process into a service.
- **Situation 4:** The external entity inputs information into the process, and then the process outputs information to another external entity. As presented in Figure 5 and Situation 4, the whole process provides the service for A and B, so the analyzer can translate the whole process into a service.
- **Situation 5:** The external entity outputs the process, and then the process inputs information into another external entity. As presented in Figure 5 and Situation 5, A and B are two separate services that do interact with each other.
Figure 4. Integrating the system components and business processes

Function to Function

External Entity to Function

Figure 5. Services analysis
• **Situation 6**: Two external entities input different functions. As presented in Figure 5 and Situation 6, A and B are two separate services that do not interact with each other.

• **Situation 7**: Two external entities input different functions. As presented in Figure 5 and Situation 7, A and B are two separate services that do not interact with each other. Finally, we can get the outputs - service component specification and services-oriented EPC model.

After translating the related process to service components, the service components can be presented by the EPC event to repaint the EPC model, as shown in Figure 6.

**Step 5:**
**Translate to Web Services Standard**
After the EPC model has been built, the services components and the services-oriented process need to be translated into a suitable Web services standard. This study translates it to business process execution language for Web services (BPEL4WS), which is one of the Web Services Composition Languages. The user applies the translated Web services components and BPEL4WS process model to build the Web Service Architecture-based system. In this step, the research does not evaluate which component technology should be used to encapsulate the component, but translates the EPC model to BPEL4WS by using the approach of Huang and Yu (2004). Since a services component includes many associated system components, it is required to encapsulate the system component, process, and interface into a new component.

**CASE STUDY**
As shown in Figure 7, the Prototype system includes three layers (i.e., Interaction layer, Translation layer, and Repository layer). Figure 8 shows the system snapshot of the prototype.

The users input the legacy system information (DFD, ERD, user interface) in the prototype system, and then the components translator analyzes this information and stores the component into a

![Figure 6. Translation of services to process](image)
Metadata database. The users can get the information flow by the IFBP translator and build the service components as well as services-oriented process by using the WS analyzer.

To validate this prototype, this article presents a case study of the reengineering of an Accounts Receivable System (ARS). By employing experts’ help, the ERD of
the database and DFD are described in Figure 9.

According to Step 1, reengineering system component, the seven reengineering components, and eleven reengineered methods are shown on Table 4. This case study named various methods of components from the original data flow of DFD.

In Step 2, the business process of this case study is analyzed from the DFD model of the legacy system—Accounts Receivable System. The output is the EPC model as shown in Figure 10.

In Step 3, the business process and system components are integrated, and the methods of system components are employed onto the business process components. It is shown in Figure 11.

In Step 4, the services for the integrated model of this case study were analyzed, and the services-oriented architecture was consequently built. It is shown in Figure 12. The prototype implementation combined seven components and eleven methods into two services, as shown in Figure 13.

In Step 5, services components and services-oriented processes need to be translated into a suitable Web services standard. The users can then edit some information about the services components and translate

Figure 9. The DFD of accounts receivable system
<table>
<thead>
<tr>
<th>Methods</th>
<th>Sources</th>
<th>Objectives</th>
<th>Type</th>
<th>Components (class)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receivable Receipt</td>
<td>Receivable Data</td>
<td>Account Detail</td>
<td>Insert, Update, Delete</td>
<td>Account Detail</td>
</tr>
<tr>
<td>Account Data</td>
<td>Sale Month Balance</td>
<td>Receivable Data</td>
<td>Insert, Update, Delete</td>
<td>Receivable Data</td>
</tr>
<tr>
<td>Receivable Data</td>
<td>Receivable Data</td>
<td>Write-off</td>
<td>Select</td>
<td>Receivable Data</td>
</tr>
<tr>
<td>Write-off</td>
<td>Write-off</td>
<td>Receivable Data</td>
<td>Insert, Update, Delete</td>
<td>Receivable Data</td>
</tr>
<tr>
<td>Sale Data</td>
<td>Sale Data</td>
<td>Sale Month Balance</td>
<td>Select</td>
<td>Sale Data</td>
</tr>
<tr>
<td>Write-off</td>
<td>Write-off</td>
<td>Account Age List</td>
<td>Insert, Update, Delete</td>
<td>Account Age List</td>
</tr>
<tr>
<td>Export Sheet</td>
<td>Sale Month Balance</td>
<td>Make Check Sheet</td>
<td>Function to Function</td>
<td>ARS</td>
</tr>
<tr>
<td>Receivable Data</td>
<td>Receivable</td>
<td>Write-off</td>
<td>Function to Function</td>
<td>ARS</td>
</tr>
<tr>
<td>Export Sheet</td>
<td>Storehouse</td>
<td>Receivable</td>
<td>External Entity to Function</td>
<td>Storehouse</td>
</tr>
<tr>
<td>Write-off</td>
<td>Receivable Customer</td>
<td>External Entity to Function</td>
<td>Customer</td>
<td></td>
</tr>
<tr>
<td>Loan</td>
<td>Customer Receivable</td>
<td>External Entity to Function</td>
<td>Customer</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4. Components and methods of ARS**

**Figure 10. The EPC of accounts receivable system**

Copyright © 2007, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
it into business process execution language for Web services (BPEL4WS). Figure 14 demonstrates the system snapshot of this translation and the result.

In this case study, the legacy Accounts Receivable System (ARS) is a stand-alone system. In order to provide more flexibility, improve the corporation’s productivity, and enhance the capability to fit into the capricious business environment, the Web services architecture is chosen. By using this proposed migration methodology, the original seven components and eleven methods are, in fact, combined into two services. The proposed approach uses component-based technology and builds the system based on business process. For this reason, it is more suitable for business process and management. This study analyzes service components architecture based on user interaction in the business process, so it will be more closely matched with the users’ requirements.

**CONCLUSION**

The advancement of information technology is rapid. It is hard for an old architecture to keep up with the changes in the current market. Enterprises need to use the technology of information system reengineering (ISR) to preserve the value of their existing legacy systems. Currently, a Web services-based system with Service-Oriented Architecture (SOA) is widely adopted as a solution to reengineer enterprise ISs. Using this architecture, the system will be more flexible and adaptable to fit to the dramatically-changing business environment, and hence, make it...
Figure 12. Service-oriented model

Figure 13. Service-oriented model
The purpose of this research is to propose a migrating solution to translate the architecture of the legacy system to the SOA with a systematic approach. This methodology is different from traditional object-oriented approaches (see Table 5), which migrates the system to be services-oriented without applying general object-oriented (OO) or functional oriented features. In this study, the system architecture is implemented according to the prototyping development discussed above.

The contribution of this research can be summarized as follows. First of all, this study uses a systematic approach to explore the service provided by a legacy system. In specific, this research uses a systematic approach to extract business flows from the DFD diagram of a legacy system, and analyzes the service provided by the system from a service perspective. It is indeed a reasonable fit for today’s Web Service environment. Second, this study has tightly coupled the analyzed services and the rebuilt components, which will be easily extracted by the legacy system. Consequently, this proposed approach will create a more compact system, which may increase the operational efficiency. Third, the proposed methodology presented in this study can help locate the shared semantic between system components and business processes, and use a systematic method to integrate the system component and business process tightly by carefully analyzing the process. Furthermore, this method can be employed to build a methodology to translate the traditional structured analysis to a corresponding service-oriented framework. This study analyzed the DFD, ERD, and system user interface to develop business processes and system components. In addition, this study analyzed service to construct a systematic approach, which can be utilized to translate the model diagram of a legacy system into a services-oriented EPC model. The specifications of the Web Service components can be easily built in this case, which in turn can help users to
### Table 5. Comparison between the proposed methodology and other similar products and methodologies

<table>
<thead>
<tr>
<th>Feature</th>
<th>Principle</th>
<th>Support different platform</th>
<th>Flexibility</th>
<th>Reusability</th>
<th>Support business process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cape Connect <a href="http://www.capeclear.com/products/">http://www.capeclear.com/products/</a></td>
<td>Utilize the screen-scraping technology; getting the legacy system interface definitions as input and generates the appropriate WSDL files</td>
<td>Normal</td>
<td>Low</td>
<td>Low</td>
<td>No</td>
</tr>
<tr>
<td>Software AG EntireX [<a href="http://www.softwareag.com/Corpor">http://www.softwareag.com/Corpor</a> ate/products/entirex/default.asp](<a href="http://www.softwareag.com/Corpor">http://www.softwareag.com/Corpor</a> ate/products/entirex/default.asp)</td>
<td>Employ wrapping technology translate legacy system to Web services</td>
<td>High</td>
<td>Normal</td>
<td>Normal</td>
<td>No</td>
</tr>
<tr>
<td>EXTES Xuras [<a href="http://www.beacon-it.co.jp/products/pro">http://www.beacon-it.co.jp/products/pro</a> serv/cai/xuras/index.shtml](<a href="http://www.beacon-it.co.jp/products/pro">http://www.beacon-it.co.jp/products/pro</a> serv/cai/xuras/index.shtml)</td>
<td>Utilize the wrapping technology to translate legacy system into Web services</td>
<td>High</td>
<td>Normal</td>
<td>Normal</td>
<td>No</td>
</tr>
<tr>
<td>GoXML Transform Server <a href="http://www.goxml.com/features.php">http://www.goxml.com/features.php</a></td>
<td>Integrate business processes by mapping the complex data formats such as EDI, SWIFT, COBOL, CSV, flat text, XML, XBRL</td>
<td>Normal</td>
<td>Low</td>
<td>Low</td>
<td>No</td>
</tr>
<tr>
<td>Xbridge Host Data Connect <a href="http://www.xbridgesystems.com/">http://www.xbridgesystems.com/</a></td>
<td>Enhance the Web-services’ capabilities of the OS/390 mainframe data access product</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>No</td>
</tr>
<tr>
<td>IONA Orbix <a href="http://www.iona.com/products/orbix/">http://www.iona.com/products/orbix/</a></td>
<td>Translate program to COBRA object and then convert to Web services</td>
<td>High</td>
<td>Normal</td>
<td>Normal</td>
<td>No</td>
</tr>
<tr>
<td>BALES (Heuvel, Hillegersberg, &amp; Papa-zoglou, 2002)</td>
<td>Use forward engineering technique to define business domain and use reverse engineering technique to understand the function of a legacy system; use of the Component Definition Language (CDL) to specify both legacy and business objects in order to facilitate a search for matching objects and parameters of Web-services with legacy objects</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Yes (only focus on functions reference)</td>
</tr>
</tbody>
</table>

*continued on following page*
construct Web Service components more efficiently. In addition, the IS reengineering technique can be utilized to build the prototype based on the proposed Web Services framework. To this end, this study built a prototype to validate this proposed methodology. This prototype can guide users to reengineer the legacy system into a Web Service framework. As a result, this research can be a valuable reference for other future studies.

This study adopted a Rule-Based translation methodology, which can be used to improve not only the quality of the design/development, but also to enhance the correctness of the original data. Some ambiguous or vague wordings presented in the original DFD can definitely influence the quality of a translated process. It is obvious that every phase should be properly revised, so that the output would better fit the actual practice performed in the businesses and/or industries.

In practice, some organizations can either have incomplete DFDs or even have no DFDs to perform the system/service design. This study cannot be applied to such organizations. Fortunately, techniques such as IS reverse engineering have gained considerable progress in translating source code to information flow such as DFD (Benedusi, Cimitile, & De Carlini, 1989; Lakhotia, 1995) or translating into ERD diagram from database (Computer Associates, Inc., 2006; SyBase, Inc., 2007).

This study placed focus on the conversion methodology, and paid limited attention to the techniques to develop system components, such as EJB, CORBA, or COM+. Practically, different composition techniques may have different ways to implement the Web Services. To this end, this study lacks any discussion of the different implementation alternatives to design/develop the Web Services.

The proposed methodology is tailored to IS’ assistance to various business operations. It works better in situations such as having complicated processes and external transactions. As a result, it does not have sufficient capability in dealing with a simple process or some applications without process. Furthermore, other system programming and firmware that require specific hardware cannot be easily applied with this study.

Future work needs to be done towards a full automatic reengineering process to eliminate the unexpected human factors and/or possible errors. Further study should also be investigated to determine the right component implementation technique. In addition, it is also required to enhance the participation of domain knowledge experts. In the future, additional benefit may include the development of new applications by
composing these software components found in the reengineering process in order to accelerate the development time.

ACKNOWLEDGMENT
The National Science Council, Taiwan, under Grant No. NSC95-2524-S-194-004-, has supported the work presented in this article. We greatly appreciate its financial support and encouragement.

REFERENCES


Engineering Information Systems in the Internet Context (pp. 81-103).


Shing-Han Li is an assistant professor of the Department of Information Management at Tatung University in Taiwan. He received his MS in computer science and information engineering from Tatung University, Taiwan in 1996, and PhD in information management from National Chung-Cheng University, Taiwan, 2006. His current research interests include web services, ERP system, Information quality, and information auditing and assurance.

Dr. Shi-Ming Huang received his PhD degree at the School of Computing and Information Systems, University of Sunderland, UK. He is currently a head of Accounting and Information Technology Department and a director for the Research Center of e-Manufacturing and e-Commerce at National Chung Cheng University, Taiwan. He is the 2006 president for International Chinese Information Systems Association (ICISA). He has published five books, three business software and over 50 articles in refereed information system journals, such as Information and Management, Decision Support Systems, Journal of Computer Information Systems, European Journal of Operational Research, Journal of Database Management, ACM SIGMOD, etc. He has received over 10 achievement awards in information system area. He has served as editorial board members in several international journals and has acted as a consultant for a variety of Taiwan government departments, software companies and commercial companies.

David C. Yen is a professor of MIS and chair of the Department of Decision Sciences and Man-
agement Information Systems at Miami University. He received a PhD in MIS and Master’s of sciences in computer science from the University of Nebraska. Professor Yen is active in research, he has published three books and many articles which have appeared in Communications of the ACM, Decision Support Systems, Information & Management, International Journal of Information Management, Information Sciences, Journal of Computer Information Systems, Interfaces, Telematics and Informatics, Computer Standards and Interfaces, Information Society, Omega, International Journal of Organizational Computing and Electronic Commerce, Communications of AIS, and Internet Research among others. He was also one of the co-recipients for a number of grants such as Cleveland Foundation (1987-1988), GE Foundation (1989), and Microsoft Foundation (1996-1997).

Cheng-Chun Chang is a master at the Department of Information Management in National ChungCheng University, Taiwan. He was an system engineer at the Systex INC. in Taiwan. His research includes web services, information system reengineering, object oriented techniques, enterprise application integration and software component techniques.