A runtime-adaptable service bus design for telecom operations support systems

The complexity that telecommunications companies are faced with in their business processes and their information technology (IT) systems is especially apparent in their billing systems. These systems are required not only to handle large volumes of data and frequent changes in business rules, but also to ensure that the billing be done accurately and on time. This paper describes a solution that was developed to address this problem. It consists of an operations support system that is compliant with NGOSS (Next Generation Operations System and Software) and it implements a service-oriented architecture (SOA) that relies on an enhanced enterprise service bus (ESB). This enhanced ESB, referred to here as an adaptable service bus (ASB), makes it possible to carry out changes to business rules at runtime, thus avoiding costly shutdowns to the billing application. An implementation of this system has been operational in ChungHwa Telecom Company, Taiwan, since January 2008 and provides complete support to its billing application. As a result, the billing process cycle time has been reduced from 10–16 days to 3–4 days, which cleared the way for further growth of the business.

INTRODUCTION

The information technology (IT) systems used by communications service providers (CSPs) are known as operations support systems (OSSs). Sometimes it is desirable to separate business functions from network functions, in which case the OSS is associated with network support and the IT systems providing business support are identified as business support systems (BSSs). The complexity of these systems, not insignificant to start with, is increased by the frequent need to update them for changes related to services, customers, and business rules.

The TeleManagement Forum (TMF) is a consortium of CSPs whose role is to introduce standards to the communications industry. It initiated the NGOSS (Next Generation Operations System and Software) program in order to help CSPs develop new OSSs and BSSs to better support their businesses.1
NGOSS, which consists of a set of principles and technical deliverables, identifies areas that need to be addressed—such as billing collection management and service-specific rating—but does not specify how they should be implemented. The challenge of designing state-of-the-art OSSs and BSSs is compounded by the fact that although NGOSS has been the topic of frequent discussions within the industry, it has not attracted much attention in the academic community, resulting in a scarcity of research papers in the technical literature.

The work presented here is based on a service-oriented architecture (SOA) that relies on an enhanced enterprise service bus (ESB). This enhanced ESB, referred to here as an adaptable service bus (ASB), constitutes a significant improvement over a state-of-the-art ESB because it supports changes to business rules at runtime, thus avoiding costly shutdowns of the supported applications.

Although vendors such as IBM, BEA, and Oracle provide ESB products for integrating service-based enterprise applications, these solutions sometime incur high implementation costs and present complex migration and management problems. For changing requirements, commercially available ESB products only allow such changes to be implemented at design time, which means that the ESB server has to be shut down for recompiling the application and rebooting the system. This problem has been studied and a number of solutions have been proposed to support a service bus design in which changes to the application can be carried out at runtime.

Penta et al. presented WS Binder, a framework that enables dynamic binding of service composition. It supports three binding types: pre-execution binding, runtime binding, and runtime re-binding. The two binding types at runtime enable the dynamic binding of alternative services when the primary services are unavailable. The decision policy used in this approach is based on quality of service (QoS) requirements.

Chafle et al. outlined a multistage approach to adaptive Web Service Composition and Execution (A-WSCE) that cleanly separates the functional and nonfunctional requirements of a new service, and enables different environmental changes to occur at different stages of composition and execution. In this mechanism, the service consumer selects a set of service providers with functional requirements and then selects the best service provider from the set for invocation based on feedback mechanisms and suitable ranking functions of the nonfunctional requirements.

Chang et al. proposed a dynamic composition handler (DCH) design for an enterprise service bus. This design allows services to be discovered, selected, composed, and adapted at runtime. Bai et al. propose DRESR, a framework for dynamic reconfigurable ESB service routing, which is based on dynamic routing path construction and message routing. It defines the mechanisms for routing path abstraction, instantiation and reconfiguration. The framework supports the specification of service selection preferences and facilitates service selection based on runtime testing results.

This paper proposes the adaptable service bus (ASB), a service bus that also enables dynamic composition of services. Whereas the ASB shares many of the characteristics of a conventional ESB, it places the service endpoint, service operation name, and parameter values related to the service operation in external storage. In order to modify the behavior of the application at runtime, developers can adjust the parameter values maintained in the external storage.

Table 1 documents the differences between four service bus designs: ASB, DCH, DRESR, and ESB. Four criteria are used for comparing the four designs. The first documents whether the dynamic composition of services is supported. The second criterion (adaptation support) indicates how readily a system can be adapted to a change in requirements. There are three possible values: no support, manual support, and automated support. A fully automated adaptation is not feasible with current SOA standards and toolsets. Manual support implies that changes are made by a developer through the user interface.

The decision strategy criterion refers to the method used to select a target service: decision rules or through ranking. The decision rule approach does not rely on historical performance data when choosing a service. The ranking methodology supports the selection of a target service based on its
past performance. Of the two, the later is more conducive to increased efficiency, as it allows for the selection of the “best” option. The fourth criterion identifies the deployment level, which in this case may take one of two values: “the design was tested in the lab only” (Lab-tested) or “the design was implemented and deployed in a commercial environment” (deployed). As shown in Table 1, the ASB presented in this paper improves upon existing service bus designs by combining the best values for the criteria used.

The work presented here was a joint effort, with the participation of academic researchers from the National Taipei University of Technology and a team of engineers from ChungHwa Telecom Co., Ltd. ChungHwa Telecom is the largest telecommunication company in Taiwan and the 14th largest in the world. Its annual revenue for 2007 was in excess of 197.3 billion New Taiwan dollars (NT$), or 6.57 billion U.S. dollars (US$). As part of its development program, the company sought to upgrade its existent billing system to one that was both NGOSS-compliant and SOA-based. To achieve this, a service-oriented integration platform was developed that includes both a workflow engine and the enhanced service bus described here.

The ASB-based implementation was successfully completed in November 2007, has been operational at ChungHwa Telecom since January 2008, and provides complete support to its billing application. The system supports 18 million customer accounts valued at approximately 1.2 billion NT$ (about 40 million US$). The research presented in this paper adheres strictly to the NGOSS development life cycle and is intended to provide a practical example of implementing an SOA in the NGOSS environment.

The rest of this paper is organized as follows. The next section describes the ASB and its five main components: the user interface, service registry, service router, data transformer, and service invocator. The section that follows describes the ChungHwa Telecom billing system; a system overview is followed by a description of how the NGOSS development methodology was applied and then a summary of system performance as reflected in the performance data. The last section contains a brief summary of the work presented here.

### ADAPTABLE SERVICE BUS

ESBs, which serve as intermediaries between processes and the services they consume, have been primarily limited by the fact that they cannot be reconfigured at runtime. This matters because both service consumers and service providers are relying upon an operational ESB. ESB shutdowns for maintenance affect the flow in both directions and thus impede the operation of the entire system. In this section we describe the design of the ASB, which is crucial to its support of runtime changes to business rules.

Figure 1 shows the overall architecture of the ASB and its main components in the ChungHwa Telecom billing system: the user interface, service bus registry, service router, data transformer, and service invocator. It also demonstrates the role of the ASB in the billing process.

To the left in Figure 1 is the operations support system that details requirements for task scheduling, processes for bill calculation, monitoring of results, and reporting of the results. This OSS is essentially representative of the features of NGOSS that pertain to this research.

Next is the billing process, which dictates the method through which bills will be calculated. To the right of the ASB are the billing services. These requirements are the actual means of affecting the

<table>
<thead>
<tr>
<th></th>
<th>ASB</th>
<th>DCH</th>
<th>DRESR</th>
<th>ESB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic composition</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Adaptation support</td>
<td>Manual</td>
<td>Manual</td>
<td>Manual</td>
<td>No support</td>
</tr>
<tr>
<td>Decision strategy</td>
<td>Ranking</td>
<td>Decision rule</td>
<td>Ranking</td>
<td>Static binding</td>
</tr>
<tr>
<td>Deployment level</td>
<td>Deployed</td>
<td>Lab-tested</td>
<td>Lab-tested</td>
<td>Deployed</td>
</tr>
</tbody>
</table>

Table 1 Comparison of four service bus designs
billing process by calculating account debts and credits. The detailed explanations of each component in the adaptable service bus are described below.

As noted, the ASB is composed of five components. The first, a component user interface, enables developers to interact with the system so that they can set or modify rules and parameters. The second, a component service bus registry, performs two key functions: it registers the available services and it records performance measures over time. The third, a component service router, is a rule-based routing engine that determines which services will be invoked.

One of the major challenges faced by an SOA-based OSS is the need to make use of legacy services (e.g., the billing services indicated in Figure 1), which are not NGOSS-compliant. Ensuring the support for legacy services is the responsibility of the fourth component, the data transformer. The transformer employs Extensible Stylesheet Language (XSL) to translate the older Billing Service Data (BSD), used by legacy services, into the NGOSS-mandated Shared Information Data (SID) format, and vice versa. The last component is the service invocator, which invokes a service on behalf of the process that originated the request.

Service bus registry
The service bus registry component (or simply the registry) stores information on each available service, including the service name, address, operations, input/output data types, and QoS data. Figure 2 illustrates the interaction between the service bus registry and service router within the broader ASB architecture.

For each available service, the registry maintains a Web Services Description Language (WSDL) record and a QoS record. Using this format ensures that the service router is able to access these data. Developers can enter, modify, and remove registry information via the user interface.

Service router
In response to a service request, the service router component (or simply the router) is responsible for selecting (at runtime) the service to be used. It consists of a rule-based routing engine and a rule repository. Developers can modify routing rules at runtime, without recompiling and reinstalling the routing module. Developers create or modify routing rules, which are kept in the rule repository. This can be done either when the system is not operational (in maintenance mode) or at runtime. The routing rules implement a routing policy. This structure is illustrated in Figure 2.

The routing policy allows the router to select not only an appropriate service, but also the most appropriate service available—when confronted with more than one viable option. The selection criteria, found in the routing policy, can be defined...
for different situations. For example, the following two quality attributes are used to help the router select the most appropriate available service:

**Definition 1**

The **average execution time** of a service ($p$) where $n$ is the number of times a service has been invoked:

$$p = \frac{\sum_{i=1}^{n} \text{execution time}}{n}, \quad \{n \geq 1\}$$

**Definition 2**

The **expected availability** of a service ($\alpha$) where the mean values are obtained by dividing the cumulative (over time) values of each variable, time_to_failure and time_to_repair, by the total number of failures.

$$\alpha = \frac{\text{mean time to failure}}{\text{mean time to failure} + \text{mean time to repair}}$$

These quality attributes are measured by an external monitoring component. For each service invocation the corresponding execution time is fed to the service registry, so that the service QoS record can be updated.

Developers can also use the routing policy (as expressed by business rules) to prioritize certain quality attributes. This enables the dynamic routing functionality of the ASB, which can in turn provide a manageable interface for reconfiguring routing at runtime.

The router is responsible for determining the path on which a request message travels. When a request is received from a service consumer, the router enforces the routing policy. The router then compares the request to the information available within the service bus registry, using the routing policy as a reference.

**Data transformer**

As noted, one of the major difficulties faced by telecommunications companies striving to implement NGOSS is that NGOSS is not always compatible with the existing systems. Specifically, if a service is invoked using the NGOSS-compliant SID format embedded in WSDL, then a non-NGOSS-compliant service will be unable to respond adequately. This is because the noncompliant services of the legacy systems rely upon the BSD data structure. To address this problem, the ASB incorporates a data transformer component. The transformer acts as a translator between the service consumer and the service provider. Translation is
accomplished by means of Extensible Stylesheet Language Transformations (XSLT) technology, which provides a program interface for developers to implement mapping between two different data models.

**Figure 3** illustrates the data transformation and service invocation process. The data transformation rules are kept as an XSL file for each service. The transformation engine performs the translation using the transformation policy as specified by the transformation rules.

The data transformer receives from the service router a service request that specifies the service that should be employed on behalf of the service consumer. The transformer responds to the request by looking up the data transformation rules as found in the XSL file for the requested service. The identified transformation rules are used to perform the necessary translation, and an appropriate message is created and sent to service invocator.

**Listings 1 and 2** show the same data sample in two different formats: an SID format for an NGOSS-compliant service provider (Listing 1), and a BSD format (Listing 2) used by a noncompliant service. **Listing 3** shows the transformation process used to translate, or map, one format to the other.

In Listings 1 and 2, the identification information, parameter values, and other information are represented in SID and BSD format. It should be pointed out that not all parameters are the same, because the object in Listing 2 represents a message directed to the requested service, while the object in Listing 1 represents the information in the service request. The additional information seen in Listing 1 is router-specific information that is “consumed” by the router as the request makes its way through the ASB.

Listing 3 shows the XSL transformation used to translate the object in Listing 1 into the object in Listing 2.

**Service invocator**

The component service invocator, which is designed to invoke a service on behalf of a service consumer,
is constructed using the Web Service Invocation Framework (WSIF) application programming interface (API)\(^\text{13}\).

To ensure that the ASB operates properly, a generic service invocation framework is needed. As a result, the service invocator includes a generic service proxy that runs on a Simple Object Access Protocol (SOAP) engine. The proxy acts as an agent, binding the ASB to the services to be invoked. This is illustrated in Figure 3.

The service invocator can be viewed as a dynamic service binding module. In order to communicate with service providers, service entry points, operation names, and parameter values are required. The service invocator receives all this information via the data transformer (from the service router, which extracts the data from the service registry). The service proxy packages the information as a specific request format for invoking the target service provider through the SOAP engine.

The proxy server is constructed in several steps as follows:

1. Establish a WSIF service object via the WSIF service factory component of the service proxy.
2. Define the mapping type of the service.
3. Establish a WSIF port object via the WSIF service with the service port type.
4. Create a WSIF operation via the WSIF port.
5. Set the service context using a WSIF operation.
6. Create input, output, and error message objects via the WSIF operations.
7. Set the input message object.
8. Invoke remote services.
9. Retrieve the results.
10. Forward the results to the service consumer (billing process).

This programming model demonstrates how the WSIF API is used to build a dynamic service proxy. This module can dynamically invoke remote services through WSDL.

**THE CHUNGHWA TELECOM BILLING SYSTEM**

The ChungHwa Telecom billing system prepares and issues bills for customers (account holders). The company offers a wide range of services and service packages to customers and as a result, the billing process can be quite complex. This billing process might begin with a calculation of the basic charges for the type of service, such as phone line or asymmetric digital subscriber line (ADSL). Additional charges may be levied based on use. Phone customers may be charged based on minutes of use, the location of the called number, and so on. Some customers are subject to preferential pricing. For example, large companies may receive volume discounts, whereas cell phone customers may receive discounts based on the length of contract. Special offers, outstanding debts, credits from previous bills, or charges involving another telecom company must also be accounted for.

The old billing system relied on a manually prepared list of tasks, each task associated with a service to be invoked and an invocation time. An example of a task would be calculating cell phone minutes when the connection was to a cell account from another telecom company. Because of the tight coupling between tasks and services, a change in a business rule or system requirements required that the entire list be constructed anew. Because the execution time required to complete a task varied from task to task, it was also necessary to calculate the execution time for each task. The process was inefficient and difficult to maintain. As the number of accounts and the range of telecom services offered by the company increased, the workload increased and reached crisis levels.

Although SOA was seen as a desirable architecture, problems associated with the conventional ESB limited the benefits of SOA. Foremost among these problems was the inability to accommodate changes to business rules at runtime. The frequency of

---

**Listing 2 Message in Billing Service Data format**

```plaintext
<BillingServiceData>
  <piid>piid:_199038_393907_838748</piid>
  <stepid>1</stepid>
  <billYm>200802</billYm>
  <billCyc>1</billCyc>
  <nSeqNoModDivisor>5</nSeqNoModDivisor>
  <nSeqNoModRemainder>1</nSeqNoModRemainder>
  <sFileName>LBx_200802C1_R3000000.CDS</sFileName>
</BillingServiceData>
```
changes to the billing process is so high at ChungHwa Telecom that shutting down the application for implementing these changes was deemed "not acceptable."

Figure 4 illustrates the architecture of the ChungHwa Telecom billing system. The OSS, implemented on an IBM WebSphere* Portal Server, provides support for task scheduling, process management, monitoring of results, and reporting.

The billing process, shown on the left in Figure 4, operates on an IBM WebSphere Process Server. It consists of 46 different steps, each of which entails a specific task. As mentioned, these tasks include things such as retrieving customer data and making calculations. For manageability, the billing for the 18 million customer accounts is split into six separate cycles. Task scheduling determines what will be calculated in each cycle. The second group of requirements, those for the process of bill calculation, dictate the way in which bills will be calculated. The third, monitoring of results, includes fault discovery and manages issues that threaten data consistency, such as rounding uneven dollar values. The final set of requirements determines how the actual billing will occur once account debts and credits are calculated.

The third component shown in Figure 4 is the ASB. The ASB mediates between the billing process and the services it requires to accomplish its tasks. The ASB is based on the IBM Message Queue Server.

The final stage of Figure 4 shows the services required by the process for the completion of tasks. These services, which are provided by legacy (previous generation) systems, have been developed independently on a variety of platforms and use several programming environments, such as Procedural Language/Structured Query Language (PL/SQL), C++, and shell scripts. The ASB developers designed adapters for each task in the billing process and constructed them on an IBM WebSphere Application Server.

The NGOSS design methodology
For developing NGOSS-compliant applications, the NGOSS framework calls for a four-step continuously iterating life cycle. The four steps correspond to four different views: a business view, a system

<table>
<thead>
<tr>
<th>Listing 3 XSL transformation for mapping data objects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">  
  <xsl:template match="/">  
    <BillingServiceData>  
      <piid><xsl:value-of select="piid" /></piid>  
      <stepid><xsl:value-of select="stepid" /></stepid>  
      <billYm><xsl:value-of select="billYm" /></billYm>  
      <billCyc><xsl:value-of select="billCyc" /></billCyc>  
      <nSeqNoModDivisor><xsl:value-of select="nSeqNoModDivisor" /></nSeqNoModDivisor>  
      <nSeqNoModRemainder><xsl:value-of select="nSeqNoModRemainder" /></nSeqNoModRemainder>  
      <sFileName><xsl:value-of select="sFileName" /></sFileName>  
    </BillingServiceData>  
  </xsl:template>  
</xsl:stylesheet>  

analysis view, an implementation view, and a testing and deployment view. The NGOSS development life cycle is shown in Figure 5.

The business view is concerned with the analysis of business requirements. This step is conducted by a business analyst in response to a business decision. A business process framework eTOM (enhanced Telecom Operation)\(^{15}\) is defined in NGOSS and is modeled using IBM WebSphere Business Modeler Advanced V6.0.2. If the simulation is successful then the new business model is incorporated into the OSS.

The second phase, which entails the transfer of the business model to a BPEL process and SID,\(^{16}\) is performed using WebSphere Integration Developer V6.0.2. The third phase involves the implementation of the application process, the ASB, and the service adapters (adapters used by providers of services). This is accomplished using Rational\(^*\) Software Architect V6.1.

The fourth phase involves the testing and deployment of the components constructed during the third phase. Testing is accomplished using Rational TestManager V6, whereas deployment relies upon WebSphere Process Server V6.0.2, WebSphere Message Queue V6 and WebSphere Application Server V6.0.2.

Following the completion of the four stages, the Web-based user interface is implemented. The user
interface runs on a WebSphere Portal Server V5.1. Details of the tools used to design the different stages of the ChungHwa Telecom Billing System, along with their respective servers are shown in Figure 5.

System performance
The scale of the billing process at ChungHwa is immense. For billing, the 18 million accounts were partitioned into six groups, which were processed in six different cycles. Yet even this division into smaller jobs did not sufficiently solve the problem.

Because the previous system followed a fixed process, changes or errors in the process required a shutdown of the system so that revisions could be made. Shutdowns caused a loss of intermediate results in the process and also necessitated the rollback of services to their ready state, necessitating a re-execution of the entire process. As a result, each of the six cycles required 10 to 16 days for completion. Since billing occurs on a monthly basis, this time period required the company to run multiple systems in parallel—just to ensure that all bills were calculated each month. As the company grew in size, the situation became increasingly unmanageable.

In practical terms, the primary objective was to reduce the time needed for each cycle. This is illustrated in the following equation, which provides the time $T$ required for the completion of a billing process execution:

$$T = \left( \sum_{i=0}^{m} t_i \right) + t_s$$

where $m$ = the number of times a service has been changed, $t_i$ = the execution time for incomplete cycles (days), and $t_s$ = the execution time for successful cycles (days).

The billing cycle failure corresponding to parameter $t_i$ may occur for a number of reasons. Sometimes it is the result of an error, but more frequently it is the result of a change in services. In the old system this value was extremely high, as changes are frequent, and the non-ASB system required a complete shutdown and recalculation for each change made. Indeed, for this system, the value of $T$ was between 10 and 16 days for each cycle.

By reducing the value of $t_i$, the time required for the completion of each cycle is proportionally reduced. Because the leading cause of cycle failure was the need to shut down and restart the application, the introduction of the ASB—which can be altered at runtime—means a dramatic decrease in $T$. In practice, the reduction of $t_i$ meant an average cycle of 3 to 4 days.

CONCLUSION
Information systems for telecommunications companies are complex. For ChungHwa Telecom, this complexity represented an obstacle to growth. Its non-NGOSS-compliant billing system of the past made use of a scheduling table that tightly bound process tasks to services. The billing process had to be shut down whenever errors occurred or changes were required.

For ChungHwa Telecom such events are a frequent occurrence, which meant that the billing process took between 10 and 16 days to complete. The long processing time is due primarily to the loss of intermediate results when the billing process is interrupted and the system is shut down. Because the billing required six separate processing cycles
each month, multiple servers operated in parallel, a costly and complex solution.

In order to reduce this billing burden so that the company could continue to expand, there was a growing recognition that a more efficient process was needed. The solution was identified as a move toward an SOA based on an enhanced ESB. This resulted in the decoupling of the billing process from the services used; the ASB became an intermediary whose role was to manage the interactions between them.

The current billing system shows a dramatically shortened cycle of 3–4 days. The introduction of the ASB-based infrastructure has also contributed to the manageability and predictability of the billing process. The company was able to pursue further expansion of its business by removing a key bottleneck in one of its major processes. A further benefit is its NGOSS compliance, which brings ChungHwa in line with the best practices for the telecommunications industry.

Because the billing process is just one of the target areas for NGOSS compliance, future work will be directed toward the re-engineering of other business processes, such as operations support and readiness, fulfillment, and assurance. As technical publications in the area of NGOSS-compliant OSSs and BSSs are scarce, it is hoped that this paper will serve as a reference point for further research on this subject.

ACKNOWLEDGMENTS
We thank the employees of ChungHwa Telecom Company who contributed to this project. We acknowledge with gratitude the support of the National Science Council and Ministry of Economic Affairs, Taiwan, R.O.C., under Grants NSC 96–2622-E-027–024-CC3 and 95-EC-17-A-02-S1–029.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

CITED REFERENCES

Accepted for publication March 25, 2008.
Published online July 30, 2008.

Ing-Yi (Eric) Chen
Department of Computer Science and Information Engineering, National Taipei University of Technology, No. 1, Sec. 3, Chung-
Dr. Chen received a B.Sc. degree in physics from National Central University, Taiwan, in 1984, and M.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Arizona in 1989 and 1992, respectively. He is currently an associate professor in the Computer Science and Information Engineering Department at National Taipei University of Technology (NTUT) in Taiwan. In 2000 and 2001, prior to joining NTUT, he served as chief technology officer for Chinatimes, with responsibility for the corporate strategic technology planning and for handling university relations. From 1992 to 2000 he was on the faculty of the Electronic Engineering Department at Chung Yuan Christian University (CYCU) in Taiwan. His research interests include various topics in SOA and solution frameworks for building service-oriented applications. A member of the Phi Tau Phi scholastic honor society, he was a recipient of the IEEE/ACM Design Automation Scholarship Award in 1991, and the Distinguished Teaching Award at CYCU in 1996 and at NTUT in 2008.

Guo-Kai (Ken) Ni
Department of Computer Science and Information Engineering, National Taipei University of Technology, No. 1, Sec. 3, Chung-Hsiao E. Rd., Taipei, Taiwan (t5599007@ntut.edu.tw). Mr. Ni received his B.S. in computer science and information engineering from National Taipei University of Technology (NTUT) in 2005. Currently, he is working toward his Ph.D. degree at NTUT. His areas of interest include software engineering and service-oriented architecture.

Chau-Young (Ivan) Lin
Telecommunication Laboratories, ChungHwa Telecom, 4F, No. 11, Lane 74, Sec. 4, Hsin-Yi Rd., Taipei, Taiwan (ivanlin@cht.com.tw). Dr. Lin received B.S. and M.S. degrees in computer engineering from Feng-Chia University, Taiwan, in 1985 and 1987, respectively, and a Ph.D. degree from National Taiwan University of Science and Technology in 1999. Since then he has been with ChungHwa Telecom at the Telecommunication Labs, where he led the effort to develop an NGOSS-compliant billing system. His areas of expertise include ontology modeling, SOA, NGOSS, billing systems, and ITIL. He is currently the manager of the NGOSS-BSS project, which involves 60 engineers and whose focus is the re-engineering and integration of silo-based billing systems.