Toward a Service-Oriented Development Through a Case Study

Jia Zhang, *Member, IEEE*, Carl K. Chang, *Fellow, IEEE*,
Liang-Jie Zhang, *Senior Member, IEEE*, and Patrick C. K. Hung

***Abstract*—The rapidly emerging technology of Web services paves a new cost-effective way of engineering software to quickly develop and deploy Web applications by dynamically integrating other independently developed Web-service components to conduct new business transactions. This paper reports our efforts on designing and developing a Web service of pass-through authentication (PTA) for 12 online electronic-payment Web applications. In accordance with how a PTA service is developed and integrated with a corresponding back-end e-payment system, our strategies can be categorized in three stages: end-to-end integration stage, Web-services-enabled stage, and Web-services-oriented stage. Derived from real-world industrial experience, this three-stage pathway can be applied to a broad range of Web-application development projects to guide smooth transformation from a specific application-oriented design and development model toward a reusable Web-services-oriented model. Furthermore, this paper contributes to an engineering process that leads to practical Web-services-oriented software development. New research issues revealed by this project are also reported.**

***Index Terms*—Architecture, case study, software-application development, Web-services deployment.**

I. Introduction

THE DEVELOPMENT of Web applications can be quite complex, costly, and time-consuming [10], due to their unique features such as complicated business logic, high interactivity, requests of quick delivery, high-quality requirements, and ambiguous requirements at the start. In order to increase the productivity and quality of Web-application development, Web engineering has emerged as a peculiar research area that contributes to manage the diversity and complexity of the Web-application development process [13]. Its essential goal is to provide systematic, disciplined, and quantifiable approaches to assist the design, development, and maintenance of Web applications [8].

Manuscript received January 30, 2006; revised August 16, 2006. This paper was recommended by Associate Editor H. R. Rao.

J. Zhang is with the Department of Computer Science, Northern Illinois University, DeKalb, IL 60115 USA (e-mail: jiazhang@cs.niu.edu).

C. K. Chang is with the Department of Computer Science, Iowa State University, Ames, IA 50011 USA (e-mail: chang@cs.iastate.edu).

L.-J. Zhang is with IBM T. J. Watson Research, Yorktown Heights, NY 10598 USA (e-mail: zhanglj@us.ibm.com).

P. C. K. Hung is with the Faculty of Business and Information Technology, University of Ontario Institute of Technology, Oshawa, ON L1H 7K4, Canada (e-mail: patrick.hung@uoit.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org

Digital Object Identifier 10.1109/TSMCA.2007.904820

In recent years, the rapidly emerging concept of Web services has been gathering significant academic and industrial momentum, as it has been changing the Internet from a repository of data into a repository of services. A Web service itself is a programmable Web application that is universally accessible through standard Internet protocols [9]. It is normally self-contained with well-defined interfaces with no compulsory deployment. This paradigm of Web services paves a new cost-effective way of engineering Web applications by integrating other independently developed Web-service components. As a result, numerous software organizations are going through a transformation of their engineering models, aiming at achieving faster and cheaper Web development.

In a Web-services-based Web-application development, two essential issues need to be tackled: 1) how to design and develop a new application utilizing available Web services as components and 2) how to build a reusable application that exposes part or all of its functionalities as Web services. This paper describes the process of creating an architectural model as a solution to these two issues at a software consulting firm. As a case study, we report on a project aiming at constructing a Web service of pass-through authentication (PTA) to support online-payment (electronic payment, e-payment) applications. It is the first step toward a comprehensive Web service of online e-payment. Currently, such a PTA service acts as a Web service to support our multiple e-payment deployments with three major functionalities: 1) providing a PTA service; 2) allowing a client administrator to register and update authentication criteria; and 3) functioning as a Web-service front-end to our e-payment back-end servers. This paper examines how our Web-services-based service-oriented architecture (SOA) was formed through this project and how it has benefited the corresponding Web-application development.

The remainder of this paper is organized as follows. We will first analyze the project domain and project requirements, as well as research questions. After closely examining the case study, we will present how an architectural model is extracted to facilitate Web-services-oriented Web-application development and its impact. Then, we will analyze the case study and discuss related work. Finally, we will draw conclusions.

II. Case-Study Context

We will first describe the context in which the case study was performed. Our e-payment system is an online-payment-processing application. Contrasted with PayPal [50] as a known Web-services-based payment system serving generic purposes,

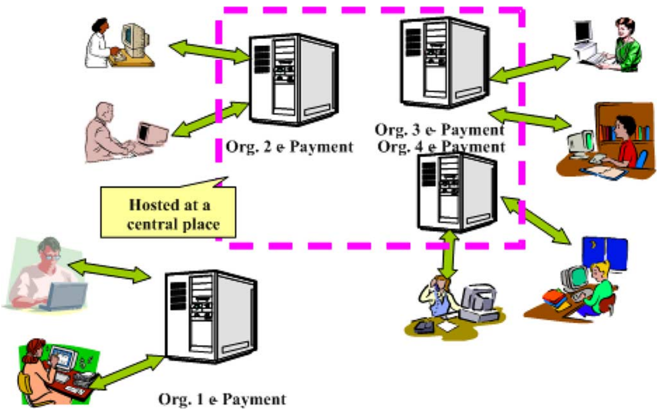


Fig. 1. Original e-payment system deployment.

our e-payment system focuses on providing a customizable and comprehensive payment facility, which can be plugged into existing Web applications as an integral part. For example, our e-payment system supports more comprehensive payment methods. The payment methods that PayPal allows are as follows: PayPal balance, PayPal buyer credit line, PayPal plus credit card, debit card, or confirmed bank account. Our e-payment system generally allows payment credit card, debit card, personal and business check, money order, and so on. Another example is that our e-payment system for a particular organization may possess very specific features, e.g., supporting a specific-transaction report format and schedule.

Our e-payment system typically contains two modes: synchronous or asynchronous, meaning that it can perform both real-time and batched payment transactions. The application handles credit-card payment, e-check payment, account-receivable updates, payment processing and reconciliation, bill presentment and payment-history presentation, and so on. As a project conducted at a Web-software-consulting firm, the e-payment system was originally designed as a stand-alone Web application that can be customized and deployed at different organizations.

Fig. 1 illustrates the overall design and distribution of the e-payment application over organizations. Based on different deployment expenses, two options are provided to organizations. The first one is called an enterprise style, meaning that an e-payment system is deployed at a server machine hosted at an organization site, as shown in Fig. 1 for Organization #1. The organization needs to maintain the server, and the deployment fee is paid up once for all at the deployment time. The second option is a so-called Application Service Provider (ASP) style [37], [46]. An e-payment system for an organization is hosted at a central server place, which is normally a dedicated third-party Web-hosting place. In Fig. 1, Organizations #2, #3, and #4 adopt this latter style; the e-payment systems serving different organizations reside on different physical server machines. Note that these e-payment systems may reside on the same physical machine, each on a different application-server instance, such as a Macromedia JRun server [17] instance. Different e-payment servers are isolated applications even though they reside at the same machine. Through this deployment option, maintenance and host fees are paid on a monthly basis. It is known that the ASP model is shifting businesses from buying software products to renting services [37]. In our experience, most organizations did adopt the second option. Although, in the long run, organizations spend more money, they benefit from no resource consumption on maintenance and easier future upgrade of the e-payment systems. In either option, users of each organization are required to access the login page of the corresponding deployed e-payment system before utilizing the application. Normally, an organization embeds a link in its own Web site to point to its corresponding e-payment server site.

As we continued to deploy e-payment applications for more organizations, this original design reveals several integration issues. First, a user uses different sets of login IDs and passwords to access the e-payment application and the original Web site of the organization, respectively. We could permit the users to access the e-payment application if we prestore all user information of the organizations. However, it means that the organizations have to transfer batch files with sensitive information, such as user names and passwords. The process apparently exposes a potential security problem. This issue becomes more severe if an organization adopts the ASP model, since the sensitive information needs to be conveyed through the public Internet to the central-server place. Second, this design forces users to go through the Web pages of the e-payment systems to utilize any functionality of the application, which may not be so desirable sometimes. For example, an organization may wish to utilize the e-payment system to process a transaction without walking through any Web pages. In other words, it is expected that the transactional information is conveyed to the e-payment system over the Internet through a hypertext-transfer-protocol (HTTP) request, and the result code is passed back through an HTTP response. This scenario requires that our e-payment system expose some functionality for direct access. Without any doubt, our e-payment system must have a means to verify whether the transaction is originated from the sites of specific trusted organizations.

A. PTA Service

As a result, we realized the necessity of PTA in our e-payment application. PTA or so-called single sign on is not a new concept. Instead, it has been widely utilized in network computing to enable users to logon to the network and access resources from computers or domains where they have no accounts [26]. In this project, we refer PTA to a transitive trust-authentication process, which allows users to be seamlessly authenticated into the e-payment system from another trusted secure Web site, without visiting any e-payment login page. The PTA process can convey either already authenticated users or detailed payment information to an e-payment system.

Since our e-payment system is usually integrated into organizations' Web sites as a dedicated online-payment-processing engine, the PTA offers several significant advantages. First, users who signed on to the original Web sites will automatically obtain the access to the e-payment applications. Definitely, a technique is required to synchronize the user information stored in the e-payment systems with the original organization sites. Second, the e-payment system can be plugged into a legacy system of an organization as a third-party component to process real-time payment transactions if so desired. Third, organizations have more flexibility to choose to adopt either a stand-alone e-payment system or a plug-in e-payment service, or both under different circumstances. Fourth, PTA facilitates the scheme of e-payment applications to shift to an ultimate ASP model, where a central e-payment server handles online-payment transactions for different organizations. Comparing to the ultimate ASP model, the current ASP model should be adjustably called a semi-ASP model, as it actually provides a central hosting for all independent e-payment servers. Fifth, delivering access to the e-payment service in the ASP manner allows small to medium enterprises to eliminate the time and costs associated with the customization, installment, deployment, maintenance, and management of their own e-payment applications. Sixth, no data integration and synchronization is needed between the e-payment systems and the original organization systems, since our e-payment systems only need to verify that a request is originated from a trustworthy source (organization). Therefore, we conclude that PTA is the first step for the e-payment application toward a universally accessible Web service.

The PTA is implemented by a process, where an e-payment system evaluates a secure HTTP/HTTPS request sent by an organization. Two categories of data are conveyed by each HTTPS request string: data information and validation information. The former contains a list of (name, value) pairs, and the latter uses standard encryption techniques (e.g., Message Digest 5 [21] or Secure Hash Algorithm (SHA1) [32]) to comprise hash data and timestamps.

As more organizations request for the PTA function, including the organizations with e-payment systems already deployed, we decided to start a project aiming to construct PTA Web services. The goal of this project is twofold: one is to integrate a PTA service into each deployed legacy e-payment application and the other is to design a reusable PTA Web service so that it will become an integral part of future e-payment applications.

The empirical study reported in this paper is a case study, which was designed at the beginning of the project. This case study intends to investigate how the concept of Web services can benefit the design of our PTA service and e-payment applications and how to transform from an application-oriented architecture toward a Web-services-based SOA. It should be noted that this paper does not discuss service-interface descriptions, resolutions, and binding, such as syntax checking, semantics understanding, and automatic tools for service searching, matching, and binding. We realize that component-interface design is one critical yet difficult aspect in services-based software development; this paper only focuses on the architectural model of a services-oriented system.

B. Research Questions

At the beginning of the project, we identified three specific research questions as follows: 1) How will the project benefit from the concept of Web services? We planned to analyze the problems and issues that challenge us in designing a PTA function and examine whether the concept of Web services provides sufficient solutions to these problems. 2) Is a new Web-services-based SOA needed? We planned to explore whether we need a new Web-services-based SOA model to facilitate the PTA-service development. 3) How can an application-oriented architecture be migrated to a Web-services-based SOA? We planned to explore a pathway of transforming to a Web-services-based SOA, if there is one.

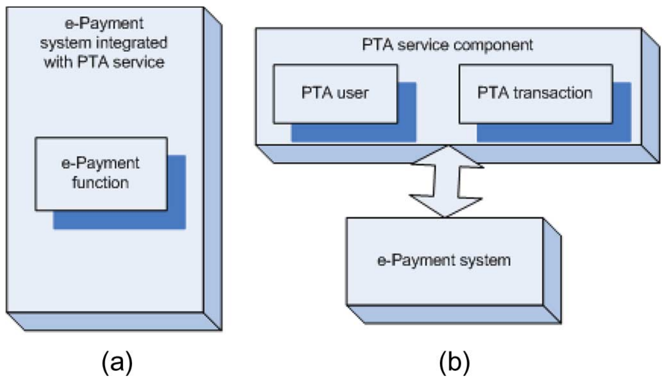


Fig. 2. Stage 1: End-to-end integration to legacy e-payment system.

III. Case Study

In this section, we will report our case study through the project. In accordance with how a PTA service is developed and integrated with an e-payment system, the project can be divided into three stages: 1) end-to-end-integration stage; 2) Web-services-enabled stage; and 3) Web-services-oriented stage. We will discuss each stage in detail. For clarification, when we use the term "service," we simply mean a function. If a Web component is developed into a Web service, we will use the term "Web service" explicitly. Similar to testing common Web applications, we used a popular unit test tool called JUnit [18] to create test cases.

Stage 1—End-to-End Integration: Due to the nature of the signed business contracts, the goal of our first stage of the project was to develop PTA functions and add them to legacy e-payment systems. The tasks were in a fixed-rate manner with urgent fixed time frames. Furthermore, at that time, there was only one architect available with several junior-level developers and quality-assurance (QA) people, so that there was not enough time to perform an overall design. As a result, for the first such customer, we implemented the PTA algorithm and integrated it into the original e-payment system. The redeployed integrated e-payment system thus became a customized self-contained application, as shown in Fig. 2(a). The issues of this strategy are apparent: The reusability and extensibility of the PTA part were low, because the code was mixed with the e-payment system and the interfaces between them were not well defined. However, we gained the initial experience of implementing a PTA function. Furthermore, some code pieces in this PTA function could be reused in later development. In addition, keeping in mind that our ultimate goal was to build a generic PTA Web service, we intentionally emphasized clear low-level application-programming interfaces (APIs) when we implemented the PTA function. The first task was finished in two months by one architect together with one junior developer and one QA resource, including requirement elicitation.

The second engagement is similar to the first one, which goal is to add the PTA service to another legacy e-payment system. At the time, more resources were assigned to participate in the engagement in order to redesign the architecture. However, because of lacking of Web-services experience and tight project deadline, we still did not adopt the concept of Web services for the engagement. Instead, by understanding that a Web service is, in fact, a Web component [9] and the paradigm of Web services intends to facilitate component engineering, we decided to construct a PTA-service component, while spending time learning Web-services technology. In the corresponding design, the original e-payment system acts as a back-end supporting system for the PTA-service component, as shown in Fig. 2(b).

The PTA-service component serves as the gate of the back-end e-payment application, replacing the original login process. With the new design, instead of embedding a uniform resource locator (URL) in the Web site of the organization to simply redirect it to the login page of the e-payment system, the organization generates an HTTPS request and sends it to the PTA-service component. There are two scenarios that an organization may request services from the PTA component: one is to pass through user credential authentication and the other one is to pass through transaction authentication. To match the two scenarios, the PTA-service component consists of two functional subcomponents: a PTA-user component and a PTA-transaction component, as illustrated in Fig. 2(b). Based on the incoming HTTPS requests, the PTA-service component will decide which subcomponent it will invoke.

In more detail, the first scenario is while a user, who has already been authenticated by the organization, wants to visit the e-Payment system. In this case, the organization will generate an HTTPS request including the user credential and send it to the PTA service component. The format of the HTTPS request is as follows:

https://epayment.com/organization/payer.do?userId=99 9999&timestamp=949924800&hash=b14ac94d2960e53d bb2f061b236d7a0a

Shown above is a simplified request that includes user credential: a user identification number. The timestamp and the hash value help to validate that the user has been authenticated by the organization. When the PTA-service component receives an HTTPS request as above, the PTA-user-service subcomponent will be invoked. The PTA user service will authenticate the user-identification information and use the user ID passed to load the user's personal information from the database and direct the user to the e-payment personal-statement Web page.

The second scenario is that the organization wants to pass through a payment transaction for processing. The format of the HTTPS request is as follows:

https://epayment.com/organization/payer.do?userId=99 9999&accountId=0011789&cardType=3&cardNumber= 6011001190119900&amount=100.00&timestamp=94992 4800&hash=b14ac94d2960e53dbb2f061b236d7a0a

Shown above is a simplified request that includes information of a payment transaction: a user-identification number, an account ID, a credit-card type, a credit-card number, and a payment amount. The timestamp and the hash value help to validate that the user has been authenticated by the organization. When the PTA-service component receives an HTTPS request as exhibited above, the PTA-transaction service subcomponent will be invoked. The PTA transaction will authenticate the transaction information, call the e-payment system to process the transaction, and pass back the result information to the organization through an HTTP response.

In either scenario, this end-to-end integration is accomplished through direct calls from the PTA-service component to the low-level functional APIs defined in the e-payment system. This strategy leaves the original e-payment system completely intact. However, each such PTA-service component is tightly coupled with the corresponding e-payment system. Since every deployed e-payment system is different, this strategy decreases the reusability of newly constructed PTA services. Another issue arising from this architectural model between a PTA component and an e-payment system is flexibility. In order to invoke a PTA service, each client generates an HTTP request string containing a hash value. The order of the parameter values on which to perform the hash algorithm is imperative. This order and the list of the parameter values are normally predefined and hard-coded into the PTA component. As a result, the PTA-service component cannot be easily reused for different e-payment applications.

The second engagement was finished in one month by two developers with one QA tester.

In this stage, from the testing perspective, testing has to be conducted over each integrated e-payment system with the PTA functionality. Reusability of test cases is limited.

Stage 2—Web-Services-Enabled Stage: To increase reusability and flexibility of the PTA-service component, we decided to construct an administration module for it. Through the administration module, an organization can dynamically set up or update the list of the parameter types and the order of the parameters for encryption. With the help of the administration module, a PTA service does not have to have the hash algorithm hard-coded any longer. In addition, the interface between the PTA service and the administration module can be standardized. Furthermore, the administration module is highly reusable to all PTA services. In other words, we decided to construct one administration module shared by multiple PTA services. As we discussed in the section of the project context, e-payment systems may reside at either the central server place or at individual organizations. Therefore, this administration module should be accessible via the Internet, which seems to fit into the picture of Web services. As a result, we decided to adopt the concept of Web services to implement the administration module.

The paradigm of Web services mainly embraces three categories of supporting facilities: communication protocols, service descriptions, and service discovery [6], [48]. Each category possesses its *ad hoc* standard, as Simple-Object

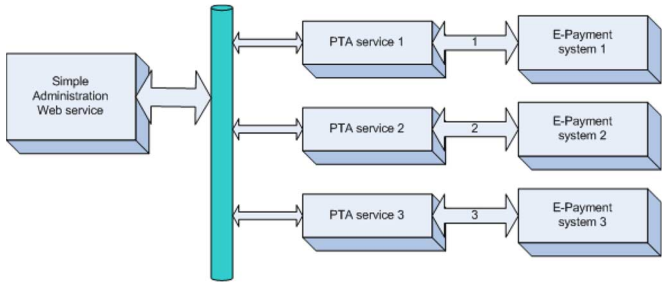


Fig. 3. Stage 2: Web-services-enabled stage to legacy systems.

Access Protocol (SOAP) [35], Web-Service Description Language (WSDL) [41], and Universal-Description Discovery and Integration (UDDI) [38], respectively. The SOAP acts as a simple and lightweight protocol for exchanging structured and typed information among Web services. The WSDL [41] is an XML-based description language that is used to describe the programmatic interfaces of Web services. The UDDI provides a mechanism to publish, register, and locate Web services. It should be noted that, here, we adopted a narrow definition of Web services that refers to an implicit definition of SOAP + WSDL + UDDI for the purpose of simplicity. Our decision implies a focus on the management of stand-alone Web services, instead of the compositions and the interactions between multiple Web services. Therefore, we utilized SOAP, WSDL, and UDDI in our project.

Thus, the architectural model between a PTA service and an e-payment system was altered, as illustrated in Fig. 3. Every deployed e-payment system is wrapped with a specific PTA Web service: e-payment system 1 with its corresponding wrapper PTA service 1; e-payment system 2 with PTA service 2; and e-payment system 3 with PTA service 3. Such an integrated system is referred to as a PTA-enabled e-payment application. The integration bus is introduced as a common communication channel for PTA-enabled e-payment applications. By integration bus, we mean that the communication channel is based upon SOAP message exchange, and constructed Web service is available from the UDDI registry. In addition, a PTA-administration Web service is constructed and shared by all PTA-enabled e-payment applications.

As shown in Fig. 3, using arrows with different notations between PTA-services wrappers and corresponding e-payment systems, the integrations between e-payment systems and their PTA-service wrappers vary from each other. The individual integrations depend on the corresponding e-payment systems. There are distinctions between PTA-service wrappers in this architectural model and those in the first stage. As shown in Fig. 3, the PTA-service wrappers all exhibit the same Web-service interfaces of registry and invocation to the integration bus. In addition, they are only accessible through the integration bus.

A simplified WSDL definition of the administration module is given as follows. A "ptaadministrationservice" Web service is defined. Here, we declare the WSDL namespace as the default namespace so that all elements belong to this namespace unless they have another namespace prefix. Here, we omitted all other namespace declarations to keep it simple. Inside the service element, we specify one port on which this service is accessible: PTAAdministrationRegistrationPort. Users can register a new PTA service to the administration Web service. The port has a unique name and a binding attribute. Our "ptaadministrationservice" can be accessed using the SOAP.

```
<definitions name = 'ptaadministrationservice'
  xmlns ='http://schemas.xmlsoap.org/wsdl/ '>

<import namespace =
  'http://localhost/ptaadministration/wsdl/definitions'
        location =
'http://localhost/ptaadministration/wsdl/ptaadministra
tionregistration.wsdl'/>

<binding name ='PTAAdministrationBinding'
  type ='defs:PTAAdministrationPortType'>
    <soap:binding style ='document'
  transport='http://schemas.xmlsoap.org/soap/http'/>
    <operation name ='RegisterUser'>
      <soap:operation soapAction =
'http://localhost/ptaadministration/RegisterUser'>
        <input>
          <soap:body use ='literal'/>
        </input>
        <output>
          <soap:body use ='literal'/>
        </output>
    </operation>
</binding>

<service name ='PTAAdministrationService'>
  <port name ='PTAAdministrationRegistrationPort'
binding='wsdlns: PTAAdministrationRegistrationBinding'>
      <soap:address
location='http://localhost/ptaadministration/wsdl/ptaad
ministrationregistrationservice.jsp'/>
  </port>
</service>
</definitions>
```

As shown above, a WSDL document "ptaadministrationregistration.wsdl" is imported. The service also specifies the binding mechanism "PTAAdministrationBinding." One operation is defined: "RegisterUser," which has one input parameter and one output parameter.

This administration Web service intends to enable users to adjust the authentication criteria after they are set. For example, one organization may decide to change the order of the parameter values for encryption. With the introduction of the administration Web service, this enforcement of the order of the parameter values does not have to be hard-coded into the PTA service; therefore, the reusability of PTA service is largely improved. Meanwhile, the single administration Web service is shared by all PTA-enabled e-payment applications through the common integration bus; thus, new PTA-enabled e-payment applications can be plugged into the integration bus and utilize the

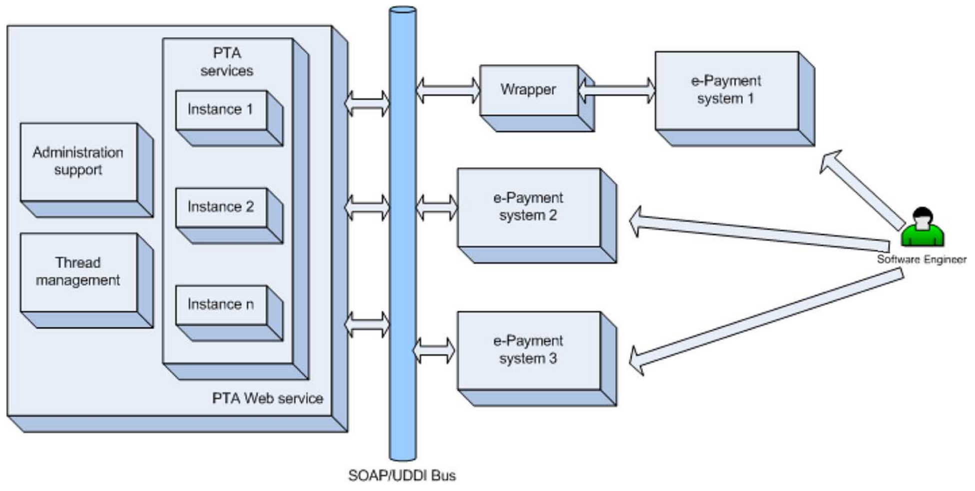


Fig. 4. Stage 3: Web-services-oriented stage.

administration Web service. Therefore, the scalability of the administration Web services is guaranteed. Considering that the single administration Web service can become a single point of failure, a fail-over administration Web service was constructed in case of outages from the primary administration Web service.

It can be seen that, in this architectural model shown in Fig. 3, the PTA service is actually a Web-service façade attached to existing e-payment applications. These facades reveal the same Web-service interface over diverse e-payment legacy systems. It can also be seen that this "Web-services-enabled stage" is, as a matter of fact, a direct application of the wrapper and mediator design patterns [11]. The key idea is that all e-payment applications are wrapped in such a way that they are accessible through a common interface, based upon the SOAP and WSDL. "Encapsulating how a set of objects interact" [11], the simple administration Web service acts as a mediator: It communicates with all e-payment systems through their wrappers.

In summary, at this stage, we divided the service-to-be PTA component into two parts: an administration part and a wrapper part. The administration part provides a capability of configuration and customization of the PTA service, while the wrapper part provides a PTA-service template that can be configured and customized. This stage is an intermediate step of adopting Web-services technique. While the compatibility with the emerging Web-services standards can be claimed by the architectural model in this stage, limited benefits are derived from such a solution. First, each PTA-service wrapper still implements a proprietary version of common services, such as logging in and notifications. Second, since each PTA-service wrapper is built on top of each individual e-payment system, the reusability of the PTA-service wrapper remains limited. Third, no central management of services is provided, such as monitoring and load balancing.

We used the same strategy to construct the PTA service for four organizations, two of which had similar e-payment systems. Therefore, we divided them into three categories, as shown in Fig. 3. The average development time for the first organization of a category was one month by two developers. The development time for the second task in the same category was two weeks by two developers.

In this phase, each Web-service-enabled PTA service has published interfaces that can be tested over different application implementations underneath. Adopting Ant [1] script to automate the test process in addition to the building process, we ran the same set of test suites of the PTA service against different e-payment systems through different configurations.

Stage 3—Web-Services-Oriented Stage: As we accumulated experience of building a customizable PTA Web-service component at the second stage, we were ready to construct a PTA-service component into a stand-alone Web service. The architectural model of the project was redesigned, as illustrated in Fig. 4. An integration bus is still used as the common communication channel for all applications. However, we built a stand-alone PTA Web service that is shared by all e-payment systems, either existing ones or future ones. As shown in Fig. 4, a PTA Web service is constructed as a self-contained and well-defined Web service, which is accessible via the integration bus by all e-payment systems. If needed, each e-payment system registers itself at the PTA Web service in order to obtain PTA services. Users are unaware of the existence of the PTA service and only access the e-payment systems as before. When the PTA service is needed, the corresponding e-payment system will prepare an HTTPS request to the PTA Web service and obtain an authentication result for further operations. As shown in Fig. 4, a user is not aware of the existence of the PTA service. Instead, he/she accesses an e-payment system to obtain PTA service.

As shown in Fig. 4, published by a set of interfaces defined in WSDL, the PTA Web service consists of three internal components: 1) a PTA-services component; 2) an administration-support component; and 3) a thread-management component. The PTA-services component provides pass-through authentication functionality, such as checking incoming user and payment-transaction credentials.

The administration-support component handles the registration of e-payment systems, as well as the configuration and customization of the basic PTA service in according to specific e-payment systems. Separated from the PTA-services component, the administration-support component not only enables clients to dynamically adjust the authentication criteria for each specific e-payment service but also implements some common services (e.g., notification and logging services) for the PTA service. As shown in Fig. 4, there is only one PTA Web service shared by all e-payment systems; therefore, all e-payments systems have to register to the PTA Web service before utilizing its services. Typically, the secret key of a specific organization has to be stored in the PTA Web service. Recalling the ASP model shown in Fig. 1, most of the organizations adopt our ASP solution, where application services are handled at a centralized location. Our new architectural model enables a hierarchical structure of the central server site. The PTA service is extracted from e-payment systems and set up as an independent Web service, which can reside on another server machine. Thus, the credential information can be protected by a dedicated layer.

Another critical component in the PTA Web service is a centralized-service-management unit, which provides system-level utility services, such as thread management and monitoring services. In our project, we call it a "thread-management" module, as shown in Fig. 4. Since the PTA Web service serves multiple e-payment systems simultaneously, it is essential to the success of the PTA Web service on how to guarantee the synchronization of different threads of operations, together with the load-balancing facility, to enhance performance and scalability. The service-management component can also provide additional services, such as message-broker functions that assist translation between application-specific data formats. As a result, this central component facilitates the applications toward an ultimate ASP model.

As shown in Fig. 4, inside of the PTA Web service, the PTA service acts as a container that supports multiple instances to perform load balancing. In our practice, the PTA service is deployed onto an application server, whose cluster-based load-balancing facility supports the functionality. The administration-support and thread-management components are shared by all PTA-service components. Furthermore, the administration-support and thread-management components can each contain a primary component and a secondary component for failing over in case of outages. Such a design can further enhance the scalability, reliability, and fault tolerance of the PTA Web service.

As shown in Fig. 4, for new e-payment systems that require the PTA service, we just embedded the code to access the PTA Web service according to its interfaces defined in WSDL (more detailed discussions about building a Web application using Web services as components will be described in the next section). For existing e-payment systems that need to add the PTA service, a simple wrapper will be added, which handles the communications with the PTA Web service via SOAP messages. In this way, the original e-payment legacy systems are largely intact and reused.

This architectural model offers a certain degree of extensibility and reusability for the PTA service. Several new e-payment systems were implemented by plugging-in the integration bus and automatically reusing the PTA Web service. In addition, several legacy e-payment systems were successfully plugged into the integration bus through corresponding wrappers to reuse the same PTA Web service. As we developed and deployed various e-payment systems, either new or old, we encountered some new requirements to the PTA service, such as adding new notification methods after the authentication process is complete. Instead of modifying any e-payment system, we upgraded the PTA Web service and redeployed it at some midnight time. The changes were automatically applied to all e-payment applications, without even known by any e-payment system. While we enabled the PTA service to more e-payment applications, we even added to the generic PTA Web service a new authentication approach through a variant SHAH1 [32] digital-signature approach. The original request came from one particular organization. After we implemented and integrated the approach into the generic PTA Web service, this new ability became reusable to all previously registered e-payment applications.

In summary, the major differences between the architectural model in Fig. 4 and that in Fig. 3 are threefold. First, the wrapper between an e-payment system and the PTA Web service becomes very thin, meaning that it does not contain any business logic regarding a PTA service. Instead, it acts merely as a proxy of the remote PTA Web service for the corresponding e-payment system, so that any e-payment system can access the common PTA Web service in a standard manner. Second, a stand-alone and self-contained PTA Web service is established containing any PTA-related business logic. This common PTA Web service can be shared and reused by any e-payment system. Any upgrade in the common PTA Web service can be automatically applied to all registered e-payment systems. Third, utility modules are implemented in the PTA Web service, such as load balancing and thread management. These utility services can enable higher scalability of the PTA Web service while providing better performance and management.

We implemented the PTA service for six organizations in this stage: three of them had legacy e-payment systems and the other three organizations request the whole package of e-payment system including the PTA service. It took us two months by two developers to develop the PTA Web service. In average, each legacy e-payment system took us one month by one developer, to adjust the legacy e-payment system to utilize the PTA Web service. For each of the new e-payment systems, the time, on average, to develop the code to utilize the PTA Web service was three weeks.

In this phase, test cases were designed against different Web-service components. The test cases were utilized and reused for newly plugged-in components.

IV. Web-Services-Oriented Engineering Model

After intensively implementing and deploying the 12 PTA engagements, we did not see immediate contracts on the to-do list. Upon predicting that more PTA engagements are forthcoming, we decided to reexamine the engineering model used in the project. By engineering model, we mean the procedure of constructing a new e-payment system using the published PTA

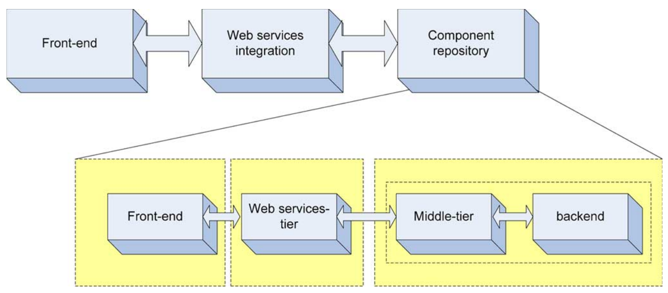


Fig. 5. Web-services-oriented engineering and architecture.

Web service or constructing a PTA Web service. Our goal is to construct an architectural model that can be reused to guide the design and development process of a Web application utilizing Web services as components.

Web-Services-Based SOA: The typical architectural model of Web application is a three-tier model [31]: front-end, middle tier, and back-end tier. The front-end tier manages interface presentations and Web-page input/output; the middle tier contains business logic; and the back-end tier stores data in databases. Using this three-tier model, considerations and compositions of Web-service components are mainly conducted at the middle tier.

Based upon the experiences gained from the project, we adjusted the three-tier model into a Web-services-based SOA model, as illustrated in Fig. 5. A Web-services-based Web-application design is organized as a three-tier structure: front-end, Web-services integration, and component repository. The front-end tier represents the interface of the application to clients. The component-repository tier contains software components that are needed for the application, exposed with Web-services interfaces. One Web-services-integration tier is used as a separate layer where components are integrated and plugged-in.

As shown in Fig. 5, each Web-service component in the repository exhibits another three-tier structure. Based upon the typical three-tier architectural model and some most recent Web technology, such as The Java 2 Platform, Enterprise Edition (J2EE) [16], our previous research yielded a Web-application architecture that is oriented to exposing Web services [44], [45]. As shown in Fig. 5, the internal architectural model of the PTA Web service is, in turn, divided into three tiers: front-end tier, Web-services tier, and back-end tier. Comparing with the traditional three-tier model, the front-end tier remains the same; the back-end tier contains the business-logic tier (i.e., middle tier), as well as the persistent tier (i.e., database tier); a new Web-service tier is introduced as an independent tier. The concept of application server in J2EE hides database details from Web-application developers. Therefore, from a software developer's perspective, the traditional middle tier and the database tier can be combined, thus, to simplify development considerations. The newly introduced Web-service layer enables a Web-service component to become a self-contained Web-application system associated with well-defined Web-services interfaces. The *ad hoc* industry standard WSDL [41] is used to define the Web-services interface, and SOAP is adopted as the information-exchange tool between the Web-services layer and the front-end and back-end tiers, respectively. To enable higher reusability, the three tiers (front-end, Web-services, and back-end tiers) of a component could be independently stored as three types of assets in the component repository, so they can be reused to compose complex and extensible applications. In practice of this project, for simplicity, we store the whole three-tier application as one component.

This hierarchical architectural model facilitates the integration of Web services as components, by providing a joint point for gluing Web services together. First, the model provides guidance on not only the application development utilizing Web services as components but also the application development serving as Web-service components. Second, this architecture implies that the system design starts from the system requirements' analysis and transformation into Web-services tier design. Third, this model proposes a practical Web-services-centered model, which will benefit organizations by accumulating Web-services component repository for future uses.

Engineering Services-Oriented Applications: Based on the architectural model and experiences from the case study, we formed an engineering process of developing Web-services-based system. Three essential steps are identified: 1) identify which Web service to use; 2) use the WSDL specification of the Web service to develop code to bind the Web service; and 3) use the WSDL specification of the Web service to develop code to invoke the Web service.

Using our project as an example, the process of engineering a PTA-based e-payment system starts from elicited requirements. Instead of starting right away to design every needed component, we started from considering available Web-services composition. Being aware of the PTA Web service existing in the component repository, we focused on how to utilize the PTA Web service. Our strategy was to embed code to access the PTA Web service according to its interfaces defined in WSDL. Since the PTA Web service is accessed via SOAP messages, the e-payment system needs to be able to generate SOAP requests. Similarly, since the replies from the PTA Web service are in the format of SOAP messages, the e-payment system needs to be capable of parsing SOAP replies. In one word, the newly constructed e-payment systems need to have the ability of handling SOAP communications.

We further summarized a Web-services-based application-development process into three phases, as depicted in Fig. 6. The initial phase is the collection of requirements, followed by the formalization of business requirements as a set of Web-service specifications in WSDL. This phase is basically human-intensive but can be facilitated by some WSDL editing tools. In our project, we wrote our own editor tool to generate WSDL interfaces.

The second phase contains two parallel development processes focusing on the front-end and the back-end, respectively. Engineers can work on the two processes independently and separately, both on the basis of the output of the first phase as contracts. Tools can provide support for the efficient production of each process. Our previous work yielded a J2EE-based Web-application code generator (WGenerator)

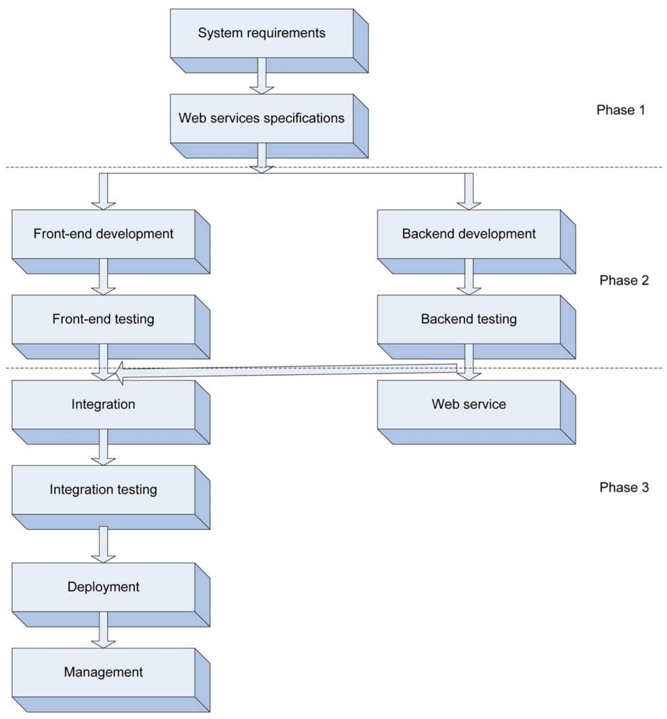


Fig. 6. Web-services-oriented engineering process.

[44], [45], which was used in this project to automatically generate significant part of code, both in the front-end and the back-end. The back-end in-memory simulation templates help generate in-memory back-end support for the front-end tier; so that, the front-end can be tested independent of the back-end development. The back-end test case templates help generate test cases for the back-end tier; so that, the back-end can be tested separately as well.

The third phase also contains two parallel parts. The first part contains four steps. The first part is the integration and deployment of the Web application with normal Web-browser interfaces. The front-end and back-end tiers are integrated through the Web-service interfaces. Integration tests are followed accordingly. The second part is the deployment of the Web application with Web-service interfaces, together with the corresponding Web-service test. This part can be extended from the test of the back-end tier directly. Fourth is the management of services. These four steps complete a life cycle of a solution development.

The second part is to expose some back-end functionalities as Web services, which, in general, is composed of five steps: 1) identify the functionalities to be turned into a Web service; 2) define the interfaces of the Web service; and 3) organize internal implementation of the Web services.

V. Use Case Discussions

Project Efforts Analysis: Table I summarizes the efforts that we spent on each task throughout the case study. As shown in Table I, in the first stage of end-to-end integration stage, we performed two tasks in building a PTA service into two legacy e-payment systems. In the first task, we did not consider much of the reusability aspect, and we took a lot of time gathering and eliciting requirements for a PTA service. It took two-person months. The second task is similar to the first task, so some development time was saved. However, it took time to refractor the PTA service into a component. As a result, the time spent was the same as that for the first one.

Tasks 3–6 belong to stage two—Web-services-enabled stage: all four tasks built a Web-services-enabled PTA service for four legacy e-payment systems divided in three categories. Building time for the first e-payment system in each category was the same as two-person months. The reason is that for each e-payment system, constructing different facades of the PTA service took almost the same time to adjust to the specific requirements and environments of each category of e-payment system. Meanwhile, the time spent on the trial of Web-services technology was also significant. Thus, the efforts were almost the same as those spent in the first stage. The building time for the second e-payment system decreased to half due to reuse.

Tasks 7–12 belong to stage three—Web-services-oriented stage, after a PTA Web service was built. Three tasks (i.e., tasks 7–9) enable three legacy e-payment systems to utilize the PTA Web service, and the other three tasks (i.e., tasks 10–12) built the PTA Web-service invocation engine in three newly built e-payment systems. The time spent on the former three tasks was one-person month, and the time spent on the latter three tasks was three-person weeks. We can see that, after the PTA Web service was created, the time spent on building the engine in e-payment systems dropped significantly.

From our case study, we also observed that the lack of standardization and a structural model of Web-services-based SOA and a Web-services-oriented engineering process prevented quick adaptation and slowed down the transition toward a Web-services-based SOA within an enterprise application. Rather than a one-step direct transition, the process exhibits a sequence of three stages: end-to-end integration stage, Web-services-enabled stage, and Web-services-oriented stage. As we discussed in the case study, each stage is triggered by the introduction and acceptance of a new architectural model. Meanwhile, each stage is characterized by an architectural-model adoption curve. The overlay of multiple adoption curves defines the overall adoption of SOA.

As of now, we find ourselves in both the Web-services-enabled stage and Web-services-oriented stage, which is defined by a relatively quick adoption cycle, but realizes only a portion of the possible benefits. For some of the deployed e-payment systems, it is likely that we will still stay in the second stage due to the consideration of decreasing budget and time. For the new e-payment projects and some deployed systems with additional budgets, we will adopt the Web-services-oriented strategy.

Experiences and Lessons Learnt: Based upon the experiences gathered from the project, we will now answer the research questions that we raised at the beginning of the case study.

Benefits from adopting the concept of Web services: At each stage of this project, we analyzed the issues and challenges with which we faced and examined what solutions the concept of Web services can provide. As shown in our case

TABLE I
EFFORTS SPENT ON EACH TASK OF THE CASE STUDY

Stage	Task number	Task description	Deployed applications	Effort (person*month)
Stage 1	1	Build in the PTA service into a legacy e-Payment system	1	2
	2	Build in the PTA service into a legacy e-Payment system as a reusable component	1	2
Stage 2	3	Build a Web services-enabled PTA service to a legacy e-Payment system in category 1	1	2
	4	Build a Web services-enabled PTA service to a legacy e-Payment system in category 2	1	2
	5	Build a Web services-enabled PTA service to a legacy e-Payment system in category 3	1	2
	6	Build a Web services-enabled PTA service to the second legacy e-Payment system in category 3	1	1
Stage 3	7-9	Build a Web services-oriented PTA service to a legacy e-Payment system (after the PTA Web service was built)	3	1
	10-12	Build a Web services-enabled PTA service to a newly built e-Payment system (after the PTA Web service was built)	3	0.75

study, the concept of Web services helps us solve all the problems. Here, we will summarize the benefits that the concept of Web services brought to our project in the following six aspects: 1) service integration; 2) testing; 3) maintainability; 4) security; 5) reusability; and 6) central management.

1) Service integration: In the first phase, a PTA service is embedded in every specific e-payment application; therefore, the PTA service has low reusability. In the second phase, every PTA service generally possesses the same service APIs; therefore, the reusability is higher for the similar types of supportive e-payment systems. However, since each PTA service is customized to specific e-payment system, its ubiquity is limited. In the third phase, the PTA service is implemented as a Web service. Its functionality is defined and fixed at the very beginning, and the communication interface between the PTA Web service and the e-payment system is also predefined. Therefore, the integration between the PTA Web service and an e-payment system is easy and simple. In addition, the PTA Web service can be considered as a reusable asset that can be combined with other components in ways not conceived by the originators. Developers will benefit from new applications being developed more quickly as a result of these kinds of reusable Web services.
2) Testing: In our case study, we examined the feasibility to perform testing in different stages. In the first phase, since the PTA functionality is embedded into each individual e-payment system, reusability of test cases of PTA functionality is limited. In the second phase, each PTA-enabled e-payment exhibits the same PTA service interface; thus, testing cases for the PTA service can be performed upon every system. In the third phase, since the PTA service is constructed into a PTA Web service to be shared by every e-payment system, testing over the PTA service can be conducted at one centralized place and does not have to be repeated over every e-payment system. Therefore, Web-services-based SOA facilitates the testing process.
3) Maintainability: In the first phase, since the PTA service has high coherence with the integrated e-payment systems, maintenance has to be conducted on each system basis. In the second phase, since the only difference between different PTA services exist in the different invocations over corresponding e-payment systems, some changes on one PTA-service case can be easily deployed to other PTA-service cases. In the third stage, since all PTA-service instances are managed in a centralized manner, maintenance becomes much easier.
4) Security: The separation of a Web service from its invocation environment enhances security. As in our case study, each e-payment system has its own security protection. Isolating the PTA service as a Web service and placing it on a separate server adds one more layer of security safeguard to the sensitive information contained in the PTA Web service as well as e-payment systems.
5) Reusability: In the first stage, the code of the PTA service is tightly coupled with corresponding e-payment systems; therefore, its reusability is limited to low-level functional segments. In the second stage, the reusability of the PTA service extends to the whole service component if the invocation environment (i.e., the e-payment system) is the same. However, reusability is limited to manual installation and redeployment of the PTA-service component. In the third stage, the PTA service is implemented as a self-contained Web service. Each e-payment system can independently invocate its service. Moreover, since the PTA Web service is accessible via XML-based SOAP messages, an e-payment system implemented using

different languages and running on different platforms can still reuse the same service.

6) Central management: In the first stage, any change to the PTA service has to be applied to every PTA-service instance that needs it. In the second stage, changes to a category of PTA service can be applied once and redeployed to the same category of integrated e-payment systems. In the third stage, since the PTA service is implemented as a Web service, any change to the PTA service only has to be conducted at the PTA Web-service site once and be applied to every e-payment system that uses the PTA Web service.

Necessity of Web-services-based SOA: A Web-services-oriented application development is twofold: 1) develop an application utilizing Web services as components and 2) build an application that exposes part or all of the functionalities as Web services. The first style focuses on quickly incorporating existing Web services into new applications; the second style focuses on building reusable Web services for future uses. As we illustrated in our case study, a new architectural model will facilitate Web-services-oriented Web-application development and Web-services construction.

From our case study, we found that our Web-services-based SOA model has the following four merits that facilitate Web engineering: 1) developer roles; 2) return on investment; 3) code mobility; and 4) parallelism in development.

1) **Developer Roles**: Our architectural model enforces layered development, i.e., development is distributed to multiple teams. Each team has a set of specific roles identified for developers, with each role requiring a set of skills. For instance, the service layer needs developers that have experience in data formats, business logic, persistence mechanisms, and transaction control. A front-end developer has to be familiar with the technologies such as JSP, HTML, and Struts tags. To the extent that developers can specialize, they will excel at their craft in a particular layer of the application. Lower-level developers can be assigned to focus on specific development tasks. For example, an architect was in charge of designing the service layer and the integration at the service layer, while a JSP developer was assigned to the front-end development task.
2) **Return on Investment**: The isolation of the Web-service layer and the engineering of the service layer has the benefit of a better return on the investment made in the Web-application development. Our case study examined two options to implement the PTA service. First was developing the PTA service to be embedded into e-payment applications; second was developing the PTA service as a separate Web service. Our case study shows that the latter Web-services solution outlives the original application life cycle. In our project, the constructed PTA Web service has been reused for newly developed e-payment systems without individual changes and redeployment. As shown in Table I, excluding the experiences earned along with the list of tasks, the efforts of establishing a customized PTA service was decreased from the original two-person months to 0.75-person months.
3) **Code Mobility**: The separation of the Web-services layer offers location transparency; therefore, code mobility becomes a reality. The lookup and dynamic binding between the PTA service and an e-payment service means that the client does not care where the Web service is located. Therefore, the client has the flexibility to move services to different machines or to move a service to an external provider.
4) **Parallelism in Development**: The benefit of multiple layers means that multiple developers can work on those layers in parallel and independently. Developers communicate using interface contracts at the start of a project and are able to create their parts independently of one another.

Pathway of transforming to a Web-services-based SOA: As shown in the case study, due to budget and time-frame requirements, transformation from an application-oriented architecture to a Web-services-based SOA normally cannot be realized in one step. Instead, a three-stage pathway is more realistic: 1) end-to-end integration; 2) Web-services-enabled stage; and 3) Web-services-oriented stage. This pathway can be applied to other projects and organizations for guiding a smooth transformation.

From this case study, we learned that the code modularization and code-by-interface are two critical strategies that facilitate Web-services identification and construction. If the code is well designed following the common object-oriented approach considering information hiding and encapsulation (e.g., different modules only communicate with each other via message passing), it is easier to transform some functionalities of an application into Web services. Meanwhile, another object-oriented design strategy of design-by-interface [3] was proved to be highly valuable in our case study. To a certain degree, changing a component into a Web service implies changing its interface definition into one that is defined using Web-services definition language (e.g., WSDL). Therefore, if a component is well designed and developed with clear interface definitions, changing it into a Web service becomes much easier.

The wrapper and mediator design patterns [11] were proved to be practical in our case study. These two patterns help to identify what needs to be placed into the administration component and what needs to be left in the generic PTA-service component. They help us promote loose coupling between the administration component and individual PTA services, thus, enhance the flexibility and usability of the PTA service.

Limitations: In this case study, we adopted a narrow definition of Web services, which has an implicit assumption that Web services means SOAP + WSDL + UDDI. This definition reflects only the functionality of Web services. Most stakeholders, however, now agree that it is imperative for successful Web services to be trustworthy, which require much broader attributes such as reliability, security, trust, fault tolerance, and orchestration [43]. There are several ongoing efforts aiming at addressing these aspects. For example, two software-industry giants, IBM and Microsoft, combine their efforts and propose

WS-Security [40] standard, which can be used to accommodate a wide range of security models and encryption technologies. For another example, BEA, IBM, and Microsoft's Business Process Execution Language for Web Services [5] is a programming abstraction that allows developers to compose multiple synchronous and asynchronous Web services into an end-to-end business flow. However, as aforementioned, the goal of this case study was to build a PTA Web service that can be used by different e-payment systems, which was the first step toward a real Web-services-oriented e-payment system. Therefore, this case study focused on exploring a way to turn part of the functionalities of a system into a Web service. In other words, in this project, we did not consider how to make the PTA Web service trustworthy. Toward our ultimate goal of turning the entire e-payment system into a Web service, we will have to weave in considerations about trustworthiness attributes.

It should also be noted that, for our Web-services-enabled architectural model (i.e., shown in Fig. 4) to work, it is necessary that the e-payment applications that rely on the PTA service are aware of the "ontology" used by the PTA Web service defined in WSDL. In other words, every e-payment application using the PTA Web service will be coded in such a way that it knows the names of the operations and their associated parameter types. This is, as a matter of fact, the limitations of the WSDL; there are ongoing efforts in the area of semantic Web-enabled services aiming at providing means for relaxing this "common ontology" assumption [22]. However, this case study did not consider the semantic meaning of Web-services interfaces.

New Research Issues Revealed: Our case study reveals a new research issue regarding Web-services construction. The literature of current efforts on creating a Web service focuses on how to build its functionality, how to define its interfaces using standard WSDL, how to compose it using other published Web-service components, how to publish it, and so on [5], [6], [9], [14], [15], [22], [30]. In other words, these efforts imply that the process of constructing a Web service is one that defines the interfaces of a Web application using specific Web-services description languages (e.g., WSDL) and publishes it using standard language (e.g., UDDI). Our case study points out that, to enable a Web service to serve different users simultaneously, the Web service itself has to weave in considerations, such as parallel processing and load balancing. Although these requirements exist for generous Web-application systems, the current technologies do not carry sufficient facilities to support them in the context of Web services. For example, since Web services are hosted by corresponding service providers and are executed through remote invocation via the Internet, load balancing of Web services has to consider dynamic availability and reliability of Web services. These issues apparently require further research efforts.

In addition, to enhance the usability of a Web service for different users, the Web service should have a certain degree of flexibility to serve different users. Our solution was to build into our PTA Web service as an administration module, so that, the PTA service can be configurable and customizable for different organizations. However, considering that this configuration and customization requirements might be applicable to a lot of Web services, maybe it should be considered and designed in a more generic way. In other words, we predict that a generic Web-services platform, which is analogous to the application-server platform to Web applications [23], should be established as a common running environment for Web services. This Web-services platform should provide system-level utilities, such as load balancing, configuration, and customization.

As our PTA Web service is used by multiple e-payment systems, it appeared that, at some peak traffic time, the PTA Web service had too heavy load, and its availability was lowered down for some registered organizations. Although our PTA Web service included a load-balancing component, it seemed that a single instance of the Web service could not afford the heavy loads. Our solution was to provide a repository of multiple PTA Web-service instances to support increasing service requests. Therefore, our case study points out a new research issue of Web services. Since a Web service is a published Web application that can be searched and used by any application on the Internet, it has to consider how to maintain its availability during heavy request loads. One possible solution is to automatically breed new instances of the Web service when needed and automatically annihilate unnecessary ones. However, how to effectively and efficiently monitor and manage dynamically generated Web-service instances remains a challenge.

VI. Related Work

It has been extensively accepted that applying the concepts of Web services and SOA in business-to-business (B2B) applications leads to the future of business on Internet. However, how to adapt the traditional application-oriented software-engineering methodologies to the services-oriented paradigm remains a challenge [27].

Radeka [29] reports experience in implementing a Web-service project in 90 days. Four elements are identified as critical aspects for delivering a successful Web-service project: scenarios, business-process modeling, carefully scoped components, and a flexible architecture- and standards-based interfaces. Sikka [34] also identifies open challenges for services-oriented data management: Web-service technology, model-driven development methodology, and service platform. Kim *et al.* [19] conclude that B2B framework standards can facilitate interoperability. Lee *et al.* [20] emphasize the importance of securing e-Business systems and reviewed current techniques and standards for securing knowledge-management systems.

A variety of research work has been conducted to expedite and automate the development of services-oriented Web applications. Architecture-based or so-called model-driven methodology has been used to simplify and automate the development of Web applications. Many researchers and practitioners have been focusing on extending existing models to serve Web services-specific features. Brambilla *et al.* [4] report an industrial experience in using Web services to integrate both data- and process-intensive Web applications. Their work focuses on visual modeling and declarative description of integration of traditional data sources with Web services. Proposing a set of primitives, they extend the Web modeling language to model Web-services-oriented integration. Quartel *et al.* [27] propose a conceptual model for service-oriented design, applying a modeling language: interaction system-design language derived from distributed computing. Stojanovic *et al.* [36] propose an SOA modeling and design approach based upon a concept of "service component" and UML modeling language. Ganesh *et al.* [12] propose a multitier SOA-based framework-supporting multiple channels. In contrast with their work focusing on conceptual modeling, our project proposes a practical services-oriented architectural model to guide the design and development of services-oriented Web applications. We used our own code generator (WGenerator) [44], [45] to generate services-oriented code.

Yang [42] proposes a concept of "service component" for Web-services composition and specialization. A service component contains two parts: interface specification and construction specification. While his work presents an abstract approach for specifying service components for using as a packaging mechanism for Web-services composition, this paper focuses on how to turn an existing component into a services-enabled component.

From the loosely coupling requirement of Web-services-based design, Pasley [25] conclude that a WSDL-first approach should be utilized for Web-services development. Our approach starts from analyzing business requirements and design Web-services interfaces. Therefore, our method matches Pasley's proposal.

Shaiva [33] proposes a services-oriented adaptive-component architecture, which utilizes an "adapter" mechanism to support adaptive services-oriented components. An iterative approach is proposed to realize a phased migration of legacy systems. Comparing to their work, we use an autonomous administration module inside of the PTA Web service to allow dynamic user adaptation.

Wieringa *et al.* [39] propose design guidelines on functional decomposition toward modular Web-services architecture. Papazoglou and Yang [24] propose functional- and nonfunctional-service-design guidelines for Web services and business processes. In their methodology, a business process is defined as a set of collaborative Web services. However, although there are a variety of principles about Web-services-oriented design and development, there lacks widely accepted design frameworks and architectural models.

Quartel *et al.* [27] propose a generic services-oriented design approach, which identifies generic design milestones and assessing design conformance at different abstraction granularities. Four stages are identified: business-process specification, application-service specification, application-service design, and application-service implementation. Contrasting to their work, our three-stage approach illustrates to a practical pathway migrating an application-oriented design to services-oriented design. In addition, while their work proposes a generic services-oriented design approach, our work proposes an architectural model to guide a feasible services-oriented design illustrated in an industrial case study.

Zhang and Yang [49] propose an approach in exposing legacy-system functionality as Web services. Their approach contains four steps: 1) service identification; 2) legacy-system understanding; 3) agglomerative-clustering analysis; and 4) service packaging and integration. Their work focuses on automatically extracting functionalities from legacy systems as publishable Web services; this paper focuses on how to build new functionalities as Web services so that legacy systems can be left intact.

There is also numerous literature reporting experiences and lessons from building systems using SOA. Dang *et al.* [7] report experience in building Tele-Immersion, a teleconferencing system using SOA. Baglietto *et al.* [2] report experience in deploying services-oriented technology in business communities. They both report business-requirement analysis and a tuned architecture to support stepwise deployment. Shan report experience in constructing an SOA-based online e-bank system. Rabhi *et al.* [28] report experiences of building an SOA-based application for capital market. In contrast to their case studies on applying SOA to design a new system, this paper focuses on exploring a practical way to migrate an existing system to an SOA-enabled system. In addition, we explore an architectural model that guides the migration and development.

Finally, our previous research yields a two-tier model-view-controller (MVC) architecture (TTMVC) for Web-application development [44], [45]. Utilizing the popular MVC paradigm for each application tier from the G front-end to the back-end components and databases, our TTMVC clearly abstracts an object-oriented component-layered structure for Web applications. This structured architecture provides a foundation for the automatic code generation of Web applications. However, our previous architecture was tailored for stand-alone Web applications and does not support Web services. The overlapping on action layer between the front-end and back-end limits the reusability of the back-end system as a Web service. In addition, our previous architecture does not leave the space for out-of-shelf third-party Web services to be plugged in applications.

VII. Conclusions

In this paper, we presented a real-life industrial case study over an on-going project concerned with the design and development of a PTA Web service for online e-payment applications. In accordance with how a PTA service is developed and integrated with the corresponding e-payment system, our strategies can be categorized in three stages: end-to-end integration stage, Web-services-enabled stage, and Web-services-oriented stage. Derived from real-world industrial experiences, this three-stage pathway can be applied to a broad range of Web applications to guide smooth transformation from an application-oriented design and development model toward a Web-services-oriented model. Furthermore, the case study contributes an engineering process that leads to practical Web-services-oriented software development. New research issues revealed by our case study are also reported.

We plan to continue to enhance our Web-services-based SOA. Our future work will be focused on working toward an engineering process on the basis of our architectural model and constructing a system environment where all aspects of our architecture-based development are supported. The ultimate goal of our future work intends to provide efficient production of Web-services-supported Web applications. In addition, we plan to investigate rapidly integrating application-service components through advanced Web-services discovery and optimal composition technology [48].

REFERENCES

[1] *Ant*, 2006. [Online]. Available: http://ant.apache.org/

[2] P. Baglietto, M. Maresca, A. Parodi, and N. Zingirian, "Deployment of service oriented architecture for a business community," in *Proc. 6th IEEE EDOC Conf.*, Lausanne, Switzerland, Sep. 17–20, 2002, pp. 293–304.

[3] G. Booch, I. Jacobson, and J. Rumbaugh, *The Unified Modeling Language User Guide*. Reading, MA: Addison-Wesley, 1998.

[4] M. Brambilla, S. Ceri, P. Fraternali, R. Acerbis, and A. Bongio, "Model-driven design of service-enabled web applications," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Baltimore, MD, Jun. 14–16, 2005, pp. 851–856.

[5] IBM Corporation, *Business Process Execution Language for Web Services (BPEL4WS)*, 2002. Version 1.0.

[6] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the web services web: An introduction to SOAP, WSDL, and UDDI," *IEEE Internet Comput.*, vol. 6, no. 2, pp. 86–93, Mar./Apr. 2002.

[7] G. Dang, Z. Q. Cheng, S. Y. Jin, T. Yang, and T. Wu, "A service-oriented architecture for tele-immersion," in *Proc. IEEE EEE*, Hong Kong, Mar. 29–Apr. 1, 2005, pp. 646–649.

[8] Y. Deshpande, S. Murugesan, A. Ginige, S. Hansen, D. Schwabe, and M. Gaedke, "Web engineering," *J. Web Eng.*, vol. 1, no. 1, pp. 3–17, 2002.

[9] C. Ferris and J. Farrell, "What are web services?"*Commun. ACM*, vol. 46, no. 6, p. 31, Jun. 2003.

[10] P. Fraternali and P. Paolini, "Model-driven development of web applications: The autoweb system," *ACM Trans. Inf. Syst.*, vol. 28, no. 4, pp. 323–382, 2000.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Reading, MA: Addison-Wesley, 1994.

[12] J. Ganesh, S. Padmabhun, and D. Moitra, "Web services and multi-channel integration: A proposed framework," in *Proc. IEEE ICWS*, San Diego, CA, Jul. 6–9, 2004, pp. 70–77.

[13] A. Ginige and S. Murugesan, "The essence of web engineering managing the diversity and complexity of web application development," *IEEE Multimedia*, vol. 8, no. 2, pp. 22–25, Apr.–Jun. 2001.

[14] N. Gold, C. Knight, A. Mohan, and M. Munro, "Understanding service-oriented software," *IEEE Softw.*, vol. 21, no. 2, pp. 71–77, Mar.–Apr. 2004.

[15] P. Holland, "Building web services from existing application," *AI J.*, pp. 45–47, 2002.

[16] *J2EE*, 2006. [Online]. Available: http://java.sun.com/j2ee

[17] *JRun*, 2006. [Online]. Available: http://www.macromedia.com

[18] *JUnit*, 2006. [Online]. Available: http://www.junit.org/index.htm

[19] D. J. Kim, M. Agrawal, B. Jayaraman, and H. R. Rao, "A framework for the comparison of business-to-business e-service solutions," *Commun. ACM*, vol. 46, no. 12, pp. 317–324, Dec. 2003.

[20] J. K. Lee, S. Upadhyaya, H. R. Rao, and R. Sharman, "Secure knowledge management and the semantic web," *Commun. ACM*, vol. 48, no. 12, pp. 48–55, Dec. 2005.

[21] *MD5*, 2006. [Online]. Available: http://www.ietf.org/rfc/rfc1321.txt

[22] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid, "Composing web services on the semantic web," *Int. Journal Very Large Data Bases*, vol. 12, no. 4, pp. 333–351, Nov. 2003.

[23] R. Monson-Haefel, *Enterprise JavaBeans*. Sebastopol, CA: O'Reilly & Assoc. Inc., 2001.

[24] M. P. Papazoglou and J. Yang, "Design methodology for web services and business processes," in *Proc. 3rd VLDB-TES Workshop*, Hong Kong, Aug. 2002, pp. 54–64.

[25] J. Pasley, "How BPEL and SOA are changing web services development," *IEEE Internet Comput.*, vol. 9, no. 3, pp. 60–67, May/Jun. 2005.

[26] *PTA*, 2004. [Online]. Available: http://www.mcpmag.com/columns/article.asp?EditorialsID=761

[27] D. Quartel, R. Dijkman, and M. V. Sinderen, "Methodological support for service-oriented design with ISDL," in *Proc. 2nd ACM Int. Conf. Service Oriented Comput.*, New York, Nov. 15–19, 2004, pp. 1–10.

[28] F. A. Rabhi, F. T. Dabous, H. Yu, B. Benatallah, and Y. K. Lee, "A case study in developing web services for capital markets," in *Proc. IEEE Int. Conf. EEE*, Taipei, Taiwan, R.O.C., Mar. 28–31, 2004, pp. 38–41.

[29] K. Radeka, "Designing a web services project for maximum value: The 90 day challenge," in *Proc. ACM Conf. OOPSLA Practitioners Rep.*, Seattle, WA, Nov. 4–8, 2002, pp. 1–10.

[30] J. Ray and A. Ramanujan, "Understanding web services," *IEEE IT Prof.*, vol. 3, no. 6, pp. 69–73, Nov./Dec. 2001.

[31] O. Robert, H. Dan, and E. Jeri, *Client/Server Survival Guide*. Hoboken, NJ: Wiley, 1999.

[32] *SHA1*, 2006. [Online]. Available: http://www.w3.org/PICS/DSig/SHA1_1_0.html

[33] V. Shaiva, "Designing adaptive components for a services oriented architecture," in *Proc. IEEE Int. Conf. ITRE*, Newark, NJ, Aug. 11–13, 2003, pp. 390–394.

[34] V. Sikka, "Data and metadata management in service-oriented architectures: Some open challenges," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Baltimore, MD, Jun. 14–16, 2005, pp. 849–850.

[35] SOAP, *Simple Object Access Protocol (SOAP) 1.2*, May 2003. [Online]. Available: http://www.w3.org/TR/soap12-part1/

[36] Z. Stojanovic, A. Dahanayake, and H. Sol, "Modeling and design of service-oriented architecture," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Hague, The Netherlands, Oct. 10–13, 2004, pp. 4147–4152.

[37] L. Tao, "Shifting paradigms with the application service provider model," *Computer*, vol. 34, no. 10, pp. 32–39, Oct. 2001.

[38] *UDDI. Universal Description, Discovery, and Integration, UDDI Specification Version 3*, 2004. [Online]. Available: http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3

[39] R. J. Wieringa, H. M. Blanken, M. M. Fokkinga, and P. W. P. J. Grefen, "Aligning application architecture to the business context," in *Proc. 15th Int. CAiSE*, Klagenfurt, Austria, Jun. 16–18, 2003, pp. 209–225.

[40] *WS-Security*, 2004. [Online]. Available: http://www-106.ibm.com/developerworks/webservices/library/ws-secure

[41] *WSDL. Web Services Description Language*, 2004. [Online]. Available: http://www.w3.org/TR/wsdl

[42] J. Yang, "Web service componentization," *Commun. ACM*, vol. 46, no. 10, pp. 35–40, Oct. 2003.

[43] J. Zhang, "Trustworthy Web services: Actions for now," *IEEE IT Prof.*, vol. 7, no. 1, pp. 32–36, Jan./Feb. 2005.

[44] J. Zhang and J. Y. Chung, "An architecture for building web service applications," in *Proc. ICWS*, Las Vegas, NV, Jun. 23–26, 2003, pp. 265–271.

[45] J. Zhang and J. Y. Chung, "Mockup-driven fast-prototyping methodology for web application development," *Softw. Pract. Exp. J.*, vol. 33, no. 13, pp. 1251–1272, 2003.

[46] L.-J. Zhang, H. Chang, T. Chao, J.-Y. Chung, Z. Tian, J. Xu, Y. Zuo, S. Yang, and O. Ao, "A manageable web services hub framework and enabling technologies for e-sourcing," in *Proc. IEEE Conf. SMC*, Tunisia, Oct. 6–9, 2002, pp. 6–13.

[47] L.-J. Zhang, T. Chao, H. Chang, and J.-Y. Chung, "XML-based advanced UDDI search mechanism for B2B integration," *Electron. Commer. Res. J.*, vol. 1, no. 3, pp. 25–42, 2003.

[48] L. J. Zhang and B. Li, "Requirements driven dynamic services composition for web services and grid solutions," *J. Grid Comput.*, vol. 2, no. 2, pp. 121–140, Jun. 2004.

[49] Z. Zhang and H. Yang, "Incubating services in legacy systems for architectural migration," in *Proc. 11th IEEE Asia-Pac. Softw. Eng. Conf.*, Busan, Korea, Nov. 20–Dec. 3 2004, pp. 196–203.

[50] [Online]. Available: http://www.paypal.com

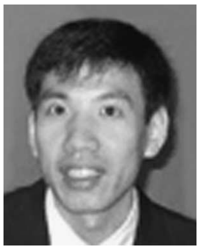
Jia Zhang (M'03) received the Ph.D. degree in computer science from the University of Illinois at Chicago, in 2000.

She is currently an Assistant Professor with the Department of Computer Science, Northern Illinois University, DeKalb, and also a Guest Researcher with the National Institute of Standards and Technology, Gaithersburg, MD. Her current research interests center around Services Computing, with a focus on reliability, integrity, security, and interoperability. She also has seven years of industrial experience as Software Technical Lead in Web-application development. She has published over 70 journal papers, book chapters, and conference papers.

Dr. Zhang is currently serving as an Associate Editor of the International Journal of Web Services Research. She is Program Vice Chair of the IEEE International Conference on Web Services (ICWS 2008 & 2007 & 2006). She is a member of IEEE and the Association for Computing Machinery.

Carl K. Chang (S'79–M'82–SM'88–F'01) received the Ph.D. degree in computer science from Northwestern University in 1982. He is currently a Professor and Chair with the Department of Computer Science, Iowa State University, Ames. His research interests include requirements engineering, software architecture, and net-centric computing.

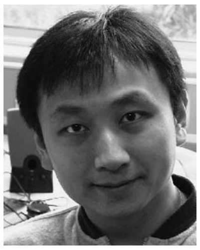
Dr. Chang is a founding member of the IEEE International Requirements Engineering Conference (RE), and served as the General Chair of ICRE 2000 and RE2003. He also chaired the steering committee for the 2004 IEEE-CS/IPSJ International Symposium on Applications and the Internet (SAINT) after serving as the Program Chair of SAINT2002 and General Chair of SAINT2003. In 2005, he was the General Chair of the IEEE International Conference on Web Services and IEEE International Conference on Services Computing. In 2006, he was the General Chair of the IEEE 30th Annual International Conference Computer Software and Applications Conference. He is also active in educational activities and spearheaded the Computing Curricula 2001 project, which was jointly sponsored by the IEEE Computer Society, Association for Computing Machinery, and the National Science Foundation. He was the Editor-in-Chief for IEEE SOFTWARE in 1991–1994. He is a Fellow of the American Association for the Advancement of Science and the 2004 President of the IEEE Computer Society.



Liang-Jie Zhang (M'99–SM'03) received the B.S. degree in electrical engineering from Xidian University, Xi'an, China, in 1990, the M.S. degree in electrical engineering from Xi'an Jiaotong University, Xi'an, in 1992, and the Ph.D. degree in computer engineering from Tsinghua University, Beijing, China, in 1996.

He is currently a Research Staff member with the Services Technologies Department, IBM T. J. Watson Research Center, Yorktown Heights, NY. He has been leading service-oriented architecture and Web services for Business Consulting Services and Industry Solutions research since 2001. He is the Founding Chair of the Services Computing Professional Interest Community, IBM Research, and lead professional activities for IBM's services-computing discipline. In 2004 and 2005, he was the Chief Architect of Industrial Standards with the IBM Software Group, where he played a leadership role in helping define IBM's strategy for industrial standards and open architecture for service-oriented business solutions. He has filed more than 30 patent applications in the areas of e-commerce, Web services, rich media, data management, and information appliances, and he has published more than 80 technical papers in journals, book chapters, and conference proceedings.

Dr. Zhang is the Chair of the IEEE Computer Society's Technical Committee on services computing and the Editor-in-Chief of the *International Journal of Web Services Research*. He is the General Chair (2008 & 2006) and Program Chair (2007) of the IEEE International Conference on Web Services (ICWS) and the General Chair (2008 & 2006) and Program Chair (2007) of the IEEE Conference on Services Computing (SCC).



Patrick C. K. Hung received the Ph.D. degree in computer science from the Hong Kong University of Science and Technology (HKUST), Hong Kong. He was a Research Scientist with the Commonwealth Scientific and Industrial Research Organization (CSIRO), Canberra, Australia, and was a Visiting Assistant Professor with the Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong. He also has prior industrial experience in e-business projects in North America and Hong Kong. Since 2000, he has been serving as a panelist of the Small Business Innovation Research and Small Business Technology Transfer Programs of the National Science Foundation, USA. He is currently an Assistant Professor of business and information technology with the University of Ontario Institute of Technology, ON, Canada. He is currently collaborating with Boeing Phantom Works, Seattle, WA, and Bell Canada's Privacy Center of Excellence on security- and privacy-related research projects in industry. He is the holder of a U.S. Patent 05-0176 Invention Disclosure "Mobile Network Dynamic Workflow Exception Handling System" with Boeing Corporation, USA. He has published more than 80 research papers in different journals, conferences, workshops, and books.

Dr. Hung is the General Chair of the 10th IEEE International EDOC Conference (EDOC 2006) "The Conference on Enterprise Computing," and the Program Committee Vice-Chair of the IEEE International Conference on Services Computing (SCC 2007 & 2006). He is an Executive Committee Member of the IEEE Computer Society's Technical Steering Committee for services computing and an Associate Editor/Editorial Board member in several international journals such as the *International Journal of Web Services Research* and the *International Journal of Business Process Integration and Management*.