ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*, Vol.17, No.6, June 2006, pp.1470−1484
DOI: 10.1360/jos171470
© 2006 by *Journal of Software*. All rights reserved.

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

# PESOI: Process Embedded Service-Oriented Architecture*

Wei-Tek Tsai[+], Yinong Chen, Chun Fan

(Department of Computer Science and Engineering, Arizona State University, Tempe AZ, 85281, USA)

+ Corresponding author: Phn: +1-480-7276921, Fax: +1-480-9652751, E-mail: wtsai@asu.edu

**Tsai WT, Chen Y, Fan C. PESOI: Process embedded service-oriented architecture. *Journal of Software*, 2006, 17(6):1470−1484.** http://www.jos.org.cn/1000-9825/17/1470.htm

**Abstract**: Service-Oriented Architecture (SOA) has drawn significant attention recently, and numerous architecture approaches have been proposed to represent SOA-based applications. The architecture of SOA-based applications is different from traditional software architecture, which is mainly static. The architecture of an SOA-based application is dynamic, i.e., the application can be composed at runtime using existing services, and thus the architecture is really determined at runtime, instead of design time. SOA applications have provided a new direction for software architecture study, where the architecture can be dynamically changed at runtime to meet the new application requirements. This paper proposes a Process-Embedded Service-Oriented Infrastructure to build SOA-based applications. This infrastructure embeds the entire software lifecycle management and service-oriented system engineering into the application developed on this infrastructure. Thus, the users can easily re-develop the applications during operation to meet the changing environments and requirements, through the supports provided by the embedded infrastructure.

**Key words**: service-oriented computing; service-oriented architecture; software architecture; architecture classification

## 1 Introduction

Service-Oriented Architecture (SOA) has received significant attention recently as major computer and software companies such as HP, IBM, Intel, Microsoft, and SAP, have all embraced SOA, as well as government agencies such as DoD (US department of defense) and NASA. The initial focus on SOA has been the development of interoperability standards and protocols, such as WSDL, SOAP, UDDI, and recently numerous SOA standards have been proposed including ebSOA, ebXML, CPP, CPA, BPEL4WS, OWL-S, and RDF.

SOA is a new software development paradigm in which software developers are grouped into three parties in terms of their responsibilities: the application builders (service requesters), the service brokers (service publishers), and the service providers (service developers), which is referred as the standard SOA. Service providers develop services independent of potential applications by following the open protocols and standards. Service brokers publish the available services to the public such that the application builders can look up desired services and compose the target application using the services. Thus a target application is built through service discovery and service composing instead of traditional process of designing and coding software.

SOA has the following characteristics that are different from traditional software:

* Received 2006-02-15; Accepted 2006-03-14

**Standard-Based interoperability**: SOA emphasizes on stand-based interface, protocols, communication, coordination, workflow, discovery, collaboration, and publishing via standard protocols. These standards allow services developed in different computing platforms can interoperate with each other with the knowledge of service specifications only.

**Dynamic composition via discovery**: SOA provides a new way of application development by composing services just discovered. Furthermore, the composition and discovery can be carried out at runtime.

**Dynamic governance and orchestration**: Execution of services needs to be controlled and several mechanisms are available for execution control. One is service *governance* by policy. Specifically, policies can be specified, checked, and enforced during both SOA development time and runtime to ensure the system complies with the organization requirements. The other is *orchestration* where process execution will be coordinated by a central controller and it is responsible for scheduling the execution of services that may be distributed across a connectivity network such as ESB (enterprise service bus).

A complex system is usually represented in layers, in Ref.[1], a 5-layered architectural template is suggested for SOA applications, as illustrated in Fig.1: Presentation, Business process choreography, Services, Enterprise components, and Operational systems, with two supporting mechanisms: Integration architecture and Management & monitoring. The supporting mechanisms can be applied to each of the five layers.

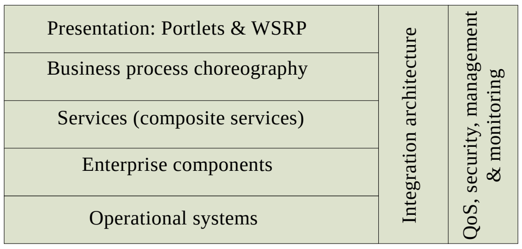

Presentation: Portlets & WSRP
Business process choreography
Services (composite services)
Enterprise components
Operational systems
Integration architecture
QoS, security, management & monitoring

Fig.1 Layers of SOA

Several studies have been conducted to investigate the architecture of SOA applications, e.g., IBM SOA Foundation architecture[1–4], Microsoft's .Net 2005 (Whitehorse)[5], SAP's NetWeaver[6], OASIS's FERA[7], enterprise SOA applications such as enterprise SOA[8] and Service-Oriented Enterprise (SOE)[9], and self healing architecture PKUAS[10].

Software architecture has been an active research areas in software engineering for the last 10 years, with numerous papers and books, such as Refs.[1,2,7–18]. Currently, several websites have been devoted to software architecture such as:

- Resources for Software Architects at http://www.bredemeyer.com/;
- Global Enterprise Architecture Organization (GEAO) http://www.geao.org;
- Software Engineering Institute at CMU software architecture at http://www.sei.cmu.edu/architecture/;
- Software architecture resource Sites at http://www2.umassd.edu/SECenter/SAResources.html;
- Handbook of Software Architecture at http://www.booch.com/architecture/index.jsp.

Most of the software architecture research projects focus on static architecture, where the emphasis is on:

- Architecture styles (layered architecture, object-oriented architecture, client-server architecture and so on);
- ADL (architecture description language) and see the list of available ADLS at the http://www.sei.cmu.edu/architecture/adl.html;
- Architecture evaluation and analysis such as SAAM;
- Documentation of existing architectures, practices, and patterns;
- Architecture views such as module structure, process structure, conceptual structure, physical structure, call structures, data flow structure, control flow, and class structure, execution architecture; and
- Formalization and specification of software architecture.

Recently, several studies have focused on dynamic software architecture, i.e., the software architecture that modifies its structure during execution[13]. The current research focuses on the formal specification techniques that can be used to reason and analyze dynamic architectures. A variety of reconfiguration rules such as graph rewriting rules, process algebra (such as CCS and CSP), predicate calculus, and architecture modification language (AML)[19] have been proposed to specify and analyze dynamic architectures[13]. However, these studies have not focused on the dynamic SOA yet. One significant difference between the existing dynamic architecture and SOA is that the dynamic architecture of SOA is fully integrated with many aspects of software development, such as service composition, code generation, and deployment.

## 2 Dynamic Architecture and Lifecycle Management

Due to the dynamic nature of SOA, the architecture of SOA-based systems has the following characteristics distinct from traditional software architectures: (1) Dynamic architecture via dynamic composition; (2) Lifecycle management embedded in the operation infrastructure architecture.

### 2.1 Dynamic architecture

Traditional software architecture is static. Once the software is developed using components and connections among the components, it cannot be changed. In SOA, the basic building block is not a component but a service. Services are loosely coupled and can be modeled as connected to a bus-like communication backbone, as described in ESB[20]. Connections among the services are controlled by a control center, which is also attached to the communication backbone, as illustrated in Fig.2.

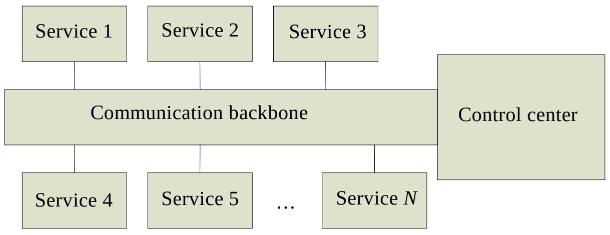


Fig.2 A generic SOA application architecture

The control center is a composition manager that specifies and controls the application composition configuration via a workflow specification, which defines how the services shall be connected together to deliver the desired results and how the messages are transferred among the services. With this kind of architectures, a user can compose applications with different architectures or even different architecture styles, such as a chain, a ring, a star, a layered structure, etc. This kind of architectures is in fact *virtual*. The connection restrictions can be enforced by policy and thus can be changed dynamically.

### 2.2 Lifecycle management

Another distinguishing characteristic of the SOA-based application architecture is that the lifecycle management can be embedded in the operation infrastructure to facilitate dynamic software composition. In this way, the SOA application *development* infrastructure and *operation* infrastructure can be merged into a single and unified SOA infrastructure. The *development* infrastructure may include: Modeling, function and policy specification, analysis, design, code generation, and verification and validation such as model checking and testing. The *operation* infrastructure may include: Code deployment, code execution, policy enforcement, monitoring, communication, and system reconfiguration.

Both IBM and Microsoft take this approach in their SOA designs. In IBM SOA Foundation Architecture[3,4], development activities (modeling and assembly) and operation activities (deployment and management) are integrated into a single process as illustrated in Fig.3.

The architecture consists of four phases: modeling, assembling, deployment, and management. Furthermore, runtime governance activities are performed to provide guidance and oversight for the target SOA application. The activities in the four phases are performed iteratively:

- Modeling: This phase models the user requirements in a system model with a set of services;
- Assembling: This phase composes applications using services that have been created or discovered at runtime according to the model specified in the previous phase;
- Deployment: In this phase, the runtime environment is configured to meet the application's requirements, and the application is loaded into that environment for execution;
- Management: After the application is deployed, the services used in the application are monitored. Information is collected to prevent, diagnose, isolate, and/or fix any problem that might occur during execution. These activities in management phase will provide the designer with better knowledge to manage the application.

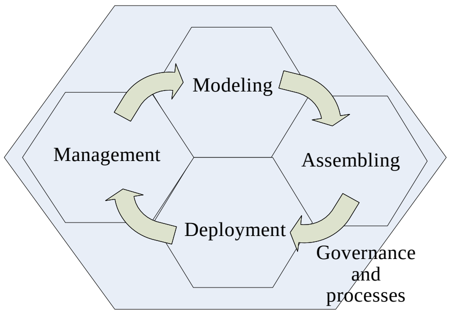


Fig.3 IBM SOA foundation architecture

The entire process will be controlled and orchestrated by the governance policies. IBM SOA foundation architecture is based on such a model-driven application development process. This looping back process along with the governance and other processes can be delivered together with the target SOA application to the user. When there is a need of changing the application architecture, the user can re-specify the system model, and the application can be re-assembled and re-deployed.

### 2.3 IBM WebSphere integration reference architecture

IBM WebSphere Integration Reference Architecture[2,21] is a SOA application model, which provides a set of services to enable business integration in a large diverse enterprise environment. Figure 4 shows the key integration functions needed to provide comprehensive and enterprise–level solutions. The core of the WebSphere Integration Reference Architecture is the connectivity services, which provides the infrastructure to support and to instantiate the Enterprise Service Bus (ESB)[20].

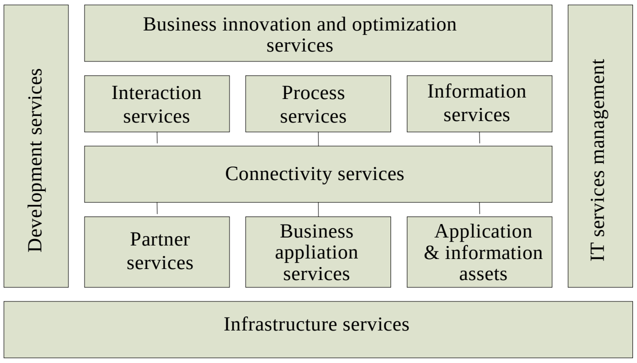


Fig.4 IBM WebSphere integration reference architecture

### 2.4 Microsoft service-oriented application development framework

Microsoft also takes this approach in Visual Studio .Net 2005 (Whitehorse)[5], which has the capabilities for modeling, architecture, code generation, code deployment, and code execution. In .Net 2005, an SOA application designer can select services from a service pool and define the architecture to specify how services can communicate with each other. After completing the architecture block diagram, the designer can execute it to see if the application can work as desired without knowing the details of the services used. The application can be easily changed by changing the architecture model. Figure 5 illustrates the architecture block diagram of an SOA-based application developed in .Net 2005.

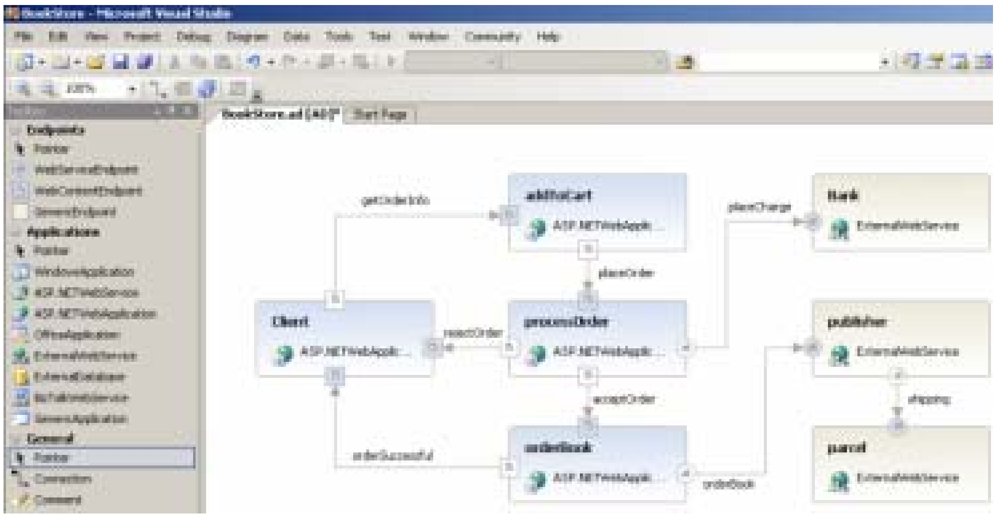


Fig.5 Microsoft whitehorse SOA designer

## 3 PESOI: Process-Embedded Service-Oriented Infrastructure

PESOI (process-embedded service-oriented infrastructure) is a framework developed at Arizona State University, which integrates the development infrastructure into the operation infrastructure, so that the application can be re-developed during its operation. In addition to the capacities that WebSphere and .Net 2005 have, PESOI also includes explicit Verification and Validation (V&V) capacities, a reconfiguration process, and a model driven approach. This section presents the overall design of PESOI, lifecycle, and the dynamic architecture of the applications developed on PESOI.

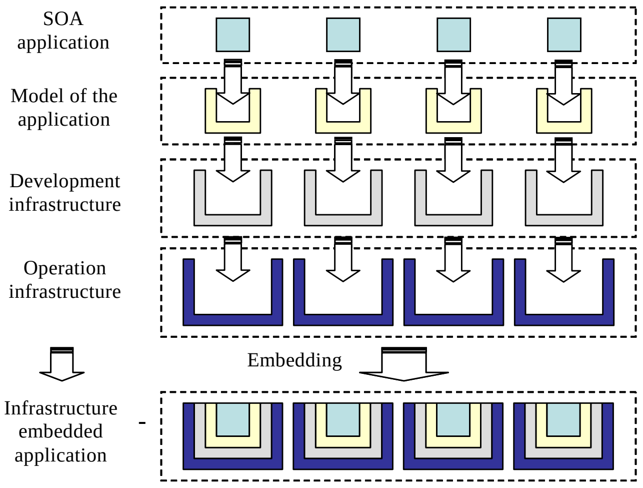


Fig.6 Infrastructure-Embedded SOA application

Figure 6 shows the layers of a related SOA application. First, the application is decomposed into the model consisting of patterns of sub-models. Each of the sub-models is developed using a tool in the development infrastructure, which in turn, maps the sub model into an executable code in the operation infrastructure.

Figure 7 outlines the lifecycle (development and operation phases) of applications developed on PESOI, which consists of the following phases: Modeling and specification, Verification of specification, Code generation for validation, Validation (simulation and testing), Assembling and deployment, Operation and monitoring, Evaluation at runtime, and Reconfiguration.

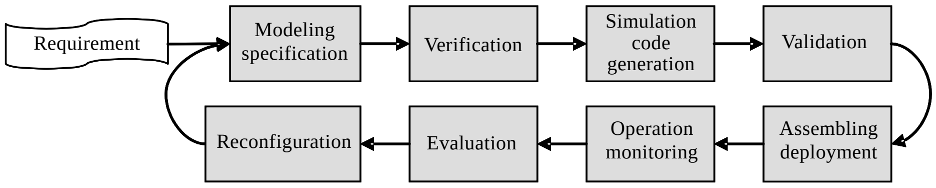


Fig.7 PESOI reference architecture

These phases form a feedback loop, where the last phase reconfiguration is connected to the first phase modeling and specification. Each of the phases in PESOI is elaborated as follows:

- Modeling & Specification phase: This phase consists of two sub-phases. The modeling sub-phase is at architecture or component level, similar to Microsoft architecture model shown in Figure 6 and IBM SCA/SDO model. The specification sub-phase uses PSML-S process specification and modeling language, which is similar to BPEL4WS, to define the flow and conditions between the components;
- Verification phase: It contains activities such as Completeness & Consistency (C&C) analysis, model checking, test case generation based on the functional and policy specification;
- Assembling phase: It consists of automated code generation from the process specification and binding of external services to obtain the executable code;
- Validation phase: It runs the executable in a simulation environment and validates the code via testing;
- Deployment phase: This phase deploys executable code into the execution environment. It involves binding the addresses of real devices into the code. Monitors and data collectors will be embedded into the code;
- Execution and monitoring phase: The executable code is executed in the real environment. The monitors and data collectors will collect and record data for later analysis;
- Evaluation phase: It analyzes the data, evaluates reliability and performance, and provides input for the reconfiguration phase;
- Reconfiguration phase: It takes reliability evaluation results as input and decides if a rebinding is necessary to replace a less reliable service through rebinding a reliable services. A service farm contains the backup spares need for rebinding. The services in the farm are tested and ranked in an independent process such as group testing[22]. If the application requirement is revised by the user, a re-composition or re-architecture needs to be performed. In this case, the composition or architecture of the application will be modified. The modified application needs to go through the entire development process, and thus, the development infrastructure needs to be embedded into the application to enable the reconfiguration.

The rest of the paper will talk about issues related to PESOI. Specifically, Section 4 will discuss the modeling issues in PESOI, Section 5 will discuss the dynamic reconfiguration issues, and Section 6 will talk about the verification and validation issues in PESOI.

## 4 Single Model Multiple Analyses (SMMA)

Numerous modeling projects have been proposed to model software, ranging from informal descriptions to semi-formal models, such as a state model, to formal models, such as process algebra CCS, CSP and ACP[1,14,16–18]. For SOA applications, informal methods are not applicable because one can not produce code from informal descriptions automatically. However, strict formal methods will need more time to mature before they can be applied to SOA applications. Most current approaches use lightweight formal techniques that can produce software design and code automatically, and can be easily used by practitioners due to the visualization capabilities of these models.

Currently, the UML from OMG [http://www.omg.org/technology/documents/formal/uml.htm] is the de factor industry standards for specifying and analysis of software models, particularly for object-oriented programs. The question is whether UML is suitable for SOA. Particularly, is UML suitable for SOA applications with on-demand dynamic composition?

UML 2.0 has 13 different types of diagrams, divided into three categories:

- Structure models: Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram, and Deployment Diagram;
- Behavior models: Use Case Diagram, Activity Diagram, and State Machine Diagram;
- Interaction models: Sequence Diagram, Communication Diagram, Timing Diagram, and Interaction Overview Diagram.

The UML has more than 20 models. While each individual model within UML is useful for modeling and analyses for certain aspects of the system engineering, the fact that it has so many models is a concern. One problem is that it is not possible to convert one model (for example Activity Diagram) to another model (such as State Machine Diagram) automatically. Thus, a change in a model may need the involved engineers to manually update the rest of models to keep these models consistent with each other. This does not bode well with on-demand dynamic composition in SOA where code needs to be generated from the model, and possibly both the model and the code need to be analyzed, verified, and validated dynamically at runtime using existing services.

We call the approach taken by UML the MMMA (multiple models multiple analyses) approach because it has multiple models and each model has its own analysis techniques. In a typical MMMA approach, models are interconnected with each other, but not fully integrated to each other, i.e., it is not possible to generate the rest of models from one model. Thus, MMMA is error-prone during system updating, and thus it is not suitable for the SOA environment, where models can be dynamically changed at runtime as applications are re-composed or reconfigured.

On the other hand, if the development is based on a single model, all other models and analyses are automatically derived from this model, the approach is called SMMA (single model multiple analyses) approach. One of the characteristics of SMMA is that a designer can focus his/her attention on the single model only. If a change is made, he/she can just update the primary model, and the rest of models and analyses can be automatically generated. Specifically, once the primary model is updated, code can be automatically re-generated, completeness and consistency analysis can be re-checked, simulation code can be automatically generated and simulation performed, state model can be automatically updated based on the new model and so on. In this way, significant system engineering tasks can be automatically performed. SSMA approach does not conflict with the principle of “separation of concerns”. Instead, it makes sure that components created by other parties can interoperate seamlessly. A demo of the SMMA approach in a network of service-oriented embedded robots can be viewed at the website at http://asusrl.eas.asu.edu/EmbeddedExplorer/.

PESOI is the only infrastructure that has taken the SMMA approach, as shown in Fig.8, where PSML (process specification and modeling language) is the single model and all the analyses are directly or indirectly based on this model. PSML-S (PSML for Services) and PSML-P (PSML for Policies) are derived from PSML, which are used to model the functionality and policies in the application, respectively[23]. Table 1 lists the operations and analyses that can be automatically performed on a PSML-S and PSML-P models.

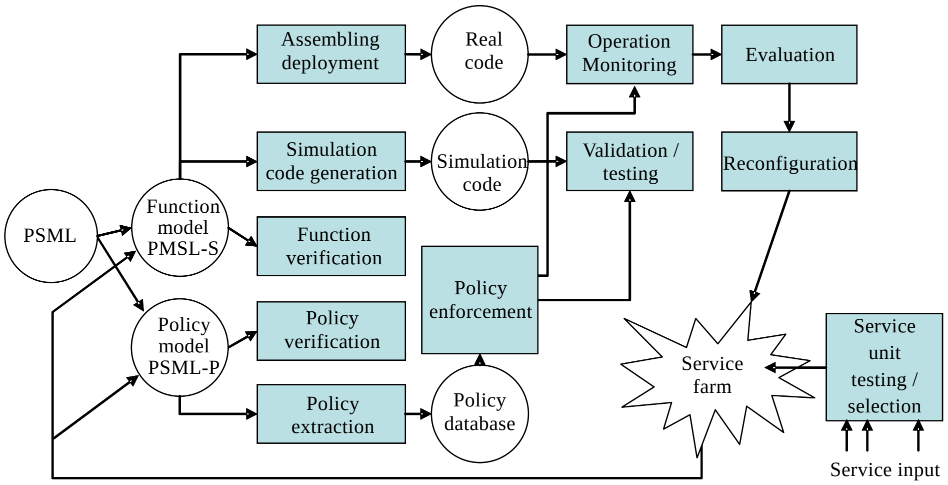


Fig.8 PESOI's approach of single model multiple analyses

**Table 1** Operations and analyses on PSML-S and PSML-P models

| Analyses | Model and tool | Comments |
|---|---|---|
| Automated code generation | C# code is generated from PSML-S. | Code is generated by following the process flow of the PSML-S. |
| Model checking | Both PSML-S and PSML-P models, as well as the generated code, can be model checked by BLAST [25] and BLADE[26]. | As process modeling in PSML and generated code are control-flow based, they can be checked by a model checkers based on control flow models. |
| Completeness and consistency (C&C) checking | Both PSML-S and PSML-P models, as well as the generated code, can be checked for C&C at runtime[30]. | The C&C checking is based on an abstracted model, and the abstracted model can be obtained either from the PSML-S and PSML-P, or generated code. |
| Policy specification, analysis, and enforcement | Many aspects of the system can be checked only at runtime. Policy is interesting because it can be specified using the same language PSML-P[27]. | The issue is to ensure that constraints specified by policies can be enforced at runtime. |
| Dynamic test case generation | Test input can be generated by analyzing the PSML-S and PSML-P models. | Various testing techniques can be used including partition testing, random testing, Swiss cheese testing techniques[30]. |
| Distributed service-oriented simulation | The code generated can be executed as the simulation of the model specified[24]. | The difficult part is the simulation infrastructure to coordinate the execution run of concurrent processes. |
| Reliability modeling and assessment | The reliability of application as well as that of participating services can be estimated using the data collected. | The architecture-based reliability models can be used and they can be validated using the DREP model[29]. |

## 5 PESOI Support for Dynamic Reconfigurations

Generally, three levels of reconfigurations are available in SOA, once an application is developed and deployed, as illustrated in Fig.9:

- Rebinding: A service in the application is replaced by another service that has the same functionality. The change required is to change the binding address related to the service. Rebinding is mainly used for fault-tolerant computing (replacing a failed service or an unavailable) or performance upgrade;
- Re-composition: In addition to rebinding, a new service can be added to or an existing service can be removed from the application. However, the architecture style remains unchanged. Re-composition is mainly used for functionality and performance enhancement;
- Re-Architecture: In addition to re-composition, the architecture style can be changed, for example, changing the bus type of connection to the layered type of architecture. The former allows any two

components to communicate directly, while the latter only allows the communication between components in the two adjacent layers. Re-architecture allows the human designer to generate a new application at runtime time.

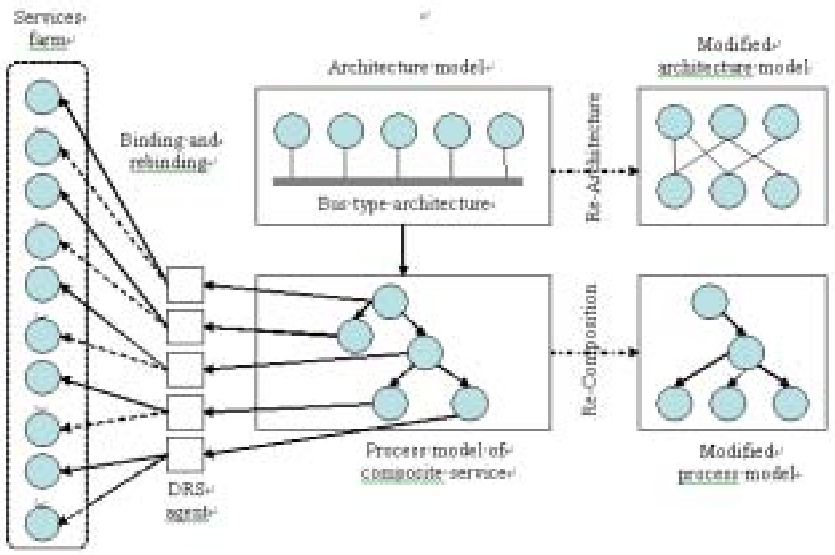


Fig.9 Illustration of rebinding, re-composition, re-architecture, and service farm

SOA application development starts with architecture model, in which the application is modeled by a set of components with certain type connectivity, e.g., bus, peer to peer, and layered type, which are called architecture patterns. The architecture model will be elaborated manually, but with the help of architectural pattern, into the process model, in which the flows between components and selections among flows are defined. Each component in the process model is linked to a service that can perform the required functions. To tolerate service faults, multiple services with the same functionality can be bound to a component through a Dynamic Reconfiguration Service (DRS) agent. Rebinding means to substitute the address of a new service for that of the service to be replaced. Re-composition means changing the process model or the flow between the components, without changing the architecture. Re-architecture means changing the type of connectivity among the components. To enable seamless runtime reconfigurations, checking points are used periodically to save the status of the target system so that, when a reconfiguration is necessary, the reconfigured system can continue its execution from the last checking point.

## 5.1 PESOI supports for rebinding

Rebinding is the basic and most frequently used reconfiguration supports provided by PESOI. The supports provided to rebinding, which also apply to re-composition and re-architecture, include:

**Service farm and service cache**: All the evaluated services that can be used to replace one of the services linked to the application will be placed in a service farm as backup services. The backup services can be evaluated according to reliability, performance, user feedback, and profiles. The services in the farm with the highest ranking will be placed in a service cache, which is used like the cache in CPU for fast rebinding. Figure 10 shows how the service farm and service cache are established and associated to the application. First, the PESOI looks up the service broker to find all services that can meet the requirement of each service in the application composition. The service testing manager will then test each service. If the number of services found is large, group testing can be applied[22]. The qualified services are put into the service farm as the backup spares and the best qualified services will be placed in the service cache.

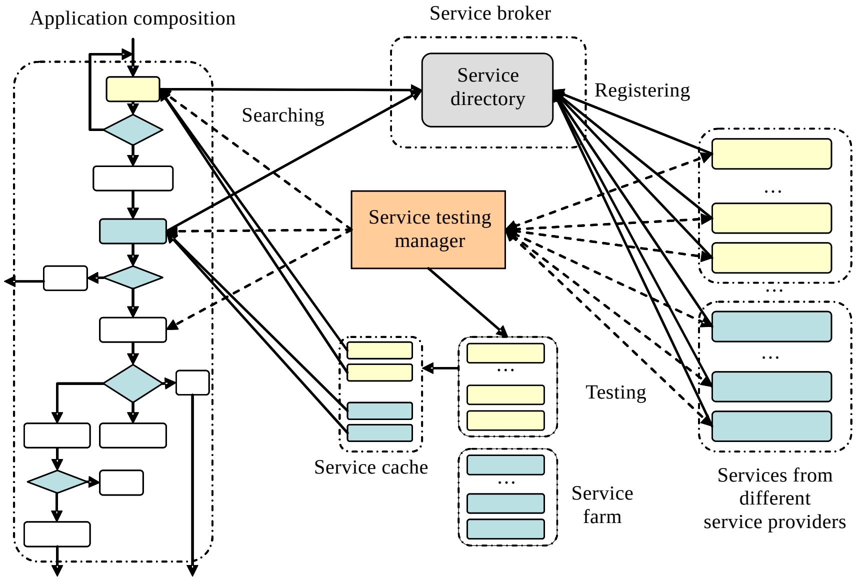


Fig.10 Establishment of the service farm

**Dynamic reconfiguration service** (**DRS**)[28]: DRS is a framework with distributed agents which will monitor, assess, and reconfigure the SOA application at runtime. The framework defines an agent-based, collaborative and self-reconfigurable architecture for service-oriented distributed computing, supported by a set of DRS services and agents, including registration service, monitoring service, and dynamic reconfiguration agents (DRS agent). DRS agents are embedded into the application, which coordinate with each other to make the application self-reconfigurable. In this example, two applications use the same encryption service. If one application wants to upgrade its encryption method, it cannot simply rebinds to a different service. It has to coordinate with its communication peers, so that they can upgrade at the same time and thus can decrypt each other's data.

## 5.2 PESOI supports for re-composition

Re-Composition is a much harder problem than rebinding because not only services may be replaced, the SOA application is actually changed. In additional to the supports listed in Section 5.1, PESOI supports the SMMA for the re-composition, i.e., the SOA application specification can be updated by changing the model only. Once the model is changed, it will be automatically re-verified. Then, the code can be automatically re-generated, and the generated code can be re-validated and re-evaluated. During the code re-generation, new services may be searched, discovered, and bound into the new application.

## 5.3 PESOI supports for re-architecture

In additional to the supports listed in Sections 5.1 and 5.2, PESOI uses architecture patterns to support re-architecture. The patterns are restrictions defined in policies. The modification to the policies leads to the architecture change and architecture patterns help the fast re-generation of the process specification in PSML-S and PSML-P.

Because of the re-architecture, applications developed using PESOI no long have fixed architecture, such as bus, layered, or point-to-point. The architecture can be changed after the deployment of the application.

## 5.4 Service access and service cache

Current SOA application supports two kinds service accesses: remote invocation and code importation with local access. The former technique binds the URL of a service into the application and each access to the service is a remote invocation. The latter technique, called service cache, imports the executables of the services into the same computer that runs the application or different nodes in a local distributed system or network.

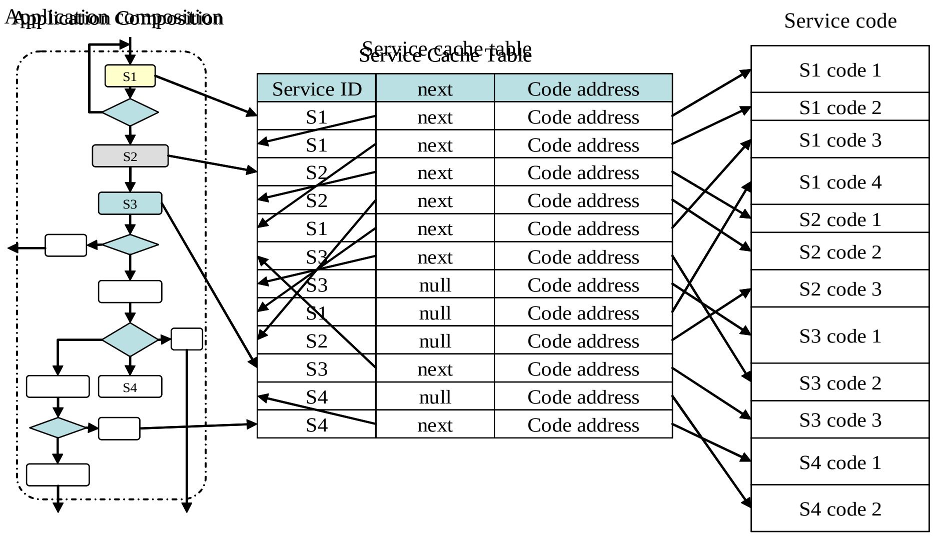
Imported services eliminate the internet access and are much faster and much more reliable. However, it uses local resources, which are often limited, and rebinding will take much longer because the entire service code needs to be loaded and deployed into a local machine. Assume that an application needs (composed of) *N* external services and has *M* slots (processors and memory) to import (host) external services. PESOI implements the service farm and service cache as follows.

First, the service test manager will find services that meet the requirements of the application and put them all into the service farm. The service farm only needs to store the service IDs and URLs of the services. The number of services that can be listed in the service farm is practically unlimited. However, the service cache needs to store the entire executable code of the services and thus is limited and the selection of services needs to be optimized. Table 2 lists the relationship between N and M and the PESOI placement and replacement policies in each case.

**Table 2** Relationship between *N* and *M* and the PESOI placement and replacement policies

| $N$–$M$ relation | No. of services using local accesses | No. of services using remote accesses | No. of backup services using local accesses per service | Replacement and replacement policies in the service cache |
|---|---|---|---|---|
| $M$=0 | 0 | $N$ | 0 | All services use remote invocation |
| $N$≥$M$ | $M$ | $N$–$M$ | 0 | *M* out of *N* services use local accesses. Most frequently used services will be imported |
| $N$<$M$ | $N$ | 0 | ($M$–$N$)/$N$ | All services use local access. The most frequently used services will have more backups |

As shown in Fig.11, the cache table contains three entries: service ID, which identifies the service corresponding to this service code; initial address where the service code is located; and the table entry number of the next backup service. That is, for each service ID, a linked list of service code is provided to each service.



| Service ID | next | Code address |
|---|---|---|
| S1 | next | Code address |
| S1 | next | Code address |
| S2 | next | Code address |
| S2 | next | Code address |
| S1 | next | Code address |
| S3 | next | Code address |
| S3 | null | Code address |
| S1 | null | Code address |
| S2 | null | Code address |
| S3 | next | Code address |
| S4 | null | Code address |
| S4 | next | Code address |

Fig.11 Service cache table and imported service code in local memory

In the example showing the figure:

- $N$=4: Four external services are needed by the application;
- $M$=12: the service cache can hold 12 service code;
- Four linked lists services are formed, each for a service:
  - o Service $S1\rightarrow S11\rightarrow S12\rightarrow S13\rightarrow S14\rightarrow null$
  - o Service $S2\rightarrow S21\rightarrow S22\rightarrow S23\rightarrow null$
  - o Service $S3\rightarrow S33\rightarrow S32\rightarrow S31\rightarrow null$
  - o Service $S4\rightarrow S42\rightarrow S41\rightarrow null$

PESOI uses different length of backup services, instead of equal length, to reflect the fact that some services may have less reliable link or services and need more frequent rebinding than the others.

## 6 A Case Study—Pet Shop

An application called Pet Shop is built for illustrating the development and operation phases in PESOI. An integrated modeling environment for constructing the PSML-S model is provided, which allows the application builders to specify the detailed system model information using PSML-S language, the application composition logic, and the application configuration policies based on the users requirements. Figure 12 illustrates the PESOI editor's window, which support convenient writing of the PSML-S and PSML-P specifications. For the pet shop application, the PSML-S specification consists of 12 Actor elements, 16 Action elements, 3 Condition elements, 2 Data elements, 2 Event elements, 4 processes, and one default application composition configuration. A part of the specification script is shown in the editor window in Fig.12. A PSML-S model can consist of the following model elements: Actors, Actions, Attributes, Conditions, Data, Events and process that defines the flows among the model elements[23].

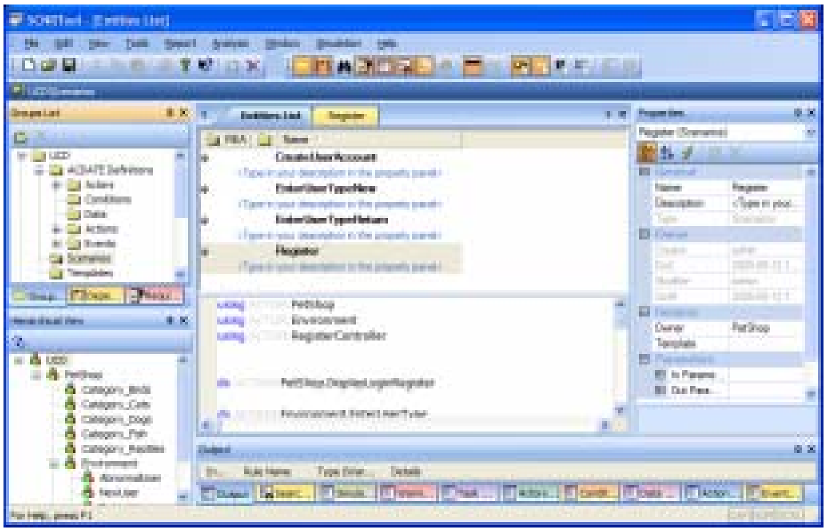


Fig.12 PSML-S editor

Following the SMMA, we have performed the C&C analyses, model checking, test case generation, simulation code generation, simulation, and testing.

Among these operations, the simulation validates the execution logic of the PSML-S model to check if it always generates an acceptable outcome for each test case. Two user's interfaces are implemented to observe the simulation traces. A graphical simulation interface highlights the current point of execution in the flowchart of the

PSML-S specification. Figure 13 shows a snapshot of the execution. A text simulation interface records every trace that has been tested and the output corresponding to each input. A snapshot of the interface is given in Fig.14.

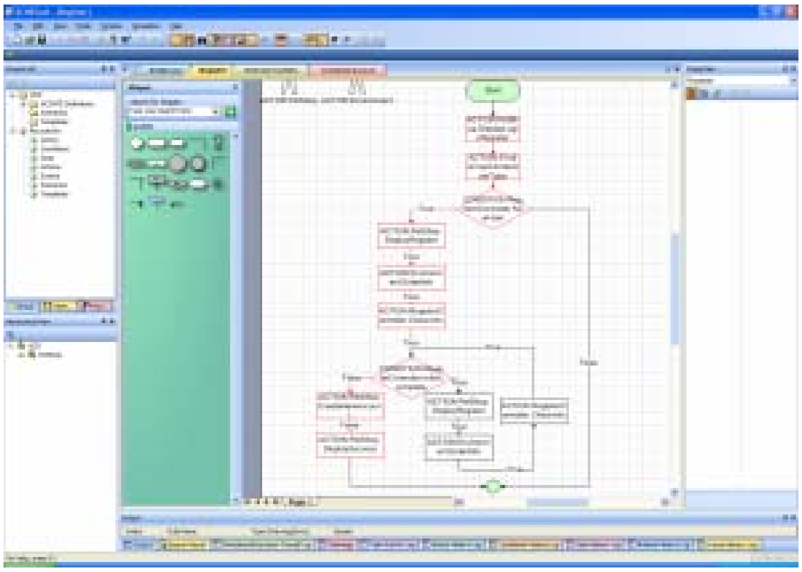
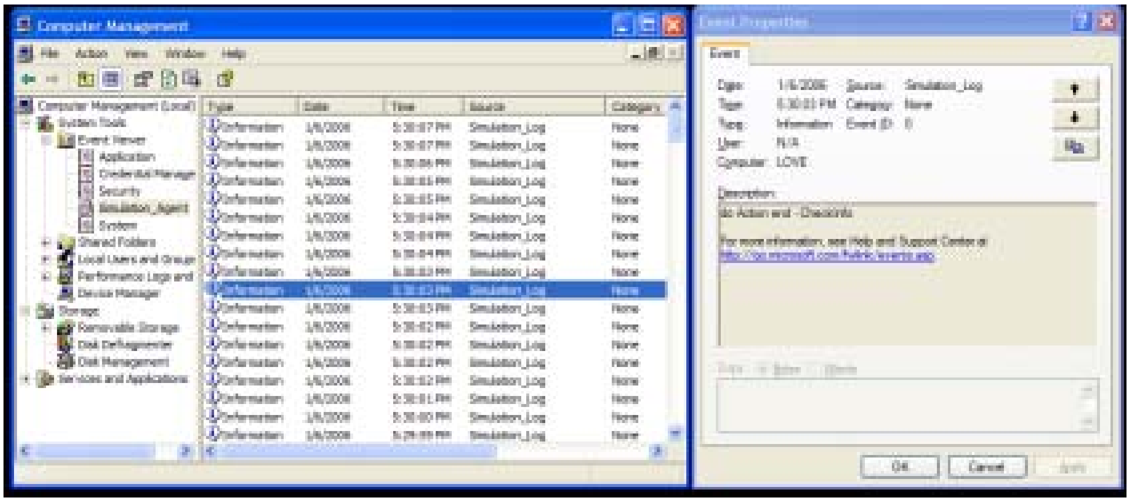


Fig.13 Graphical simulation trace animation


Computer Management

| Type | Date | Time | Source | Category |
|---|---|---|---|---|
| Information | 1/6/2006 | 5:30:07 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:07 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:06 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:05 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:05 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:04 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:04 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:04 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:03 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:03 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:03 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:02 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:02 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:02 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:02 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:01 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:30:00 PM | Simulation_Log | None |
| Information | 1/6/2006 | 5:29:59 PM | Simulation_Log | None |

Event Properties

Date: 1/6/2006 Source: Simulation_Log
Time: 5:30:03 PM Category: None
Type: Information Event ID: 0
User: N/A
Computer: LOVE

Description:
[illegible] Action end - CheckInfo

For more information, see Help and Support Center at
http://go.microsoft.com/fwlink/events.asp

Fig.14 Text simulation trace

Using the graphical interface, users can visualize the simulation execution step by step from the control flow chart of the PSML-S model. Thus users can closely observe the behavior of the PSML-S model. This feature is extremely useful when users want to perform debugging on the PSML-S model.

The text simulation interface records the detailed system event logs. Multiple runs of simulations with different configurations can be performed. The simulation results stored in the event log can be analyzed later for different purposes such as detecting reliability evaluation.

## 7 Conclusion

This paper presents the PESOI infrastructure for SOA application development, which has the following unique features:

- PESOI takes an SMMA approach, that is, all the analyses and evaluations are based on the PSML-S model. When the application needs to be modified, only the PSML-S model needs to be modified and the rest of the system is automatically modified accordingly;
- PESOI supports three levels of reconfigurations, rebinding to external services, re-composition of the application logic, and re-architecture of the application. Different level of reconfiguration requires different level of V&V support;
- Dynamic reconfiguration needs dynamic V&V to ensure the dependability of the dynamically created applications. PESOI enforces V&V in every phase of the development and operation.

PESOI is proposed based on the experience in several experimental and industrial projects that we have implemented. Different mechanisms have been implemented in different projects but we have not implemented all the V&V and evaluation mechanisms in a single project. We are currently implementing a prototype that put all these components together in order to obtain the complete and coherent lifecycle data.

**References**:

[1] Arsanjani A. Service-Oriented modeling and architecture: How to identify, specify, and realize services for your SOA. Whitepaper from IBM, Nov 2004. http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/

[2] High RJr, Kinder S, Graham S. IBM SOA foundation: An architectural introduction and overview, Version 1.0. 2005.

[3] IBM Developers Works. New to SOA and Web services. http://www-128.ibm.com/developerworks/webservices/newto/

[4] Simmons S. Introducing the WebSphere integration reference architecture: A service-based foundation for enterprise-level business integration. IBM WebSphere Developer Technical Journal, 2005. http://www-128.ibm.com/developerworks/websphere/techjournal/0508_simmons/0508_simmons.html

[5] Randell BA, Lhotka R. Bridge the gap between development and operations with whitehorse. MSDN magazine, 2005. http://msdn.microsoft.com/msdnmag/issues/04/07/whitehorse/default.aspx

[6] SAP NetWeaver product introduction. http:// www.sap.com/solutions/netweaver/index.epx

[7] Brown G, Carpenter R. Successful application of service-oriented architecture across the enterprise and beyond. Int'l Technology Journal, 2004. http://www.intel.com/technology/itj/2004/volume08issue04/art09_successful/ p07_references.htm

[8] Krafzig D, Banke K, Slama D. Enterprise SOA: Service-oriented Architecture Best Practices. New York: Prentice Hall, PTR, 2005.

[9] Erl T. Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services. New York: Prentice Hall, PTR, 2004.

[10] Mei H, Huang G, Tsai WT. Towards self-healing systems via dependable architecture and reflective middleware. In: Proc. of the 10th IEEE Int'l Workshop on Object-Oriented Real-Time Dependable Systems. Sedona, 2005. 337−344.

[11] Allen R, Douence R, Garlan D. Specifying and analyzing dynamic software architectures. In: Proc. of the '98 Conf. on Fundamental Approaches to Software Engineering (FASE'98). Lisbon, 1998.

[12] Dobrica L, Niemelä E. A survey on software architecture analysis methods. IEEE Trans. on Software Engineering Archive, 2002, 28(7):638-653.

[13] Bradbury JS, Cordy JR, Dingel J, Wermelinger M. A survey of self-management in dynamic architecture specifications. In: Proc. of the Int'l Workshop on Self-Management Systems. 2004.

[14] Garlan D, Allen R. Formalizing architectural connection. In: Proc. of the 16th Int'l Conf. on Software Engineering. 1994. 71−80.

[15] Garlan D, Monroe RT, Wile D. Acme: Architectural description of component-based systems. In: Leavens GT, Sitaraman M, eds. Foundations of Component-Based Systems. Cambridge University Press, 2000. 47−68.

[16] Inverardi P, Wolf A. Formal specification and analysis of software architectures using the chemical, abstract machine model. IEEE Trans. on Software Engineering, Special Issue on Software Architecture, 1995,21(4):373−386.

[17] Shaw M, Garlan D. Formulations and formalisms in software architecture. Computer Science Today: Recent Trends and Developments, LNCS 1000, Springer-Verlag, 1995.

[18] Shaw M, DeLine R, Klein DV, Ross TL, Young DM, Zelesnik G. Abstractions for software architecture and tools to support them. IEEE Trans. on Software Engineering, 1995. 314−335.

[19] Dowling J, Cahill V. Dynamic software evolution and the k-component model. In: Proc. of the Workshop on Software Evolution, OOPSLA. 2001.

[20] Chappell D. Enterprise Service Bus. New York: O' Reilly Media, 2004.

[21] IBM eServer and Automatic Computing. http://www-03.ibm.com/servers/autonomic/

[22] Tsai WT, Bai X, Chen Y, Zhou X. Web service group testing with windowing mechanisms. IEEE Int'l Workshop on Service-Oriented System Engineering (SOSE). Beijing, 2005. 213−218.

[23] Tsai WT, Paul RA, Xiao B, Cao Z, Chen Y. PSML-S: A process specification and modeling language for service oriented computing. In: Proc. of the 9th IASTED Int'l Conf. on Software Engineering and Applications (SEA). Phoenix, 2005. 160−167.

[24] Tsai WT, Fan C, Chen Y, Paul RA. DDSOS, distributed service-oriented simulation. In: Proc. of the 39th Annual Simulation Symp. (ANSS). Huntsville, 2006. 160−167.

[25] Henzinger TA, Jhala R, Majumdar R, Sutre G. Lazy abstraction. In: Proc. of the 29th Annual Symp. on Principles of Programming Languages. 2002. 58−70.

[26] Huang H, Tsai WT, Paul RA, Chen Y. Automated model checking and testing for composite Web services. In: Proc. of the 8th IEEE Int'l Symp. on Object-Oriented Real-Time Distributed Computing (ISORC). Seattle, 2005. 300−307.

[27] Tsai WT, Chen Y, Paul RA. Dynamic simulation verification and validation by policy enforcement. In: Proc. of the 38th Annual Simulation Symp. 2005. 2005. 91−98.

[28] Tsai WT, Song W, Paul RA, Cao Z, Huang H. Services-Oriented dynamic reconfiguration framework for dependable distributed computing. In: Proc. of the IEEE COMPSAC 2004. 2004. 554−559.

[29] Tsai WT. Service-Oriented system engineering: A new paradigm. In: Proc. of the IEEE Int'l Workshop on Service-Oriented System Engineering (SOSE). Beijing, 2005. 3−8.

[30] Tsai WT, Wei X, Chen Y, Xiao B, Paul RA, Huang H. Developing and assuring trustworthy Web services. In: Proc. of the 7th Int'l Symp. on Autonomous Decentralized Systems. Chengdu, 2005. 43−50.

**TSAI Wei-Tek** is a professor at Arizona State University. His research areas are service-oriented computing, embedded systems, and system verification and validation.

**FAN Chun** received his Ph.D. degree at Arizona State University in 2006 and is now with Motorola, USA. His research areas are service-oriented simulation and software development life cycle.

**CHEN Yinong** is a senior research scientist at Arizona State University. His research areas are service-oriented computing, embedded systems and dependable computing.