

# PSML-S: A PROCESS SPECIFICATION AND MODELING LANGUAGE FOR SERVICE-ORIENTED COMPUTING

W.T. Tsai, Ray A. Paul\*, Bingnan Xiao, Zhibin Cao, Yinong Chen  
Department of Computer Science and Engineering, Arizona State University  
PO Box 8809, Tempe, AZ 85287-8809, U.S.A.  
\*Department of Defense, U.S.A.  
wtsai@asu.edu

## ABSTRACT

This paper presents a process specification and modeling language for service-oriented system development. The language, PSML-S, is defined by a metamodel consisting of four model packages: core model, behavior model, structure model, and constraint model. The core model defines the data types and operations in other models; the structure model defines the constructs to specify the static structure of the system under modeling; the behavior model defines the constructs for describing the dynamic behaviors; and the constraint model defines constructs for specifying the non-functional constraints in the structure and behavior models. As an illustrative example, the PSML-S is used to specify a virtual bookstore and the related systems.

PSML-S is part of a development environment for service-oriented systems. PSML-S and the development environment have been used in several industrial and governmental projects.

## KEY WORDS

Modeling language, Web services, behavior, and constraint

## 1. Introduction

Service-Oriented Computing (SOC) including service-oriented architecture (SOA) and Web Services (WS) receives significant attention recently [6][8][11][12]. Most major corporations and government agencies are pushing this technology [3][4][5][10]. The idea that a software program as a service that can be discovered, matched, composed, executed, verified, and monitored in real time and at runtime provides a new paradigm of computing.

SOC will have a great impact on software engineering and software business as companies focus on their core competency and leveraging their existing services. Organizations will have different kinds of dependency from that before, and started to have different collaboration models among themselves, via services. Instead of buying and selling software, the organizations may now think in terms of providing services and using

existing services from the Internet. The evolutionary and technological shift from products to services means that the value is moving from the technology itself to how the technology is being applied [10]. The value (in terms of increased productivity, total cost of ownership, improved efficiency and effectiveness, return on investment formula or benchmark, project completion time, and increased revenue) must be explicit and reflected from the technology's investment.

SOC has the following impacts on the software engineering:

- Knowledge of existing services is as important as ability to build a new one;
- System interaction via loosely coupled service interaction;
- The critical role of SOC architecture;
- Quality assurance issues;

SOC is much more than Object-Oriented Computing (OOC). The system structure and its implications to system composition, reliability, verification and validation, security, reconfiguration capabilities are different from OOC [10][15]. Instead of static composition (with dynamic objects and dynamic binding) in OOC, SOC need to support dynamic composition in real time and at runtime with knowledge of the service interface only. UML is a popular modeling language for OOC [8]. As a universal and common language, UML provides excellent behavior diagrams, interaction diagrams, and structure diagrams for modeling and provides a common vocabulary between different stakeholders such as PMs, systems engineers, QAs, and system architects. UML is initially designed to facilitate the evolution from procedure programming to OOC paradigm. When applies UML to the SOC, the interconnected models in UML has weak support for dynamic architecture with run-time composition, re-composition, and reconfiguration. We need a fully integrated model to address these dynamic issues in Service-Oriented Architecture.

The emerging SOC requires a new set of process and service modeling language. Process Specification & Modeling Language for Services (PSML-S) provides a new service modeling language and process to adopt the new specifications in SOC.

## 2. PSML-S Design Considerations

The PSML-S is a language for specification and modeling of services in system engineering, software engineering, and service engineering. It facilitates the specification, analysis, composition, recomposition, simulation, execution, and testing of the system based on the system specification.

### 2.1 Motivation

Developing a model for an industrial-strength software system prior to its construction or renovation is as essential as designing a blueprint for a large building. Good models are essential for communication among project teams and to assure architectural soundness. We build models of complex systems because we cannot comprehend such system in its entirety. As the complexity of systems increases, so does the importance of good modeling techniques.

In the past, when we build a complex system, we do not know what the system will look like until it is fully developed. With the system model specified, we can evaluate the system architecture and simulate system behaviors in the early phase of software engineering life cycle. Cost needed for changing the model at the early stage of software life cycle will be much lower than changing the system after it has been constructed.

The advance in compilation and code generation technologies makes it possible to directly translate the specification into executable, which greatly simplify the programming tasks. In SOC, system developers are divided into three parties: service providers, service directories, and service requesters (application builders). The tasks of application builders are using a high-level specification language such as WSFL (Web Service Flow Language from) [5], and WS-CDL (Web Services Choreography Description Language) [4], BPEL4WS (Business Process Execution Language for Web Services) [7], to specify the application logic using the available web services. PSML-S is such a high-level specification language. The features of this language are further described in the following sub sections.

### 2.2 Rapid Development and Runtime Change

PSML-S is designed to support rapid development of services in the SOC. Once the PSML-S model is constructed for a system, static analyses and dynamic analyses can be performed on the system specification. The analyses results can be used by the system designers

to evaluate and verify their design before the system is built [2][13][14][15].

Besides the rapid evaluation and verification capability, PSML-S also supports automated code generation from the specified applications based on the available component services, simulation execution, and validation of the code by testing [13]. In other words, the application is specified using the services found through service directories.

Since changes are inevitable during software development, it is essential for a model to be able to apply changes rapidly. PSML-S provides various analysis and simulation capabilities assisting designers to evaluate and validate the system design. When changes are needed, the designers need only to change the specification and the changes may be applied to the system with the automation (regeneration of the simulation code). Furthermore, the on-the-fly changes may also be applied at the system runtime.

### 2.3 Design Goals of PSML-S

The goals of PSML-S are to provide a modeling methodology and a specification language that are:

- Self-contained and coherent: The PSML-S model for a system is comprehensive. Multiple static analyses (Completeness & Consistency Analysis, Dependency & Ripple Effect Analysis, Event Sequence / Event Tree Analysis, and model checking) can be performed based on the information specified using the language provided. The results of these analyses can be used to remove inconsistency and potential deadlocks of the model and make the model self-contained and coherent.
- Hierarchical: The PSML-S is a multi-layer hierarchical model following the system-of-systems concept. The structure and the behavior are consistent and complete within the scope of each layer in the hierarchy. One can perform specification, analysis, simulation and execution on each layer independently.
- Service-oriented and dynamic composition and re-composition: PSML-S supports SOC. Each constituent (sub) system is a service that can reside locally or remotely. The ports of communication between two services are dynamic bound and cooperation agreement is dynamically negotiated. Services can be dynamically discovered and bound into the system at runtime. Once the system or a sub system is recomposed, the code generation service will regenerate the executable code from the new specification.

- Dynamic verification, validation, simulation, runtime monitoring, and evaluation: The recomposed model will be dynamically verified by Completeness and Consistency check and model checking. After the code generation, the model will be tested and validated in the simulation environment. Then, the executable for deployment will be generated. During the operation, the simulation environment will be used to monitor the execution process and the data collected during the operation can be used for performance, reliability, safety, a security evaluation.
- Data-centric: PSML-S model is a data-centric system model. System status is decided by the combination of observable values of different key attributes. Each key attribute of the target system is presented as a PSML-S data element. Data / data value can be transferred in the format of parameters.
- Event-driven and time-triggered: PSML-S support both event-driven and time triggered workflow. System workflow is essentially driven by events. Events can have parameters to carry more information. The system workflow can also be triggered by timing events. A timing service can be programmed to issue timing event at certain time points drive certain workflow forward.
- Policy-driven: PSML-S has a policy specification model and policy enforcement engine. The policy specification is separated from system's functional specification and is stored separately from the executable code, which allows dynamic modifications of the policies. The policies will be automatically checked and enforced during the analysis, simulation and execution. The policy model offers a centralized and distributed control on policies, and hence reduced the difficulty of policy specification, implementation and modification.

### 3. PSML-S Metamodel and Models

The PSML-S metamodel is defined following the four-layer modeling architecture from UML (Unified Modeling Language) [8], which is a generally accepted framework for defining the precise semantics required by hierarchical models. The four layers of models defined in PSML-S are:

- At meta-metamodel layer: the MOF (Meta Object Facility) meta-metamodel [9]
- At metamodel layer: the PSML-S metamodel
- At model layer: PSML-S model
- At user object layer: a real system

This paper focuses on the PSML-S metamodel. The constructs or building blocks of metamodel are called model elements. The model elements are grouped into

four logic packages, each of which describes a different logic aspect of the system under modeling. The four packages are:

- Core model package;
- Structure model package;
- Behavior model package; and
- Constraint model package.

The relationship among the packages is depicted in Figure 1, which represents a model of models, or a metamodel. The core model package defines basic model constructs, on which the other three model packages are based, as indicated by the arrows from the three models to the core model. The behavior model is built on the static structure model, as indicated by the arrow from the behavior model to the structure model. Based on the structure and behavior model, constraint model defines the constructs for specifying static or dynamic constraints. The rest of this section presents the individual models and their interactions.

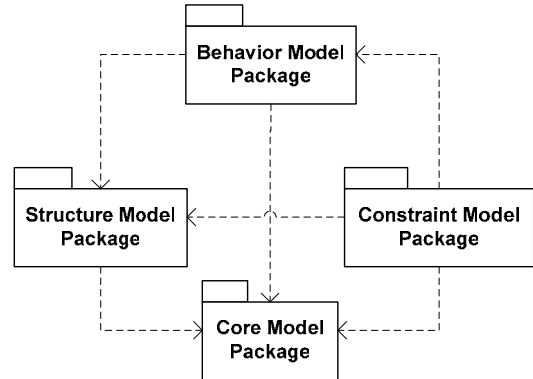


Figure 1: PSML-S metamodel

#### 3.1 Core Model Package

The core model package consists of the data types and operators that are used for defining other packages. The core package can be further decomposed into two sub-packages: the Data Types and the Operators. The former defines a set of data types which are used to categorize PSML-S model elements and the latter defines the relations and operations among the model elements.

#### 3.2 Structure Model Package

The structure model package consists of model elements for defining the static structure of a PSML-S model (program). The model elements defined in this model package include:

**Actor:** An actor is a model element that represents a system or a component with clear boundary that interacts with other actors. An actor is implemented as a service agent that can reside locally or remotely.

**Action:** An action is a model element that represents an operational process to change internal status of an actor.

**Attribute:** An attribute is defined to provide security, safety, performance, timing or reliability information. An attribute can be associated with any model element.

**Condition:** A condition is a predicate on data elements used to determine the course of a process taken by actors.

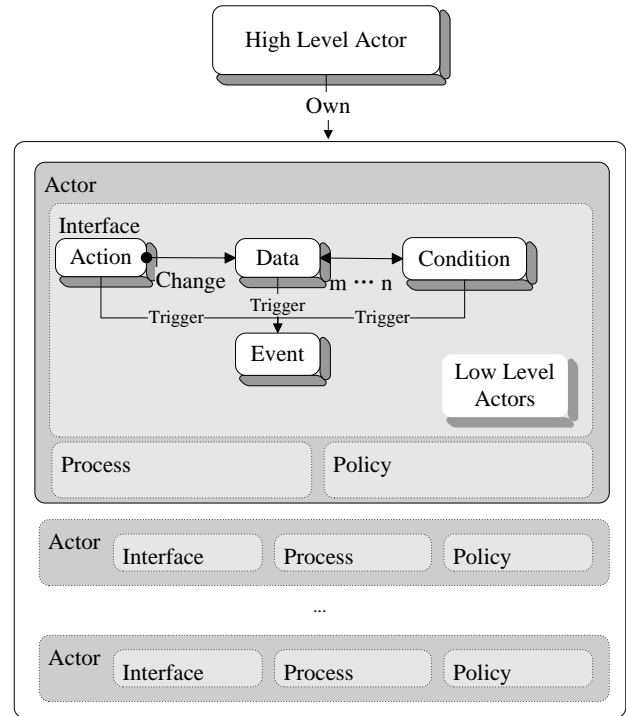
**Data:** A data element is an information carrier that represents the internal status of an actor.

**Event:** An event is a model element that represents an observable occurrence with no time duration of input to an actor or output from an actor.

**Relation:** A relation is defined to provide relationship information between any model elements. There are different relations defined between any two model elements.

The process specification is specified using the model elements defined in the structure model. In this way, the static structure model defines the building blocks for constructing the dynamic system behavior specification.

Like a system can consist of subsystems or components, an actor can consist of (own) a number of actors, as shown in Figure 2. Besides the actor-actor relation, an actor can own data elements. The values of data elements can be used on conditions and actions, which can trigger an event. In other words, an action can change the value of a data element and the change can trigger an event.



**Figure 2:** The hierarchical structure of model elements

Figure 2 illustrated a part of the relations among the model elements. The more detailed cross relations are summarized in Table 1. The relations are not symmetric. Each cell in the white area represents the relations from the element on the left to the element on the top.

**Table 1:** Cross relations among PSML-S elements

Element	Actor	Action	Condition	Data	Event
Actor	Own Compose Composed of	Own Perform	Own	Own	Own Initiate Receive
Action	Belong to Performed by	Sequential Concurrent	Change	Change Carry	Trigger Handle
Condition	Belong to Describe	Pre-status Post-status	Compose Composed of	Specify	Trigger
Data	Belong to	-	Compose	Compose Composed of	Trigger
Event	Belong to Initiated by Received by	Triggered by Handled by	Change	Carry	Compose Composed of
Attributes	Associate to	Associate to	Associate to	Associate to	Associate to

The actors and actions are the elements that contain program code to perform computing tasks. We differentiate actors and actions as atomic and composite. An atomic actor and an action may not be or do not need to be further represented into smaller components either

because there is no further information available or there exists a local service or a remote service that represent the actor or can perform the required action. A composite actor or action is built on the atomic actors and actions and other necessary model elements. Furthermore, a



Different from reconfiguration and re-composition processes, policy specification is more volatile. It can be modified at runtime and without disrupting the execution of the functional code. For reconfiguration and re-composition processes, when reconfiguration or re-composition in progress, the execution of the functional code is halt until the reconfiguration or re-composition has completed and verified (by model checking) and validated (by testing).

Figure 4 shows the implementation of the policy mechanism. The system specification is analyzed and the policies (non functional constraints) are extracted and represented in Policy elements. The Policy elements are stored in the Policy database. The policy engine element enforces the policy and triggers an event if a policy element is violated.

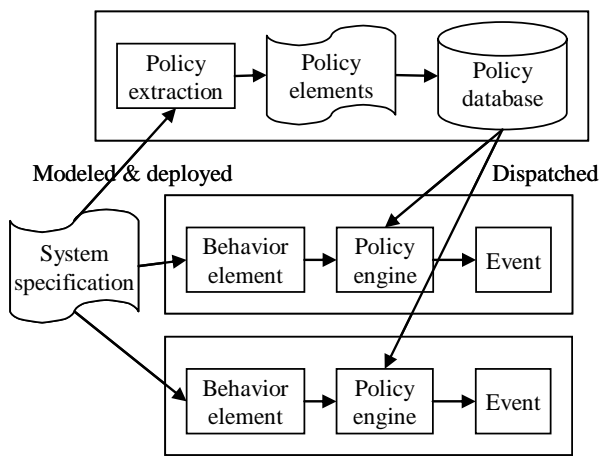


Figure 4: Policy specification and enforcement

#### 4. PSML-S Modeling / Execution Processes

System requirements or informal specifications are normally written in natural language. These requirements are easy to understand for human users but difficult to do programmatic analysis directly based on them. To obtain the PSML-S model (the process specification written in PSML-S language) from the informal specifications, users need to analyze the requirements, decompose the information into PSML-S model elements, and manually write the PSML-S specifications. Fortunately, PSML-S is very close to the natural language and the effort is much lower than writing code to implement the requirements directly using a programming language.

According to the PSML-S metamodel, the requirements must be first grouped into three packages: structure package, behavior package, and constraint package. Then, each package is decomposed into the model elements of the package.

Structure decomposition and specification: Decompose the system into components that can be described by the

PSML-S building blocks (actors, actions, attributes conditions, data, and events). In order to perform automated execution, the atomic actors and actions must be able to map to existing services. .

Process (behavior) specification: Use the structural building blocks to compose processes (composite actors and actions). If alternative (redundant) actors and actions exist and fault-tolerant computing is necessary, compose reconfigurable or re-composable processes. The process specification includes internal (local) data and control flows and collaboration specification with external services.

Policy specification: Constraint-related items in the requirements are specified as Policy elements which will be enforced by the Policy engine element.

The overall modeling process in PSML-S is shown in Figure 5.

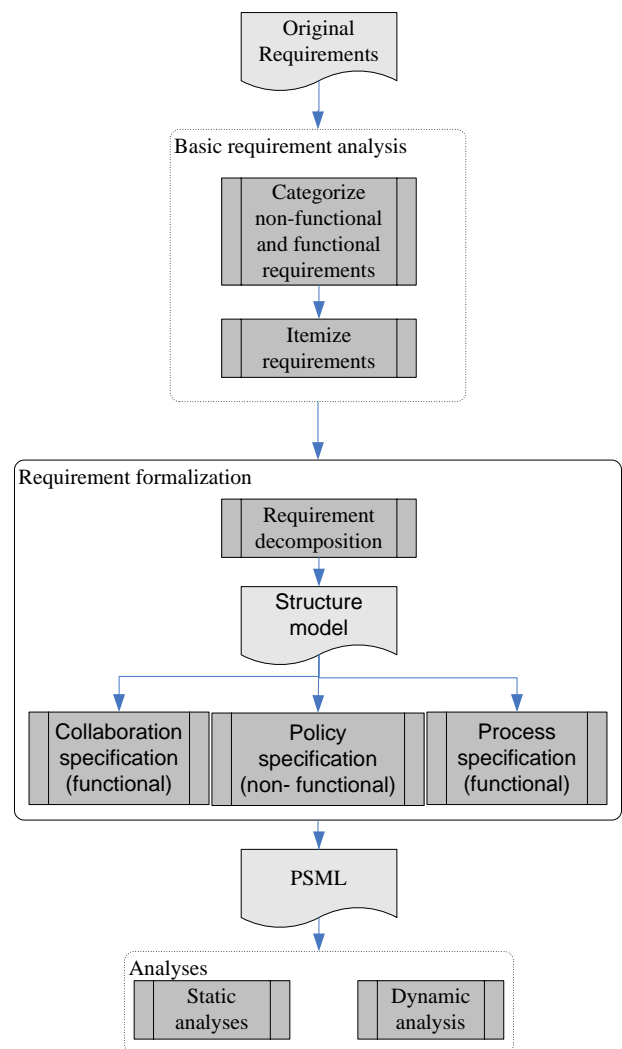


Figure 5: PSML-S modeling process

## 5. Case Study

This section uses PSML-S to specify the structure model and the behavior model of a virtual bookstore. As shown in Figure 6, the structure model contains five actors. Within each actor, there are data, condition, and event elements.

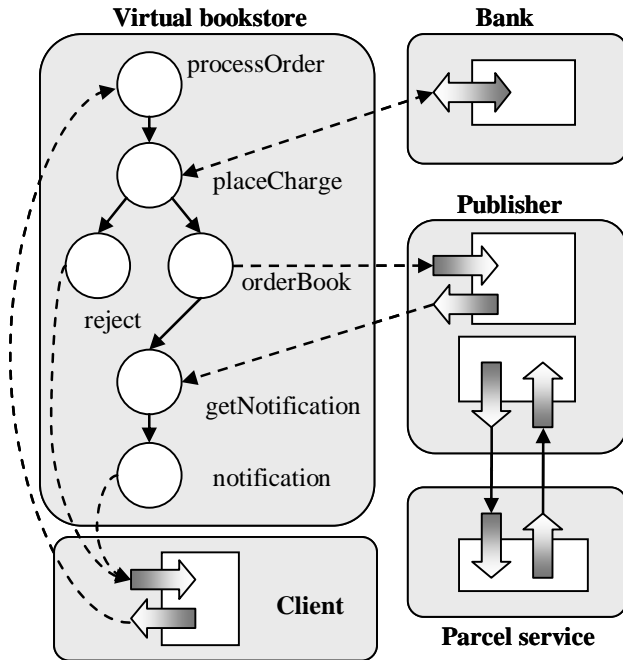


Figure 6: Virtual bookstore model

1. **Actors:** There are five actors in the system. They are the client, the bookstore, the bank, the publisher, and the parcel service.

- **Client:** It is the interface between the end user and the system. It takes the end user's order and sends the order to the actor: Bookstore. The order contains the book information to be ordered and the credit card information of the end user.
- **Bookstore:** It is the main actor we focus on in this example. It sends charge message to the bank actor, which contains the credit card information and the amount of charge. Depends on the return message from the bank, the bookstore actor will either reject the order or ask the publisher actor to deliver the book. Once it receives the delivery confirmation from the publisher, it notifies the client.
- **Bank:** It charges the credit. It returns the failure message if the charge fails, otherwise, it confirm the completion of charge.
- **Publisher:** It receives the order from the bookstore and requests its shipper to deliver the book. Once the book is shipped, it returns the confirmation to the bookstore.

- **Parcel service:** It receives the shipping request from the publisher, ships the book, and confirms the shipment.

2. **Condition** in the actor bookstore (in this example, we discuss the bookstore actor only). One condition is defined: `ChargeFail`, it is false if the bank confirms charge, otherwise, it is true.

3. **Data** in bookstore actor: Data item from the client: (book detail, credit card detail, deliver address). Data to bank actor: (credit card detail, amount to charge); Data to publisher: (book detail, deliver address). Data to client: (rejection) or (confirmation).

4. **Actions** in bookstore actor. There are six actions on the bookstore actor:

- `ReceivingBookOrder`
- `RejectingOrder`
- `SendingBookOrdertoPublisher`
- `GeneratingReceipt`
- `SendingReceiptToClient`
- `Events in bookstore actor`
- `RequestCreditcarCharge`
- `SendingBookRequestiontoPublisher`

Based on the structure model elements above, the behavior model element, the process element can be constructed as follows. Note, on the process the process element related to the bookstore actor is given.

- **BeginToProcessOrder:** It is associated to the event: `CustomerOrderReceived`.

```
using ACTOR:Bookstore
do ACTION:Bookstore.ReceivingBookOrder
emit EVENT:Bookstore.RequestCreditcarCharge
```

- **BusinessProcessRejection:** It is associated to the event: `ChargeFail`.

```
using ACTOR:Bookstore
using ACTOR:Client
do ACTION:Bookstore.RejectingOrder
do ACTION:Client.SendingCustomerRejection
```

- **BusinessProcessNormal:** It is associated to the event: `SuccessfullyCharging`.

```
using ACTOR:Bookstore
using ACTOR:Client
do ACTION:
Bookstore.SendingBookOrdertoPublisher
emit EVENT:
Bookstore.SendingBookRequestiontoPublisher
```

- **BusinessProcessReceipt:** It is associated to the event: `SendingConfirmation`.

```

using ACTOR:Bookstore
using ACTOR:Client
do ACTION:Bookstore.GeneratingReceipt
do ACTION:Bookstore.SendingReceiptToClient
do ACTION:Client.SendingCustomerReceipt

```

The PSML-S is not a standalone specification language. The entire development environment has been implemented. Once the PSML-S code is written manually, the rest of the task can be done automatically: A verification tool can verify the completeness and consistency among the conditions in the specification and a model checking tool can check deadlock conditions and reachability of actions [2][13][14][15]. The code generation tool can translate the PSML-S specification into executable code. The code can be executed and tested in a simulation environment before the code is deployed. However, these topics are beyond the scope this paper.

## 6. Conclusion

As a part of an integrated Web service development environment, PSML-S provides the modeling and specification capacity. The model specified in PSML-S can be verified by completeness and consistency and model checking tool, and then, automatically translated into executable for simulation and testing. This paper presented the metamodel of PSML-S and the model elements in different model packages.

## References

- [1] X. Fu, T. Bultan, and J. Su, "Formal Verification of E-Services and Workflows," Proc. Workshop on Web Services, E-Business, and the Semantic Web (WES), LNCS 2512, Springer-Verlag, 2002, pp. 188–202.
- [2] H. Huang, W. T. Tsai, R. Paul and Y. Chen, "Automated Model Checking and Testing for Composite Web Services", 8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC), Seattle, May 2005, 300-307.
- [3] Intel: Service-Oriented Enterprise, The Technology Path to Business Transformation. [http://www.intel.com/business/bss/technologies/soe/soe\\_b\\_ackgrounder.pdf](http://www.intel.com/business/bss/technologies/soe/soe_b_ackgrounder.pdf)
- [4] Nickolas Kavantzias, David Burdett, Tony Fletcher, Yves Lafon, Web Services Choreography Description Language (WS-CDL) Version 1.0, W3C Working Draft 17 December 2004. <http://www.w3.org/TR/ws-cdl-10/>
- [5] Frank Leymann, Web Services Flow Language, Version 1.0, May 2001. <http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>
- [6] OASIS: ebSOA proposal: <http://www.oasis-open.org/committees/ebsoa>
- [7] OASIS: Business Process Execution Language for Web Services (BPEL4WS), 2003. <http://xml.coverpages.org/bpel4ws.html>
- [8] Object Management Group, Unified Modeling Language (UML), version 2.0. <http://www.omg.org/technology/documents/formal/uml.htm>
- [9] Object Management Group, Meta Object Facility (MOF) 2.0 Core Specification. <http://www.omg.org/docs/ptc/04-10-15.pdf>
- [10] R. Paul, "DoD Towards Software Services", Tenth IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 05), February 2005, pp. 3-6.
- [11] M. P. Singh, M. N. Huhns, *Service-Oriented Computing*, John Wiley & Sons, 2005.
- [12] Zoran Stojanovic, Ajantha Dahanayake, *Service-Oriented Software System Engineering: Challenges and Practices*, Ideal Group Publishing, Hershey, 2005.
- [13] W. T. Tsai, R. Paul, L. Yu and X. Wei, "Rapid Pattern-Oriented Scenario-Based Testing for Embedded Systems," in Software Evolution with UML and XML, edited by H. Yang, Idea Group Publishing, London, 2005, pp. 222-262.
- [14] W. T. Tsai, Y. Chen, Ray Paul, Hai Huang, Xinyu Zhou, Xiao Wei, "Adaptive Testing, Oracle Generation, and Test Script Ranking for Web Services," in Proc. 29th Annual International Computer Software and Applications Conference (COMPSAC), Edinburgh, July 2005, pp 101-106.
- [15] W.T. Tsai, Y. Chen, R. Paul, "Specification-Based Verification and Validation of Web Services and Service-Oriented Operating Systems", Tenth IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 05), Sedona, February 2005, pp.139 - 147.
- [16] O. Zimmermann, Sven Milinski, M. Craes, F. Oellermann, "Second Generation Web Services-Oriented Architecture in Production in the Finance Industry", OOPSLA '04, Oct. Vancouver, 2004.