# Requirement Engineering in Service-Oriented System Engineering

W. T. Tsai, Z. Jin*, P. Wang* and B. Wu*
Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287-8809
*Academy of Mathematics and System Science,
Chinese Academy of Sciences, Beijing 100080, China

## Abstract

*Service-oriented computing has received significant attentions recently, and many applications are being developed using this approach. Thus there is a need to analyze system requirements. This paper examines issues related to service-oriented requirement engineering (SORE). SORE focuses on modeling, specifying, and analyzing application requirements for software that will be developed in a service-oriented manner running in an SOA infrastructure. This paper also presents key features of SORE and some technical challenges.*

## 1. Introduction

Service-Oriented Architecture (SOA) receives significant attentions recently. This paper examines the key issues of Service-Oriented Requirement Engineering (SORE). SORE, like traditional requirement engineering, concerns with specification and analysis of system requirements and constraints.

SORE is an important topic in Service-Oriented System Engineering (SOSE) and is an emerging area. Requirement engineering has evolved from classical methods such as SREM [1], to OO methods using UML [1], and to SORE. Several projects have initiated in this direction. An SORE workshop in 2004 was held to discuss the requirement engineering theories and technologies for SOA systems. IBM proposed the key activities in service-oriented modeling [1], stated the motivation [4], and provided guidelines for applying various patterns and selecting services [9]. In 2007, another workshop was held to address SORE.

This paper is organized in this manner. Section 2 lists some SOA variants that can be used to develop applications. Section 3 lists some key features of SORE. Section 4 discusses some main technical issues to support SORE.

## 2. SOA Variants

Traditionally, only services can be published and discovered, but recently several new SOA frameworks have been proposed where many other artifacts can be published and discovered, and they will affect SORE. Table 1 shows a partial list of things that can be published and discovered.

Table 1: SOA Publishable Items

| Reusable artifacts | Description |
|---|---|
| Services | They are basic building blocks in SOA, and allow software development by composition. |
| Workflows or collaboration templates | These specify the execution sequence of a workflow with possibly multiple services. They allow rapid SOA application development. |
| Application templates | These specify entire applications with their workflows and services, and they allow rapid SOA application development. |
| Data, data schema, and data provenance | Data and associated data schema such as messages produced during SOA execution can be published and discovered. |
| Policies | Policies are used to enforce SOA execution and can be published to be reused.. |
| Test scripts | Consumers, producers and brokers can publish test scripts to be used in verification by other parties. |
| Interfaces | GUI design can be used and linked at runtime to facilitate dynamic SOA application with changeable interfaces. |

The current SOA is **producer centric**, i.e., producers publish services, and consumers search and discover their needed services, and use them in their application development.

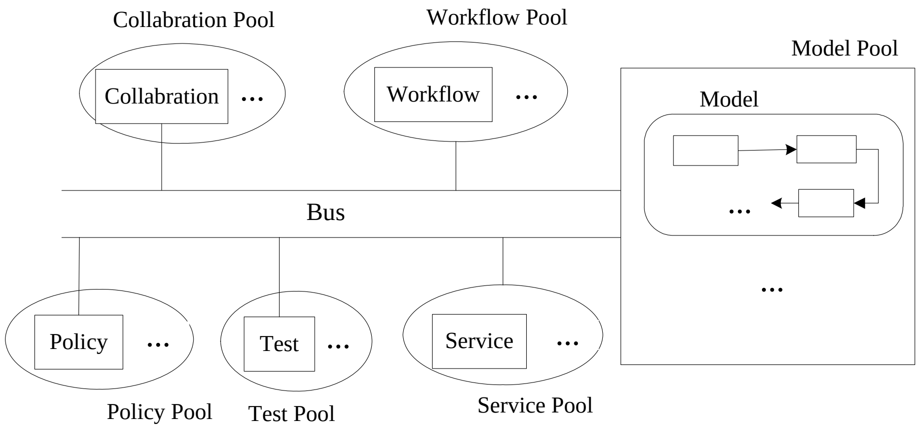

**Figure 1** Service-Oriented Infrastructure

But SOA can be **consumer centric**, i.e., consumers publish their needs, and let providers to supply the needed services and/or workflows, collaboration templates, or even application templates.

SOA can also be **broker centric**, i.e., brokers publish test scripts or specification, and let producers supply the services/workflows, and let consumers discover the services and use the test scripts for testing.

Essentially, these new SOA variants allow a variety of SOA artifacts to be published and discovered in various ways including producer-centric, consumer-centric, or broker-centric manner. They also allow a flexible way of composing applications using a variety of reusable artifacts. Figure 1 illustrates an SOA infrastructure where these artifacts are available through a communication backbone such as Enterprise Service Bus (ESB), and controlled by a controller, the controller manage the discovery, binding, and execution by using reusable assets connected to the communication backbone.

## 3. Service-Oriented Requirement Engineering (SORE)

SORE differs from traditional requirement engineering as it assumes that the concerned application will be developed in an SOA framework running in an SOA infrastructure. Many SOA infrastructure systems are developed on top of an OO infrastructure, e.g., C# is both an OO language as well as an SOA language because it has classes and methods. Thus an SOA application may still have OO classes and methods, however, this paper will focus on the SOA aspects only.

While SORE involves many activities commonly in traditional requirement engineering including modeling, specification, static and dynamic analyses, but the entities identified and the process used are different, and the way models are used in the development process are different. Instead of identifying objects and classes, SORE focuses identifying **services and workflows**, and modeling the application using identified service and workflow specifications so that real services ad workflows can be discovered either at design time (static SOA) or at runtime (dynamic SOA). Similarly, SORE may identify the needed policy specification for execution control and management, and let the SOA application discover those policies from the policy pool either at design time or at runtime. Table 2 compares SORE and OORE (OO requirement engineering).

A dynamic SOA application may also involve re-binding, re-composition, and re-architecting systems by choosing different services, workflows, collaboration, and system architecture at runtime. These need to be considered at the SORE stage so that system reconfiguration can be executed according to a plan. A dynamic SOA application may change its services, workflows, user interfaces, policies, data, data schema based on the system status as well as new user needs.

### 3.1. SOA Application Lifecycle

Requirements specified are often affected by the development lifecycle. In the traditional Waterfall model, significant effort is spent to acquire, specify and analyze the requirements during the analysis phase; however, in an agile process, requirements are stated only as informal scenarios during development, as the focus of the agile process is mainly on code development. Thus, the way requirements are stated and used are distinct in different lifecycle models. Several SOA development lifecycle have been developed, and they are different from traditional software development lifecycles.

**Table 2** Distinction between SORE and OORE

| | **SORE** | **OORE** |
|---|---|---|
| System structure | Publish and reuse services, workflows, policies, collaboration templates, and application templates. | System is structured with interacting objects, classes and design patterns. |
| Languages | A verity of modeling and specification languages can be used, including BPEL, C#, PSML, WSDL. | OO programming language: C++, Java, Smalltalk, and OO modeling languages such as UML. |
| Infrastructure environment | Architecture consists of brokers, consumers and providers, and an SOA computing infrastructure uses SOA protocols and standards such as SOAP, BPEL, WSDL. | OO computing environment such as CASE tools and IDE. |

**IBM SOA Foundation Architecture**

The IBM SOA Foundation Architecture [5] describes a lifecycle model consists of two development activities (modeling and assembly) and two operation activities (deployment and management) integrated into a process as illustrated in Figure 2.

Furthermore, it also has runtime governance activities to provide guidance and oversight for the target SOA application. The four phases are performed iteratively. The entire process will be controlled and orchestrated by the governance policies.

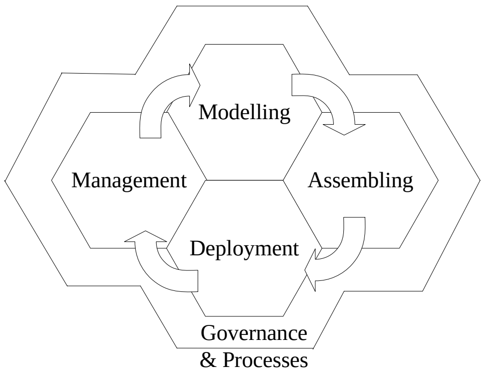

**Figure 2** IBM SOA Foundation Architecture

The lifecycle model is based on a model-driven 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.

This lifecycle model shows that the SOA application lifecycle is distinct from traditional lifecycle including Waterfall, Spiral, and agile processes. One distinction is that the SOA lifecycle includes an explicit process of discovering reusable assets such as services and workflows. Even though traditional lifecycle may also take a model-based approach, but the SOA modeling is unique. In addition to model applications using services and workflows, the model is often executable as both workflow and service specifications are often executable. This is important because once an application is modeled, a service-oriented simulation can be immediately performed [10] to evaluate the system during SORE.

Based on this lifecycle model, Arsanjaini suggested service-oriented modeling need to include analysis and design of services [8]. He proposes the process of service-oriented modeling and architecture: service identification, service classification and categorization, subsystem analysis, component specification, service allocation, and service realization. He also separates customer role from provider role in modeling activities. Maiden [8] emphasizes the availability of services in registries during SORE, and suggests that existing services guide the requirement modeling.

**Other SOA lifecycle**

Other SOA development lifecycle models are also available. For example, even though IBM SOA Foundation model has verification and validation activities, but they are not explicitly shown in the diagram. In [3], another SOA lifecycle model is presented that explicitly show these activities such as simulation, testing, model checking, and completeness and consistency checking. Simulation is important in the SOA lifecycle and it can be performed at each stage. During SORE, simulation can be performed using actual services and workflows as well as based on service and workflow specifications. Once the model is updated, simulation can be performed again to validate the system requirements.

This lifecycle model also has a feedback loop, i.e., if there is any change happen after management phase, a new round of modeling, verification and validation, code generation, deployment, execution, monitoring, and management can be initiated.

## 3.2. SORE Framework

SORE assumes an SOA framework will be used, thus some design decisions are made before even starting any requirement analysis. This is different from traditional requirement engineering where analysis and design are often separate activities, with design following requirement analysis, probably in an iterative manner. SORE has the following major features.

**Reusability-Oriented and Cumulative**

SOA emphasizes on reusability, once an item is published, it can be reused by others. Recall that in various SOA variants, various SOA items can be published to be reused by others including services, workflows, collaboration templates, application templates, and interfaces. Similarly SORE also emphasizes on reusability, specifically, SORE can reuse not only SORE items developed for the same project during the next iteration, but also those items developed in other projects but published. As SORE items are identified and developed, they can be stored for reuse, in this way, as more projects are developed in the SOA manner, more reusable items will be available for reuse. Reusable items include services, workflows, collaboration templates, and application templates, and ontology items.

Not only SORE *assets* can be reused, but also SORE *processes* can be reused. Furthermore, SORE processes, such as simulation, model checking, and policy management, may be automated and reused either as services or workflows in an SOA manner as illustrated in Figure 3. In this manner, SORE is treated as an application and various activities are treated either as services or workflows, and SORE engineers can compose various SORE activities to form a specific SORE process.  In other words, an SORE process can be a re-configurable process, and an SORE engineer can tailor his/her process by picking up the needed SORE services or workflows for specific application development. In this way, Figure 3 illustrates a family of SORE processes that can be composed and re-composed according to specific needs of a process. In this way, SORE engineers can keep on adding SORE processes as workflows or services for reuse in the next phase or project. Eventually, a large collection of SORE processes can be automated and available for reuse.

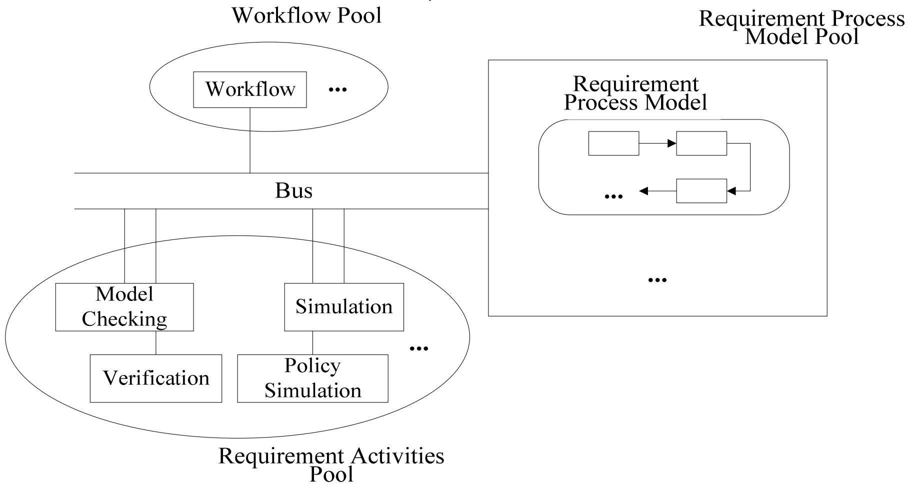

**Figure 3** SORE Processes Reused in an SOA Manner

**Domain-Specific**

Currently, most requirement processes/techniques are often domain independent, as they are often applicable to a wide range of applications and domains. However, an important feature of SORE is that it will use domain-specific items including ontology, services, workflows, collaboration templates, application templates, user interfaces, and policies, and thus SORE is inherently domain specific. For example, a banking application may use reusable banking-related ontology, services, workflows and application templates, but it probably will not find biomedical ontology, services and workflows helpful. Thus, as SORE getting acceptance and attention, domain-specific SORE assets and processes will be developed, potentially a large number of SORE domains may be available in future for major application domains such as e-business, banking, biomedical, transportation, and retail. These domain SORE assets will also grow as more people practice SORE.

**Framework-Oriented Analysis**

As reusable assets will be accumulated, many assets will be available, and these assets will be organized in some manner, such as according to the classification tree in an ontology system or other manners. The OO approach also encountered similar issues. One solution used is to develop an OO application framework and most well known framework is IBM San Francisco Framework [6]. One issue for application development using an OO framework is that requirement analysis is no longer confined to analyzing the requirements given, but also examining the reusable assets available in the given OO framework to see if these assets can be reused. This process is called framework-oriented analysis [14]. As SOA reusable domain-specific assets will be organized in some kinds of framework, eventually SORE will become framework oriented, but this time the framework involved will be a service-oriented framework rather than an OO framework as elements will be arranged as services and workflows rather than classes and methods.

## Model-Driven Development

Most SORE assumes a model-driven approach. For example, the first step of the IBM SOA Foundation model is to model the application, and the next step is to discover the needed services based on the model specified. One important issue here is what kinds of modeling language should be used in this phase. While UML is a popular modeling language and has been widely used for OO applications, it is not clear that it is the most suitable language for SOA applications. Furthermore, SOA may follow the Web. 2.0 (http://en.wikipedia.org/wiki/Web_2) and the modern SaaS (Software as a Service) approach (http://en.wikipedia.org/wiki/Software_as_a_Service) where each new version will be delivered rapidly possibly on the web, and it has been reported some companies already deliver a new version each thirty minutes today. Thus, the SOA application may be developed and deployed rapidly, but in such rapid development, the model needs to be in sync with the code. However, one of the problems of UML is that it is difficult to keep UML models in sync with the associated code.

Recently, another approach has been proposed. This approach is called SMMA (Single Model Multiple Analyses) where a set of core models are maintained, and a variety of analyses can be performed from the set of core models. Furthermore, code may be automatically generated from the core models. One such modeling language is PSML [4]. It can be used to model ontology, services, workflows, collaboration templates, policies, and application templates, and various analyses such as completeness and consistency analysis, path analysis, and simulation.

## Evaluation-based

Services can be discovered, selected, and deployed at runtime, but this also means that services and applications may need to be tested and evaluated at runtime unless only pre-selected services can be discovered and used, and all the pre-selected services/workflows must be thoroughly pre-evaluated before they can be placed in a service or workflow pool. The three stages for SOA evaluations are described in Table. 2.

**Table 2** Three Stages of Evaluations in SOA

| Stages | Evaluations Activities |
|---|---|
| Preparation stage | Pre-evaluation of services, workflows, collaboration templates, and application templates by various methods including simulation. |
| Composition stage | The composed application can be evaluated by various static and dynamic methods including completeness and consistency checking, model checking, and simulation. |
| Runtime stage | During runtime, various analyses can be performed including policy enforcement and simulation: |

The preparation stage focuses on evaluation of SOA items such as services, workflows, collaboration templates before they are discovered for application. Thus, this stage prepares these items so that they will be almost ready for composition.

In the composition stage, many static analyses are available such as model checking, completeness and consistency checking. In the static SOA, these analyses can be done at design time so that extensive analysis can be performed before deployment. In the dynamic SOA, as the services or workflows may be just identified and composed into an application, these analyses must be performed as a part of the dynamic composition process. To save time and effort, results from the preparation stage such as test scripts and test results can be reused.

Finally, in the runtime stage, various dynamic evaluations can be performed including policy enforcement, reliability modeling, and dynamic monitoring.

Simulation can be performed at each stage, e.g., in the preparation stage, sample applications can be evaluated with candidate services and workflows; in the composition stage, the actual composition can be simulated with selected services and workflows; in the runtime stage, simulation can be performed simultaneously with the actual system to compare results.

Many system behaviors can be checked only at runtime, and thus it can be enforced by a policy mechanism including policy specification, policy analysis, policy simulation, policy enforcement, and policy data analysis. The requirement model should support policy specification, analysis, verification, policy code generation, enforcement, monitoring and evaluation dynamically. This is a part of non-functional requirement system. Policy has its own lifecycle in parallel with the development process of service application, each decomposition level can be considered as reusable service. This is illustrated in Figure 4.

**User-Centric Analysis and Specification**

As web-based application becomes more end user-oriented, particularly in the Web 2.0 world, many end users will be able to compose applications without knowing details of software or programming languages. The web now enables users to constructs webblog, and mashup applications using online software services. These marshal a new way of software development with increasing emphasis on user interface, user interaction, rapid software composition using existing services, lightweight development processes, and visual validation. SORE can play two roles here, one is to help these end users to identify their application requirements, possibly using a set of visual tools, and the other is to identify tool features that support end users to rapidly develop their applications.

**Policy-Based Computing**

Another interesting feature of SOA is that it explicitly supports policy-based computing where computation constraints are specified separately but are enforced at runtime. Most of constraints today are security constraints, however, other kinds of constraints are also possible. While the policy mechanism is useful, it also introduces a new dimension of issues, i.e., policies need to be specified, analyzed, enforced, and evaluated, and thus like software it has a lifecycle model that is parallel with the software development as shown in Figure 4. Both functional requirements and constraint (or policy) requirements need to be acquired and represented in a policy specification language that is executable. While policies specified are often executable, however, policy evaluation and execution are different from software evaluation and execution because policies need to be evaluated together with the functional software to be enforced.

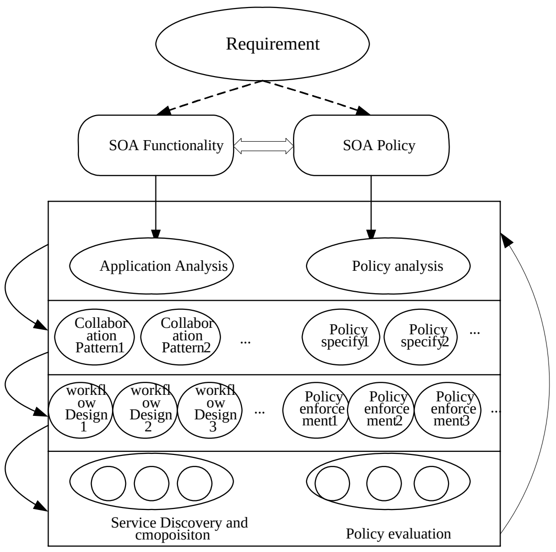

**Figure 4** Policy lifecycle

## 4. Technical Approaches

This section outlines key technical approaches.

### 4.1. Software-Oriented Ontology

Ontology defines the entities, relations and rules comprising the vocabulary of a domain. It facilitates computer to interpret and process the terms. Ontology is used in SORE. Ontology is used in many other tasks or projects such as semantic web. However, ontology in SORE provides service-oriented modeling and analysis, facilitates code generation, and promotes software reusability. Traditionally, ontology is often used for classification, navigation, concept identification and query, and reasoning. While these ontology capabilities are still useful for SORE, but the main use of ontology is to construct a domain model that is capable of developing SOA applications rapidly in a model-driven SOA lifecycle model. As SOA software will have kinds of structuring such as application architecture, collaboration structure, services and workflow, it is advantageous to divide an ontology system into different kinds of ontology systems, one for each level, thus software-oriented ontology is needed. In contrast to traditional way of constructing an ontology system for the entire SORE process, this approach follows the divide-and-conquer approach commonly used in software engineering.

Each ontology system now focuses on one aspect of SOA software development, and they can cross reference each other for easy identification to maximize reusability and facilitate rapid code generation.

Another distinct feature is if these ontology systems use the same modeling language for ontology, services and workflows, then these ontology systems can be seamlessly integrated with the SOA software development platform. For example, an SOA application development can have five ontology systems:

1) Application ontology (AO): this defines concepts and relationships related to applications.

2) Collaboration ontology (CO): this defines various collaboration templates with associated workflows and services, and the CO cross references to AO, WO and SO for easy service and workflow identification;

3) Workflow ontology (WO): this defines concepts and relations of workflows and their patterns. In WO, specific workflows from different domains are classified and relations are also specified facilitating collaboration. A domain-specific WO can also be developed to facilitate rapid workflow identification. Needed workflow can be discovered and composed to form a collaboration pattern. New workflow can be added into ontology to complement it [16].

4) Service ontology (SO). This defines concepts and relationships of services.

5) Test ontology (TO). This defines concepts and relationships of test scripts and cases.

If these ontology systems are specified using the same modeling language, it will promote consistency among these ontology systems and support cross references [13]. This feature also supports discovery and matching. Instead of depending on keywords or feature matching using an ontology system, it is possible to use the cross references or the context of specific items. For example, a given application in AO may reference several collaboration templates in CO, and each collaboration template may reference several services and workflows in SO and WO. In this way, a user may find the context of a given service or workflow by tracing from SO and WO to CO, and eventually to AO, as shown in Figure 5.

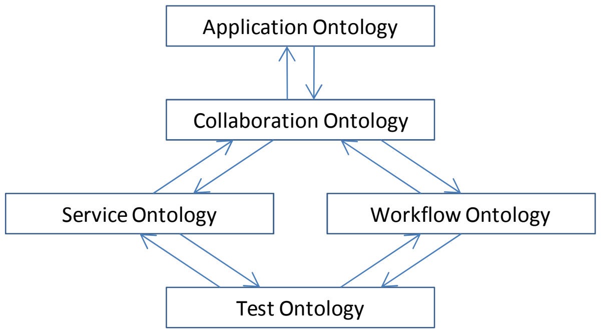

**Figure 5** Cross Referencing Ontology Systems

## 4.2. Service-Oriented Simulation

Simulation is an important step for evaluating system requirements, and it can reveal and analyze dynamic system behaviors that might not be visible from static analyses. Traditional requirement engineering tools often allow specifications to be executable, e.g., SREM, an early requirement engineering tool, already allows R_Net to be simulated by tracing the execution path. This feature allows simulation to be performed based on the requirement specification. A service-oriented simulation framework with its infrastructure will be needed to perform service-oriented simulation for SOA applications. Recently, several service-oriented simulation frameworks are available including DDSOS [10]. Using a service-oriented simulation framework, it is possible to simulate the SOA application according to its specification, possibly with many existing and reusable services and workflows. In case existing services and workflows are reused, this is hybrid simulation as some parts of system will be run using actual code rather than simulation code. As service-oriented simulation is a new research area and lots of work still can be developed.

## 4.3. Model-based Development

One key issue for SOA model-based development is the development of modeling language suitable for rapid SOA application development. In such rapid application lifecycle model, many tasks will be performed based on the model specified. Specifically, various static and dynamic analyses such as completeness and consistency and simulation will be based on the model developed, even based on partially developed with existing services and workflows; design or assemble will also based on model developed with various ontology systems and service brokers; code will be automatically generated based on the assembled model with reusable services and workflows; test scripts will be generated based on the assembled model and reusable test scripts associated with services and workflows; policies will be identified and specified together with functional model using the same modeling language. In this way, as the software needs change, the model will be updated, re-analyzed, and re-assembled, and code will be re-generated and re-tested and re-evaluated. These new requirements put significant burden on the modeling language.

## 4.4. Nonfunctional Requirements

Traditional non-functionality (such as security, reliability, and integrity) requirements still need to map to SOA systems, but unfortunately ensuring these properties on SOA systems are often open research problems at this time. For example, multilevel security on SOA systems is still a research problem, software reliability estimation and modeling is another open research problem. Thus, as an SOA application may consist of many reusable services and workflows, and these nonfunctional requirements need to apply across the software including those services and workflows that are reused. It is necessary to develop a mechanism to specify and analyze these nonfunctional requirements for SORE.

## 5. Conclusion

As SOA becomes popular, SORE will become important and research in SORE has just started recently. SORE has its own features: model-based, architecture-based, reused-oriented, framework-oriented analysis, simulation-based analysis with formal analysis, ontology-based. Compared with traditional requirement engineering, the distinction lies on modeling techniques, model-based development process, runtime behaviors including publishing, discovery, composition, monitoring and enforcement, Formal methods can be applied including reasoning and model checking, but they need to be updated to handle SOA concepts such as services and workflows.

## References

[1] Arsanjani A., “Service-Oriented Modeling and Architecture”, IBM Technical Report, 2004, available at http://www.ibm.com/developerworks/library/ws-soa-design1/

[2] Chonoles M. and James A. Schardt (2003). UML 2 for Dummies. Wiley Publishing. ISBN 0-7645-2614-6.

[3] Cotroneo D., A. Graziano, and S. Russo, “Security requirements in service oriented architectures for ubiquitous computing”, Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing, p172 – 177.

[4] Endrei M., et al, “Patterns: Service-oriented Architecture and Web Services Redbook”, SG24-6303-00, April 2004.

[5] IBM Developers Works. IBM SOA Foundation: An architectural introduction and overview, http://www.ibm.com/developerworks/webservices/library/ws-soa-whitepaper.

[6] Kara D., “IBM's San Francisco Project - Java-based application development software”, Software Magazine, Oct, 1997.

[7] Lichtenstein S., L. Nguyen, A. Hunter, “Issues in IT Service-Oriented Requirements Engineering”, Australasian Journal of Information Systems, Vol. 13, No. 1, 2005.

[8] Maiden N., “Servicing Your Requirements”, IEEE Software Volume 23, Issue 5, Sept.-Oct. 2006 Page(s):14 – 16.

[9] Tsai W.T., B. Xiao, R. A. Paul, and Y. Chen, “Consumer-Centric Service-Oriented Architecture: A New Approach”, in Proc. of IEEE 2006 International Workshop on Collaborative Computing, Integration, and Assurance (WCCIA), April 2006, pp. 175-180.

[10] Tsai W.T., C. Fan, Y. Chen, R. Paul, "A Service-Oriented Modeling and Simulation Framework for Rapid Development of Distributed Applications", Simulation Modeling Practice and Theory, Elsevier, 14 (2006), pp. 725-739.

[11] Tsai W.T., Xiao Wei1, Ray Paul, Jen-Yao Chung, Qian Huang, and Yinong Chen, "Service-oriented system engineering (SOSE) and its applications to embedded system development," Service Oriented Computing and Applications, Springer London, March 2007.

[12] Wang P., Z. Jin, and Lin Liu, “An Approach for Specifying Capability of Web Services based on Environment Ontology,” Proceedings of IEEE International Conference on Web Services (ICWS’06), pp. 365-372, 2006.

[13] Weerawarana S., “Business Process with bpel4ws: Understanding BPEL4WS (part 1)”, Technical report, IBM, 2002. http://www-106.ibm.com/developerworks/webservices/library/wsbpelcol1/. This seems to be an old reference, new versions of BPEL is available.

[14] Zhu F. and W. T. Tsai, “Framework-Oriented Analysis”, Proc. of IEEE COMPSAC, 1998, pp. 324-329.

[15] Zimmermann O., Pal Krogdahl, and Clive Gee, “Elements of Service-Oriented Analysis and Design”, IBM Technical Report, 2004, available at http://www.ibm.com/developerworks/library/ws-soad1/.