Service-Oriented Software Reengineering: SoSR

Sam Chung[1], Joseph Byung Chul An[1], Sergio Davalos[2]
[1]Computing & Software Systems
Institute of Technology
University of Washington, Tacoma
{chungsa, anba} @u.washington.edu
[2]Milgard School of Business
University of Washington, Tacoma
sergiod@u.washington.edu

Abstract

Service-Oriented Computing (SOC) enables the development and design of loosely coupled software components for integration with other software system. Since most legacy system were not designed and developed with services components, current legacy software systems require modernization (reengineered) into a target system made up of a set of loosely coupled services. A methodology for service-oriented software reengineering (SoSR) is proposed for applying SOC to legacy systems. The SoSR methodology, a synthesis of best practices, is architecture-centric, service-oriented, role-specific, and model-driven. It is conceptualized from a three-service-participants model, 4+1 view model, and RACI chart. The SoSR methodology is applied in the modernization of a legacy system, a retail business information systems. The results show that this methodology can help software developers and system integrators in reengineering tightly coupled legacy information systems into service-oriented information systems. By including a business process engine for executing composite services with existing applications and database servers, SOC can affect future information system design, deployment, and integration.

1. Introduction

Service-Oriented Computing (SOC) [15] has emerged as a software development paradigm that enables the design of software systems as a set of services, i.e. Software as a Service (SAAS) [6, 13]. SOC software systems can be recursively constructed as a set of services. Compared to current distributed middleware technologies such as RMI, CORBA, and mobile agents, SOC results in software systems that are more naturally integrated due to the use of standardized service interfaces and interaction protocols. The standard specifications for SOC supporting technologies evolving and implementation platforms such as Microsoft .NET and J2EE are being improved.

The SOC paradigm enables a software developer to design loosely coupled software components that can be integrated with other software systems. Most legacy software components were not designed and developed as services. Current legacy software systems need to be converted into a set of loosely coupled services before integration with other software systems. Software development methodologies such as dynamic systems development method, crystal methods, scrum, lean development, feature-driven development extreme programming (XP), and adaptive software development [7] focus on how to develop a software system, not on modernizing current legacy components into a set of loosely coupled services.

The purpose of this project is a Service-Oriented Software Reengineering (SoSR) methodology for reengineering a legacy software system into a service-oriented software system(s) based on SOC. Software engineering or reengineering methodologies with or without Web services do not specify tasks and responsibilities in terms of the roles of participants in the reengineering process [1, 5, 12]. In addition, how current methodologies for object-oriented software development can be streamlined and combined with ones for service-oriented software development has not been explored.

Service-Oriented Architecture (SOA), a key concept of the SoSR methodology, is conceptualized in terms of a three-service-participants model: service consumer, service broker, and service provider. Based upon the three-service-participants model, the reengineering of a legacy system to a target system is defined in terms of the tasks of the three participants. The tasks of each participant are modeled using the 4+1 view model that has been broadly adopted in object-oriented modeling [8]. The 4+1 view for each participant can be further defined in terms of sub-tasks and sub-roles by using the "Responsible, Accountable, Consulted, Keep Informed" (RACI) chart [14, 16]. Also, the 4+1 view for each participant is modeled in Unified Modeling language (UML) [2]. To demonstrate application of the SoSR methodology in modernization of a legacy system, a retail business information system is reengineered. Analysis of the results determine that this methodology can help software developers and system integrators reengineer current tightly coupled legacy information systems into the loosely coupled, agile, and service-oriented information systems.

2. Previous Efforts

The term, "modernization," was used for software reengineering of a legacy system. Software reengineering in this case consists of reverse engineering and forward engineering [15, 12]. Seacord and et al. introduce a risk-managed modernization approach for modernizing a legacy system [12]. The approach introduces general procedures for software reengineering at a very high level: 1) A candidate legacy system to be modernized is selected. 2) The stakeholders for this modernization are identified. 3) The requirements for modernizing the legacy system are determined. 4) Based on the identified stakeholders and requirements business cases are created. 5) The business cases are reviewed. If the created business cases satisfy business analysts, then the modernization plan continues in the next step. If the business cases do not satisfy business analysts, the modernization is terminated. 6) Two procedures for understanding both legacy system and target technology are conducted in parallel. 7) Evaluation of the technology is performed. 8) The evaluation is used to define a target architecture and a modernization strategy is consequently defined. 9) The defined modernization strategy is reconciled with the needs of the stakeholders. 10) All the procedures are completed and then an estimate of the resources needed is performed. 11) If the strategy is not feasible, this procedure revisits either evaluation of technology and the target architecture or the modernization strategy. If the strategy is feasible, the modernization plan is defined.

However, the risk-managed modernization approach alone is not enough for modernizing a given legacy system into a service-oriented system(s). It lacks providing an explanation of how the software reengineering process is conducted by whom and with what tasks. In addition, the approach does not consider how the use of Web services affects the modernization.

Bieberstein and et al. introduced the issue of roles and tasks for developing a system using SOA [1]. They identified new roles added due to SOA in addition to existing roles. The existing roles are: Information Technology project manager, business analyst, architect, developer, security specialist, system and database administrator, service deployer, service integration tester, toolsmith, and knowledge transfer facilitator. The new roles are: SOA architect, service modeler or designer, process flow designer, service developer, integration specialist, interoperability tester, UDDI administrator, UDDI designer, services governor, SOA project manager, and SOA system administrator. Since the tasks may overlap between roles, the tasks cannot always be assigned to specific roles. Also, they do not specify different roles in terms of three service participants: service consumer, service broker, and service provider. In addition, tasks are not specified in terms of modeling 4+1 views.

Erl [5] suggests 30 best practices for integrating Web services based on planning service-oriented projects, standardizing Web services, designing service-oriented environments, managing service-oriented development projects, and implementing Web services. Best practices for planning service-oriented projects include: knowing when to use Web services, knowing how to use Web services, knowing when to avoid Web services, moving forward with a transaction architecture, leveraging the legacy, building on what you have, understanding the limitation of a legacy foundation, budgeting for the range of expenses that follow Web services into an enterprise, aligning Return on Investments (ROI) with migration strategies, and building toward a future state. Although they describe three roles - service broker, service provider, and service requestor - for developing and integrating an information system with Web services, the best practices presented are not specific in terms of these three roles. No visual models are provided, either.

3. SoSR Methodology

3.1. Fundamental Concepts

Several key concepts are employed in the SoSR methodology: 3-layered architecture, n-tier architecture (distributed processes), SOA, ROC, business process design and execution, role-based model, 4+1 view model, and RACI chart.

The 3-layered architecture is an architectural pattern for a software system consisting of three logical layers: presentation layer (1st layer), business logic layer (2nd layer), and data access layer (3rd layer). Each n-th layer depends on the (n-1)-th layer only. The 3-layered architecture helps software developers design a software system by allowing the developers to focus on designing a specific layer.

The n-tier architecture is an architectural pattern in which a software system consists of n number of distributed processes. For example, a typical 4-tier architecture for a web application is as follows: A process for a Web browser that handles client-side presentation is located at the first tier. A process for a Web server that handles server-side presentation is located at the second tier. An application server process for handling business logics is located at the third tier. And, the fourth tier is a database server for processing data accesses. This n-tier architecture helps a software developer assign and distribute a set of related software components at the proper tier. Since each tier consists of components with the same purpose, deployment and maintenance of components are more effective.

SOA is an architectural pattern in which a service is published to a service registry and the published service is discovered and bound to a service client when the client requests the service. Because of that, the SOA has been described as the “publish-discovery-binding” model. If Web services technologies are used, an interface of the software component is represented in Web Services Description Language (WSDL). Interface related information such as the location of a WSDL described service, business entity, and etc. is published onto a UDDI service registry. Requested services can be searched for and discovered in the UDDI service registry. The service client is bound to the service through an interaction protocol, Simple Object Access Protocol (SOAP).

The power of SOC comes from the recursive definition of a service. A set of services can be composed to form another service called a composite service that represents a designated business process. Currently, a business process using Web services can be described in Business Process Execution Language (BPEL) and executed by a business process engine. The representation of a business process and its execution reduces the information technology gap between business and software professionals.

In SOC, three participant roles can be identified: Service Consumer (SC), Service Broker (SB), and Service Provider (SP). A service consumer discovers a required service and invokes the discovered service either directly or indirectly through a service broker. A service provider develops a service in a programming language and publishes its service specification to a service registry. A service broker can provide access to a service or compose a set of services as a new service and administer the registry. Currently, these three roles are manually implemented. With the development of new engines using semantic Web services for service publishing, discovery, and composition, these roles can be automatically performed in the near future.

Modeling software systems has been broadly accepted for the design and implementation of object-oriented applications. Several design methodologies have been proposed, and some of them have been incorporated into Computer-Aided Software Engineering (CASE) tools. Among them, OMG’s UML [2] and IBM’s Rational CASE tool and Rational Unified Process (RUP) [9] have received strong industry attention. The RUP methodology adopted Kruchten’s 4+1 view model [8] to support object-oriented application development.

The 4+1 view model, which is shown in Figure 1, describes the architecture of software-intensive systems in terms of 4 views (Design View, Implementation View, Process View, and Deployment View) sharing a Use Case View. These multiple views describe logical/physical and static/dynamic properties of a system. A use case view is used to represent use case scenarios such as what user requirements are and who use the system. A design view shows class hierarchy, the message passing among objects, and algorithms for methods. An implementation view describes the dependency of physical files. A process view describes a set of activities and their flows. A deployment view shows the location of executable components.

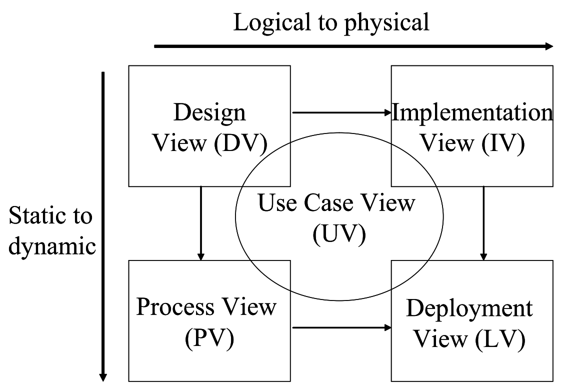

Figure 1. The 4+1 View

The 4+1 view indicates that design and process views are relatively logical, compared to implementation and deployment views. Likewise, design and implementation views are relatively static, compared to process and deployment views. When Kruchten's 4+1 view model was proposed, the 'software as services' concept had not been developed. With the advent of Web services concepts and technologies, there is a strong demand for models to develop and integrate service-oriented software intensive system.

In any project, listing all of the tasks for each role is a very important step. The RACI chart accomplishes this purpose [14, 16]. The RACI chart is used to depict the four types of responsibility: responsible (the role who owns the problem), accountable (the role of who must approve on work before it is effective), consulted (the role who provide input to help perform the task), and keep informed (people with a vested interest who should be kept informed). Table 1 is an example of RACI chart from [11].

Table 1. A Typical RACI Chart [11]

Tasks	AR	SA	DE	TE	SP
Security Policies		R		I	A
Threat Modeling	A		I	I	R
Security Design Principles	A	I	I		C
Security Architecture	A	C			R
…	…	…	…	…	…
Hosts Security	C	A	I		R
Application Security	C	I	A		R
Deployment Review	C	R	I	I	A

AR: Architect, SA: System Administrator, DE: Developer, TE: Tester, and SP: Security Professional

The RACI chart is analyzed vertically and horizontally. RACI approach outlines key questions to analyze. For example, when the chart depicts a lot of Responsibility (R) on a particular role, this may be an indication of too much responsibility. When too many Accountable's (A) are in a column (a role), the role can become a bottleneck. Horizontal analysis is on a particular task. If there are no R's in a row, then the task may not be completed since no Responsibility role exists. If there are too many A's, then there can be a lot of confusion since there is no single designated decision maker.

3.2. Reverse and Service-Oriented Forward Software Engineering

The SoSR methodology has two main processes: reverse software engineering and service-oriented forward software engineering [3, 4], see Figure 2. The reverse software reengineering process of a legacy system to a target system begins with determining the modernization requirements for a target legacy system based on user input. The requirements specify what features of a legacy system will be modernized into a target system(s) using the SOC paradigm. In the reverse software engineering process, a visual model for the legacy system is constructed by analyzing the given legacy system and the modernization requirements. These are then used to generate a target system through a service-oriented forward software engineering process (SoFSE).

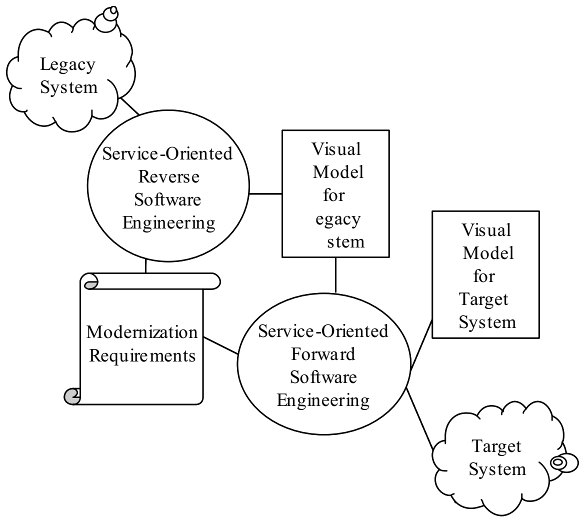
Legacy System
Service-Oriented Reverse Software Engineering
Visual Model for egacy stem
Visual Model for Target System
Modernization Requirements
Service-Oriented Forward Software Engineering
Target System

Figure 2. Service-Oriented Software Reengineering

We first examine how the three-service-participants model is related to the 3-layered architecture. A service consumer is interested in implementation of user interfaces for presentation logic and invocation of business logic. A service broker is interested in composition of business processes and creation of an interface so that a service consumer can invoke the service identified. A service provider is interested in creating components to handle the required functionality.

Based upon the three-service-participants model, reengineering of the legacy system to the target system is defined in terms of the 4+1 view model. The 4+1 view for each participant is further developed in terms of tasks and different roles by using RACI charts. In addition, the 4+1 view for each participant is modeled in UML.

Since we assume that no SOA was employed for the development of the legacy system, a reverse software engineering needs to be applied to the legacy system. No service stakeholder type exists. Five RACI charts are proposed for the 4+1 view in terms of a main role of a software developer who has several sub-roles and tasks such as analysis, design, implementation, testing, and deployment, etc. In Figure 3, a cell shows a Reverse Software Engineering (RSE) vector, RSE[i] for one of the five views. Its content is a RACI chart and a visual model.

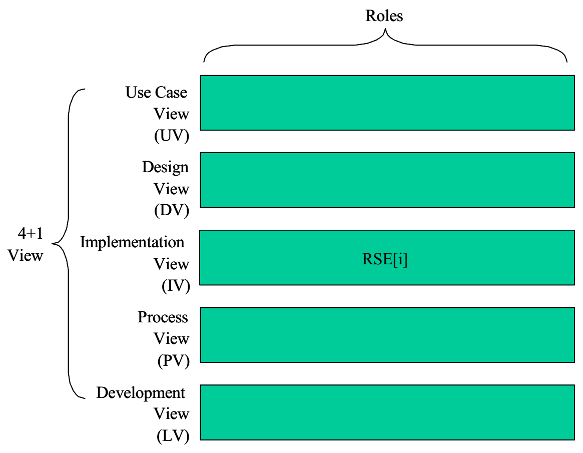

Figure 3. RSE[j] Vector, where j ∈ {UV, DV, IV, PV, LV}

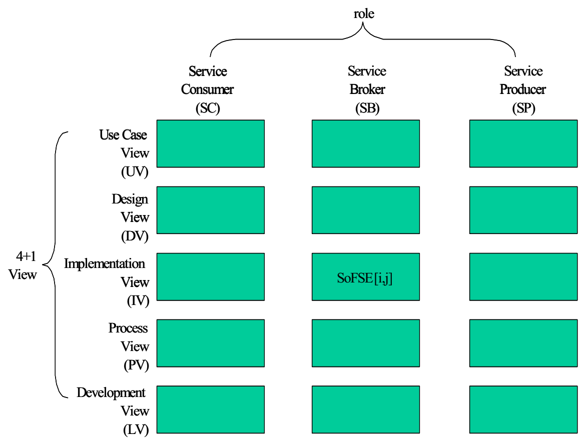

Figure 4. SoFSE[i, j] Matrix, where i ∈ {SC, SB, SP} and j ∈ {UV, DV, IV, PV, LV}

Based upon the RACI charts, a software reverse engineering process is conducted. As a result of the reverse engineering, the legacy system is documented as a visual model. The visual model for the legacy system is then used in the next process – service-oriented forward software engineering (SoFSE).

In the SoFSE process, three roles of the project participants are used. For each role, five RACI charts are described for the 4+1view. Thus, a total fifteen RACI charts are generated and employed for the SoFSE process. As the results of the SoSFE, the target system is documented in the form of a visual model. Figure 4 depicts the possible combinations of service stakeholder type and each view has an RACI chart and a visual model.

3.3. Properties

The SoSR methodology is a set of best practices that is architecture-centric, service-oriented, role-specific, and model-driven. First of all, the SoSR methodology is architecture-centric since three architectures are employed: 3-layered architecture for design, n-tier architecture for deployment, and SOA for integration. Secondly, the SoSR methodology is service-oriented since Web services are employed as the main building block for integration and a set of Web services, a composite service, is used to represent a business process to be executed on a business process engine. Thirdly, the SoSR methodology is role-specific. A service-oriented architecture is conceptualized by using three-service-participants model. Since a RACI chart uses tasks and roles to analyze a system design, it is role-specific. Lastly, the SoSR methodology is model-driven since it produces visual models for each service participant and model view.

4. Legacy System

As an example case, the GMPO (Gran Mercado at Port Orchard) Retail Store Inventory System is introduced. It was developed from October 2002 to July 2003 by a developer for a small retail/wholesale store. The GMPO legacy system consists of three components for three departments: inventory, purchasing, and sales. The GMPO system is designed to support daily Point Of Sale (POS) at a small retail store. Based on daily transactions and updated inventory, a purchasing manager generates a list of orders and manually submits a hard copy of purchasing orders to suppliers. This project is focused on the purchasing department. The purchasing department decides which items and how many items need to be ordered based on current inventory and market research for new items. The Java Server Page (JSP) technology is currently used to implement business processes at the server side.

There are several steps in generating a purchase order. The purchasing manager selects a supplier. The purchasing manager then fills the purchase order form out. The order number can be automatically generated by system or can be entered by the manager. For items that are ordered frequently, the item information can be automatically filled in once the item number is entered. Some fields such as the subtotal of the order are derived by the system. An interface displays selected items for purchasing.

However, the purchasing manager at the retail store wants to automate the process of sending the purchase order to suppliers so that suppliers can prepare the shipment sooner. Since the current systems for both retailers and suppliers are tightly coupled to software components, it is very difficult for a retailer to send purchase orders to suppliers electronically.

Enabling an organization's Business Information System (BIS) interaction with another organization's BIS can improve the whole supplier chain. For example, a supplier can improve on-time delivery when the purchase order of a retailer directly accesses the supplier's database.

5. Target System

5.1 Retailer's SoBIS

A Service-Oriented Business Information System (SoBIS) for GMPO retail needs to provide support for all existing business processes and enhance existing/re-occurring business components for reuse and sharing. Also, a retailer needs to add functionality for transmitting a purchase order to a supplier. The interface used to display a purchase order is the same as that of the legacy system.

In the retail legacy system, the major functionalities are tightly fixed to a platform's infrastructure. However, in the retailer's SoBIS, the Web services that contain the business processes can be performed on any platform since Web services are loosely coupled. For instance, a Web service called 'get_purchasing_list' accesses two database tables, returns an order form, and executes another process that sends the information to supplier(s). A JSP file representing a presentation layer calls Web service methods that are used to achieve server side business process functionality.

5.2. Supplier's SoBIS

The design of a supplier's system is very similar to that of the retailer's one. A retailer's system sends a purchase order list to a supplier's one and the supplier's system saves the purchase order list to a database as a requested purchase list. The database at the supplier side needs to add tables to handle purchasing request from retailers. Purchasing order data consist of header information containing purchasing request number, requester's information, order total, and line items specifying item number, description, estimated price, amount, etc. A Web service at the supplier side stores the XML-based purchasing list to the supplier's database. A supplier then retrieves the requests and takes an action to generate a shipment for delivery to the retailer.

5.3. Integrating Retailer's and Supplier SoBISs

A purchasing manager at the retailer side initiates a Purchase Order (PO) request and a sales manager at the supplier side replies to the request via Web services. The purchasing manager sends a PO to the supplier and a Web service stores the PO information to the supplier's database. Then the supplier sales manager at cooperates with a supplier inventory manager to decide how the order can be filled from existing inventory. When the check is done, the sales manager provides the retailer's purchasing manager with information regarding availability of the requested items. The purchasing manager then uses the confirmed information and authorizes delivery of the items.

Oracle BPEL Designer was used to create a composite Web service with access to two databases: at the retailer side and at the supplier side. An Oracle BPEL Project Manager (PM) server provides the interface to test the Web service unit. A composite Web service called 'PurchasingProcess' resides on the BPEL PM server not on the supplier's SoSR BIS. If a specific purchase invoice id is submitted from the retailer, the composite Web service returns a message back to acknowledge that the process succeeded. The Web service at the supplier side sends a message "good" to let the Web service at the retailer side know that the inserting process for purchasing list is completed successfully. A 'SaveWholeSaleOrder' Web service supports the supplier to restore the purchase request from the retailer.

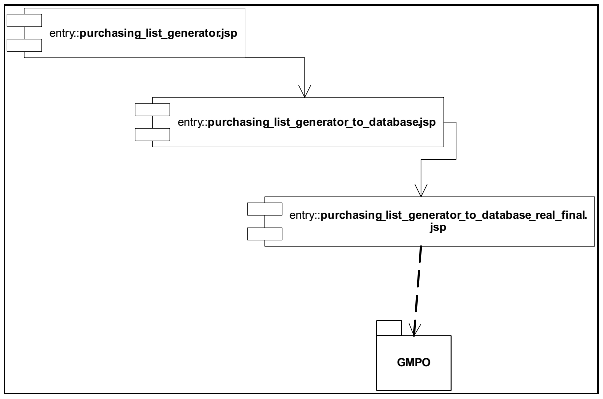
entry::purchasing_list_generator.jsp
entry::purchasing_list_generator_to_database.jsp
entry::purchasing_list_generator_to_database_real_final.jsp
GMPO

Figure 5. An Implementation View (IV) of the Legacy System Saving Order List Process: RSE[IV]

6. Analysis of GMPO's SoSR Model

6.1. Legacy System

A Web interface communicates with business functions at the server side and a business process stores user's decisions to a server side database by having connections to data interface. In Figure 5, we show a component diagram of an implementation view of the ordering process. A purchase order from an end user, generated by a JSP page 'purchasing_list_generator.jsp,' is sent to another JSP page 'purchasing_list_generator_to_database.jsp' to display order confirmation. When the order is confirmed, a purchasing list is saved to the retailer's database by another JSP page called 'purchasing_list _generator_to_database_real_final.jsp'.

In Table 2, the implementation view of the sub-tasks and sub-roles for the legacy system are described by using a RACI chart. A project manager is accountable for the tasks. Various roles are distributed to an architecture and design reviewer, to a component deployer, and to a resource scheduler. An architecture and design reviewer is responsible for analyzing dependency of components. A component deployer is responsible for managing uptime of components and registering local components to the local system. A resource scheduler is responsible for analyzing availability of components so that other roles can contact the resource scheduler and find out the availability of components.

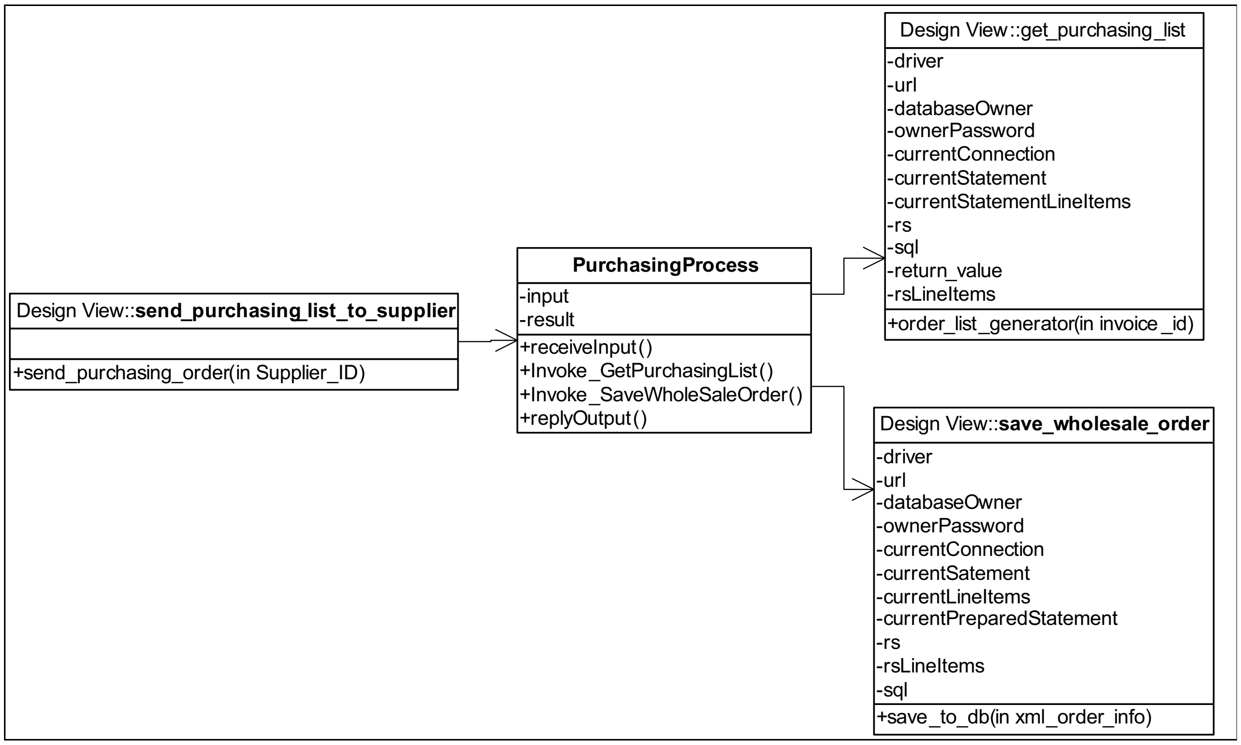
Design View::send_purchasing_list_to_supplier
+send_purchasing_order(in Supplier_ID)
PurchasingProcess
-input
-result
+receiveInput()
+Invoke_GetPurchasingList()
+Invoke_SaveWholeSaleOrder()
+replyOutput()
Design View::get_purchasing_list
-driver
-url
-databaseOwner
-ownerPassword
-currentConnection
-currentStatement
-currentStatementLineItems
-rs
-sql
-return_value
-rsLineItems
+order_list_generator(in invoice _id)
Design View::save_wholesale_order
-driver
-url
-databaseOwner
-ownerPassword
-currentConnection
-currentSatement
-currentLineItems
-currentPreparedStatement
-rs
-rsLineItems
-sql
+save_to_db(in xml_order_info)

Figure 6. A Design View (DV) of Sending a Purchasing List to a Supplier: SoFSE[i, j] where i ∈ {SC, SB, SP} and j = DV.

Table 2. A IV RACI Chart on the Legacy System

		BA	A& DR	PM	RS	CD	TE
IV	Analyze dependency of components	I	R	A	C		C
	Manage uptime of components	I	I	A	C	R	C
	Register local components to the local system			A		R	I
	Analyze availability of components	I	I	A	R	I	C

BA: Business Analyst, A&DR: Architect and Design Reviewer, PM: Project Manager, RS: Resource Scheduler, CD: Component Deployer, TE: Tester

6.2. Target SoSR BIS Systems

A design view of service provider, broker, and consumer is shown in Figure 6. Both 'get_purchasing_list' and 'save_wholesale_order' classes can be designed by one or two service providers. The 'PurchasingProcess' is a composite service that was composed by a service broker. A service consumer invokes another Web service 'send_purchasing _list_to_supplier'.

The service broker's role is used for illustration purposes: A service broker composes a composite service using a set of simple Web services. In this case, new roles such as BPEL designer and BPEL design reviewer are needed. Figure 7 is an activity diagram that depicts how a service broker chooses the right supplier based on consumer name (or ID) and supplier name (or ID). A consumer (retailer in this case) initiates sending a purchasing list to a service provider (supplier in this case). However, the consumer does not send this directly, but rather, sends the request to a service broker and the broker then forwards the consumer information to the supplier.

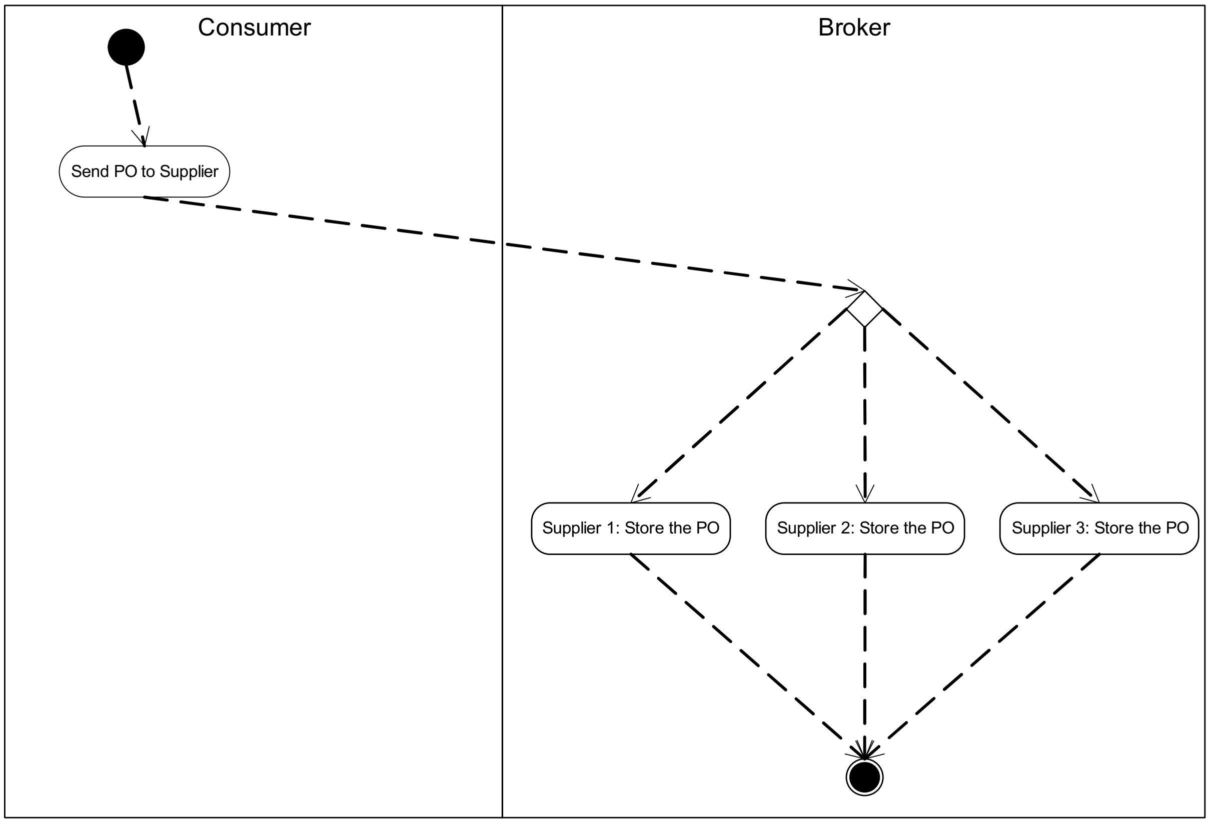

Figure 7. A Process View (PV) of Sending a PO to a Supplier Process: SoFSE[i, j] where i =SB and j = PV

Table 3. An RACI Chart at Process View for a Service Broker

		SB						
		Business Analyst	Project Manager	Resource Scheduler	Security Admin	BPEL Designer	Web Service Searcher	WS Tester
PV	Collaborate processes		A	C	I	R	I	C
	Specify activities at time	A	R					C
	Configure Input/Output parameters		A		I	I	R	C
	Define roles	A	R		I	I		

The design of this system assumes that there are multiple suppliers.

Table 3 shows RACI chart at process view for the service broker. For example, a project manager is accountable in many tasks. The project manager is responsible for specifying activities and the business analyst accountable for these tasks. A BPEL designer is responsible for business process collaboration. The business process collaboration needs to be consulted with a Web service designer.

7. Conclusions and Future Research

The SoSR methodology can help software developers and system integrators in organizing and conducting reengineering of tightly coupled, rigid, non-service-oriented legacy information systems into loosely coupled, service-oriented information systems. This SoSR methodology brings several benefits in the modernization of a legacy system into a target system based on using the SOC paradigm.

First of all, the scope of a project participant is very clearly defined. A person joining a software modernization project wants to know his or her scope in terms of roles, tasks, and views. Based on the SoSR methodology, participants of a service-oriented software modernization project can identify their scope in terms of the three service stakeholder types, service consumer, broker, and producer and one of the 4+1 views, use case, design, implementation, process, and deployment views. And then, the scope of a participant's roles and tasks are defined through a RACI chart. Also, since the diagrams in a visual model are related to the combination of a specific service stakeholder type and view, the participant knows which diagrams (visual models) need to be referenced or created within the selected scope.

Secondly, concrete artifacts are provided for the project participants. As soon as the participant knows his scope, a RACI chart is used to guide the reverse or forward software engineering and a visual model is used to visualize, specify, construct, and document a specific scope of the project participant. For example, if participant is a service provider and is interested in a process view, the participant has a role as service composer and a task to discover available services, compose them, and describes the service composition using a UML activity diagram.

Thirdly, this methodology provides agile guidelines. After the scope of a participant is chosen, the RACI chart can be adjusted to accommodate different project styles. The tasks and roles in a RACI chart for a specific scope can be added, omitted, or changed. The intersection cells between roles and tasks are used to ensure that all roles are well defined in terms of RACI: Responsible (R), Accountable (A), Consulted (C), and Informed (I). The assignment of the R, A, C, and I can be adjusted according to the changes of modernization environment.

Fourthly, the reverse software engineering process is streamlined with the forward software engineering. The same approach of using a RACI chart for a specific scope for the reverse and the forward software engineering can be used. In both the reverse and the forward software engineering, visual models are generated as one of artifacts of each software engineering.

Lastly, popular approaches that have been employed to develop better object-oriented software systems are naturally integrated into this SoSR methodology: SOA is used to describe the three different types of service stakeholders: service consumer, service broker, and service producer. A three-layered architecture is distributed over the three different types of service stakeholders: service consumer for presentation layer, service broker for business logic layer, and service producer for business logic and data access layers. A multi-tier architecture is used to depict the deployment view of each service stakeholder type. The 4+1 view has been used for model-driven and object-driven development. Especially, the inclusion of a business process engine for executing the composite services shows how SOC can affect future information system design, deployment, and integration.

However, there are not enough related examples for the SoSR methodology. The examples in this project are just a starting point to introduce a SoSR methodology in parallel with the advent of SOC. The SoSR knowledge base needs to grow by developing more examples of SoSR solutions, and proper responsibilities will be assigned to proper roles. A community of SoSR researchers can contribute to this.

8. References

[1] Bieberstein, N., Bose, Sanjay, Fiammante, M., Jones, K., Shah, R. (2005). Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap. IBM Press.

[2] Booch, G., Rumbaugh, J., Jacobson, I. (2005). Unified Modeling Language User Guide, (2nd Ed). Boston, MA: Addison-Wesley.

[3] Chikofsky, E. J., Cross, H. J, II. (1990). Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software Vol. 7, No. 1. Jan. 1990. pp. 13-17.

[4] Chung, S. and Lee, Y. S. (2000). Reverse software engineering with UML for Web site maintenance. The Proceedings of the First International Conference on Web Information Systems Engineering. Vol. 2. pp. 2157-2161.

[5] Erl, T. (2004). Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services. Upper Saddle River, NJ: Prentice Hall.

[6] Goepfert, J., Whalen M. (2002). An Evolutionary View of Software as a Service. IDC White paper.

[7] Highsmith, J. (2002). Agile Software Development Ecosystems. Boston, MA: Addison-Wesley.

[8] Kruchten, P. B. (1995). The 4+1 View Model of Architecture. IEEE Software, 12 (6), pp. 42 – 50.

[9] Kruchten, P. (2003). The Rational Unified Process: An Introduction, (3rd Ed.) Boston, MA: Addison-Wesley.

[10] Lea, D., Vinosk, S. (2003). Middleware for Web Services. IEEE Internet Computing, IEEE, 7 (1), 28 – 29.

[11] Meier, J. D., Mackman, A., Dunner, M., Vasireddy, S., Escamilla R., Murukan, A. (2003). Improving Web Application Security: Threats and Countermeasures. Microsoft Corporation.

[12] Seacord, R. C., Plakosh, D., Lewis, G. A. (2003). Modernizing Legacy Systems: Software Technologies, Engineering Process, and Business Practices. Boston, MA: Addison-Wesley.

[13] Turner, M., Budgen, D., Brereton, P. (2003). Turning Software into a Service. IEEE Computer. Vol. 36, No. 10. pp. 38-44.

[14] Value Based Management.net. RACI Model. http://www.valuebasedmanagement.net/methods_raci.html (accessed on May 19, 2006).

[15] Weerawarana, S., Vurbera, F., Leymann, F., Dtorey, T., Ferguson, D. (2005). Web Services Platform Architectures. Upper Saddle River, NJ: Prentice Hall.

[16] Wikipedia. RACI diagram.http://en.wikipedia.org/w/index.php?title=RACI_diagram&oldid=51658531 (accessed on May 19, 2006).