Service-Oriented Software Reengineering: SoSR

Joseph Byung Chul An
Computing & Software Systems
Institute of Technology
Univ. of Washington Tacoma
Tacoma, WA 98402
anba@u.washington.edu

Project Type:
Project: TCSS 702

Committee Names:
Committee Chair: Sam Chung, Ph.D.
Committee Member: Rogene Eichler West, Ph.D.

ABSTRACT

The purpose of this project is to propose a Service-Oriented Software Reengineering (SoSR) methodology that can be applied to a legacy software system so that it can be reengineered into a service-oriented software system(s) by using Service-Oriented Computing (SOC). SOC is a software development paradigm in which software systems can be recursively constructed as a set of services that employs standardized service interfaces and interaction protocols. This "software as a service" concept allows a software developer to design loosely coupled software components and integrate them with other software systems. However, since most components in a legacy system were not designed and developed as services, the current software systems need to be converted into a set of loosely coupled services. For this purpose, a methodology for service-oriented software reengineering is proposed for a developer who wants to apply SOC to a legacy system. The SoSR methodology is a set of best practices that is architecture-centric, service-oriented, role-specific, and model-driven. A service-oriented architecture is conceptualized by using three-service-participants model. Three service participants are following: service consumer, service provider, and service broker. Based upon the three-service-participants model, the reengineering process of the legacy system to the target system is explained in terms of the 4+1 view model that has been broadly adopted in object-oriented modeling. The 4+1 view consists of design, implementation, process, and deployment views sharing use case view. The 4+1 view for each participant is more deeply explained in terms of tasks and different roles by using "Responsible, Accountable, Consulted, Keep Informed" (RACI) chart. Also, the 4+1 view for each participant is visually modeled in Unified Modeling Language (UML). In order to demonstrate how the SoSR methodology can be applied to a modernization of a legacy system, business information systems in retail business are reengineered. This methodology can help software developers and system integrators to reengineer the tightly coupled legacy information systems to the loosely coupled, agile, and service-oriented information systems. Especially, the inclusion of a business process engine for executing composite services to existing application and database servers shows how SOC affect future information system design, deployment, and integration.

1. Introduction

Service-Oriented Computing (SOC) [3] 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) [8, 13]. SOC allows software systems to be recursively constructed as a set of services. Contrary to the current distributed middleware technologies such as RMI, CORBA, mobile agents, etc, SOC allows a system to be more naturally integrated by employing standardized service interfaces and interaction protocols. The specifications of SOC supporting technologies are being evolved and its implementation platforms such as Microsoft .NET are being improved. For example, retail stores use Point Of Sales (POS) systems to track transactions. The process of sending ordering information to a supplier may require a special Electronic Data Interchange (EDI) mechanism with a common external data representation to be marshaled and un-marshaled. Instead of using legacy EDI mechanism, the advent of SOC makes the integration of two heterogeneous systems easier by using standard interaction protocols, interfaces, and common external data representations in eXtensible Markup Language (XML).

Although this SOC paradigm allows a software developer to design loosely coupled software components and integrate them with other software systems, most components in a legacy system were not designed and developed as services. Therefore, the current software systems need to be converted into a set of loosely coupled services before integrating the legacy system with others. Current software development methodologies such as Rational Unified Process (RUP) [15], eXtreme Programming (XP) [14], Scrum, [18] etc. have focus on how to develop a software system, not how to modernize a software system for efficient integration. These methodologies do not specify tasks and responsibilities to convert the current legacy components to a set of loosely coupled services

The purpose of this project is to propose a Service-Oriented Software Reengineering (SoSR) methodology that is applied to a legacy software system for reengineering it into a service-oriented software system(s) by using SOC. Although there are some software engineering or reengineering methodologies [1, 7, 12, 16] with or without Web services, the methodologies do not specify tasks and responsibilities in terms of the roles of participants in the reengineering process. Also, it was not investigated how the current methodologies for object-oriented software development can be streamlined with ones for service-oriented software development.

In order to propose the SoSR methodology, a Service-Oriented Architecture (SOA) is conceptualized by using three-service-participants model: Service Consumer, Service Broker, and Service Provider. Based upon the three-service-participants model, the reengineering process of the legacy system to the target system is explained in terms of the roles of the three participants. The roles of each participant are modeled by using the 4+1 view model that has been broadly adopted in object-oriented modeling. The 4+1 view model, which was proposed by Kruchten in [9], consists of 4 views (design, implementation, process, and deployment views) sharing 1 view (use case view). The 4+1 view has been adopted in IBM's RUP that has been employed in a famous Computer Aided Software Engineering (CASE) tool, IBM's rational Rose [17]. The 4+1 view for each participant is more deeply explained in terms of tasks and sub-roles of each participant by using the "Responsible, Accountable, Consulted, Keep Informed" RACI chart [16]. Also, the 4+1 view for each participant is modeled in Unified Modeling language (UML) [4, 9].

In order to demonstrate how the SoSR methodology can be applied to a modernization of a legacy system, business information systems in retail business are reengineered. The analyzed results show that this methodology can help software developers and system integrators to reengineer the current tightly coupled legacy information systems to the loosely coupled, agile, and service-oriented information systems. Sub-roles under participants (Service Consumer, Service Broker, and Service Provider) have specified responsibility per task listed for 4+1 different views. And then based on defined responsibilities, models between views and participants explain the reengineered system.

This paper consists of the following five sections. In Section 2, previous efforts in software reengineering are discussed. In Section 3, main concepts that are applied to the formation of the SoSR methodology are explained and the SoSR methodology is proposed with its properties. In Section 4, a legacy system to be modernized is introduced. In Section 5, the SoSR methodology is applied to the modernization of business information systems in retail. In Section 6, the reengineered business information systems are analyzed in terms of the SoSR methodology. In Section 7, we conclude the benefits of the SoSR methodology with a discussion of future research efforts that may be needed.

2. Previous Efforts

A term, "modernization," was used for software reengineering on a legacy system. The modernization is classified into three categories according to [12]: white box modernization, black box modernization, and replacement. For white-box modernization, understanding internal system deeply is required. Black box modernization, in other hand, needs software reengineers to examine inputs and outputs of a legacy system. White box modernization is harder to proceed than black-box modernization since analyzing the source code is more difficult than analyzing what it does. When either white box or black box modernization is not cost effective instead of building a new system from scratch, replacement is highly considered.

For modernizing a legacy system, Robert Seacord and et al. introduce a risk-managed modernization approach [12] in Figure 1. The risk-managed modernization approach shows the general procedures for software reengineering at very high level: 1) A candidate to modernize is selected. 2) The stakeholders for this modernization are identified on a legacy system. 3) Requirements are understood to accomplish the goal of modernization on the legacy system. 4) Business cases are created based on the identified stakeholders and requirements. 5) And then the created business cases are reviewed. If the created business cases satisfy business analysts, then the modernization plan continues to be defined. However, when the business cases do not satisfy business analysts, the modernization is terminated. 6) When the business cases are satisfactory, two procedures for both understanding legacy system and understanding target technology are processed in parallel. 7) And then, evaluation of the technology is conducted. 8) Based upon the evaluation, the target architecture is defined. Consequently, modernization strategy is defined. 9) The defined modernization strategy is reconciled with the needs of the stakeholders. 10) All the procedures are done, and the estimation of the resources needed is performed. 11) When the strategy is not feasible, this procedure revisits either evaluation of technology and target architecture, or the modernization strategy. When the strategy is feasible, defining modernization plan is completed.

However, the risk-managed modernization approach is not enough to modernize the given legacy system into a service-oriented system. It lacks an explanation of how software reengineering can be done by whom with what tasks. Since SOC was not well founded when the authors published the book, it was not considered how the use of services affects the modernization approach. Also, any visual model was not considered as one of the artifacts of the approach.

Norbert Bieberstein et al. introduce roles and tasks for developing a system using Service-Oriented Architecture (SOA) [1]. The authors identify existing roles and new roles that are added due to SOA. The existing roles are defined as Information Technology (IT) project manager, business analyst, architect, developer, security specialist, system and database administrator, service deployer, service integration tester, toolsmith, knowledge transfer facilitator, SOA project manager, and SOA system administrator. The new roles are defined as SOA architect, service modeler or designer, process flow designer, service developer, integration specialist, interoperability tester, UDDI administrator, UDDI designer, and services governor. The authors point out that the tasks may overlap between roles; thus, not all the times tasks can be assigned to specific roles. However, the authors do not specify different roles in three

participants: Service Consumer, Service Broker, and Service Provider. Also, tasks are not specified in modeling views such as 4+1 views.

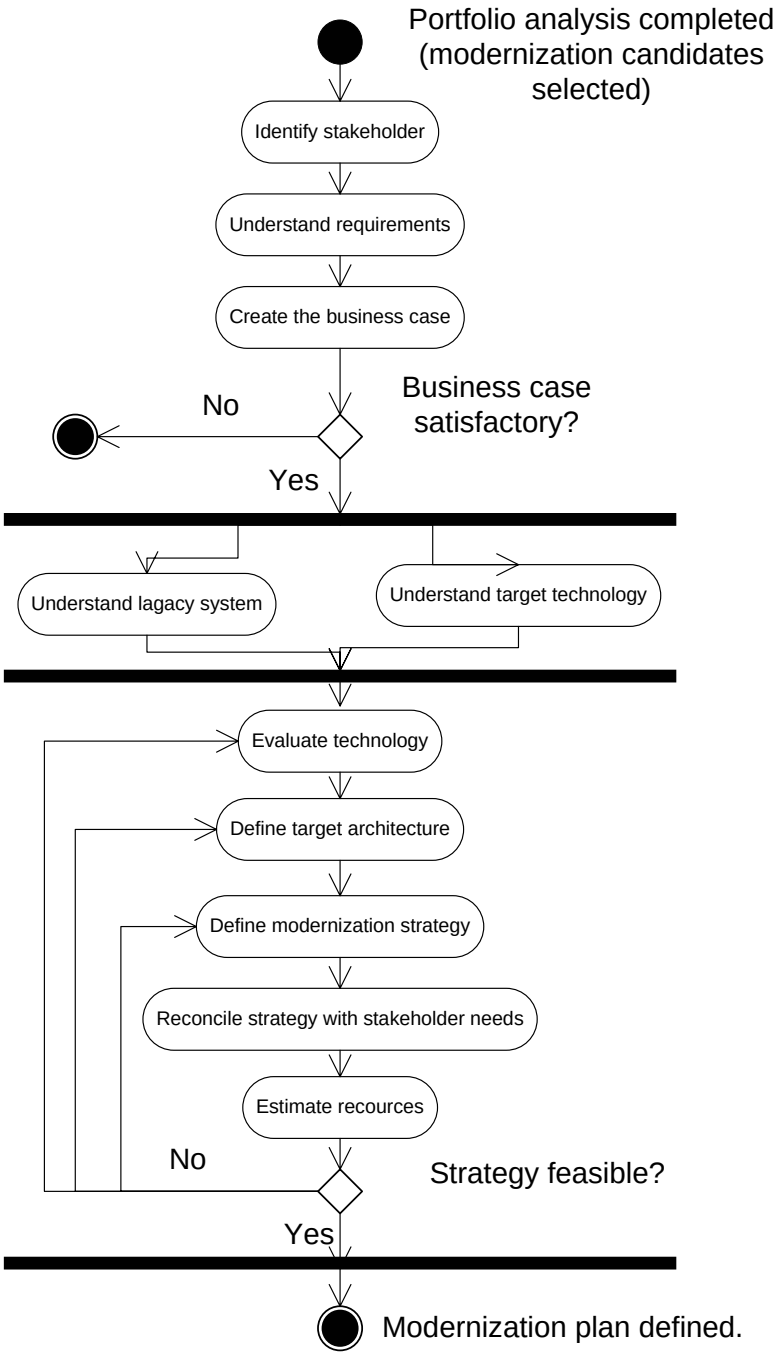


Figure 1. Risk-Management Modernization Guideline

Thomas Erl suggests 30 best practices for integrating Web services such as the best practices for planning service-oriented projects, the best practices for standardizing Web services, the best

practices for designing service-oriented environments, the best practices for managing service-oriented development projects, and the best practices for implementing Web services [7]. For example, some of the best practices for planning service-oriented projects are as follows; know when to use Web services, know how to use Web services, know when to avoid Web services, moving forward with a transaction architecture, leverage the legacy, build on what you have, understand the limitation of a legacy foundation, budget for the range of expenses that follow Web services into an enterprise, align Return on Investments (ROI) with migration strategies, and build toward a future state. Also, the author describes three roles for developing and integrating an information system with Web services: service broker, service provider, and service requestor. However, the best practices are not specified in terms of three different roles. Also, no visual models are employed.

3. SoSR Methodology

To explain what the SoSR methodology is, we first introduce fundamental concepts used in the methodology. And then, the SoSR methodology with its properties is described.

3.1. Fundamental Concepts

The following main concepts are employed in the SoSR methodology: 3-layered architecture, n-tier architecture, service-oriented architecture, business process design and execution, role-based model, 4+1 view model, and RACI chart.

The 3-layer architecture is an architectural pattern in which a software system consists of three logical layers: presentation layer (3rd layer), business logic layer (2nd layer), and data access layer (1st layer). An n-th layer depends on the (n-1)-th layer only. For example, the third layer depends on the second layer only, and the second layer depends on the first layer only. This 3-layer architecture helps a software developer to design a software system by allowing the developer to focus on designing a specific layer. The 3-layered architecture is shown in Figure 2.

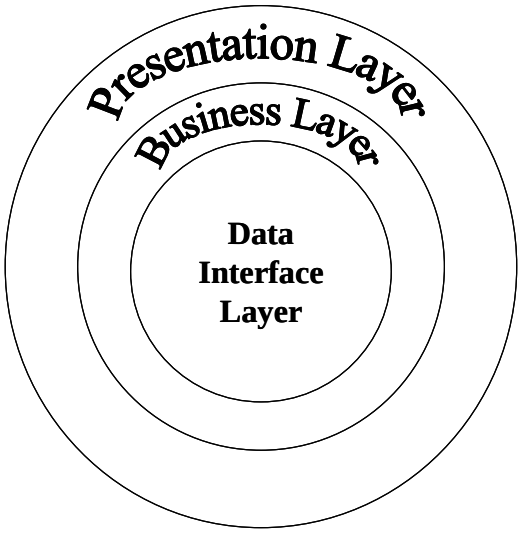


Figure 2. 3-Layered Architecture

The n-tier architecture is an architectural pattern in which a software system consists of n number of distributed processes. Figure 3 shows typical 4-tier architecture for a web application: A process for Web browser that handles client-side presentation is located at the first tier. A process for Web server and Web application 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 shows a database server for processing data accesses. This n-tier architecture helps a software developer to distribute a set of related software components at a proper tier. Since each tier consists of components with same purposes, both deployment and maintenance of components are more effective.

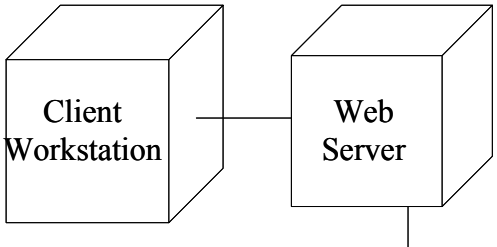


Figure 3. 4-Tier Architecture

The SOA is an architectural pattern in which a service is published to a service registry and the published service is discovered and is 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 in Figure 4. If Web services technologies are used, an interface of software component is represented in WSDL. The interface related information such as the location of WSDL, business entity, etc is published onto a UDDI service registry. If a service is requested, the service can be discovered at a UDDI service registry. The service client is bound to the service through an interaction protocol SOAP.

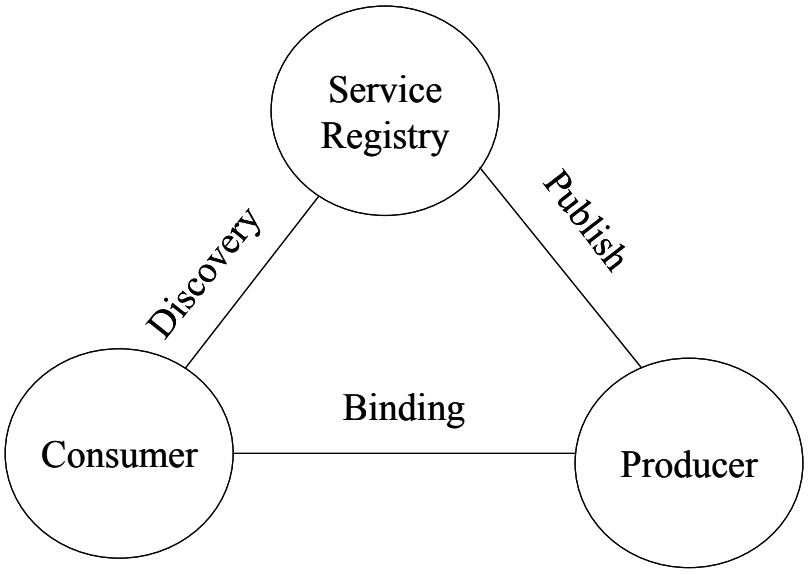


Figure 4. Service-Oriented Architecture

The power of SOC comes from the recursive definition of a service. A set of services is composed to be another service called a composite service that represents a business process. Currently, a business process using Web services is described in BPEL and is executed by a

business process engine. The representation of a business process and its execution reduces the IT gap between business and software professionals. Figure 5 shows a business process in BPEL.

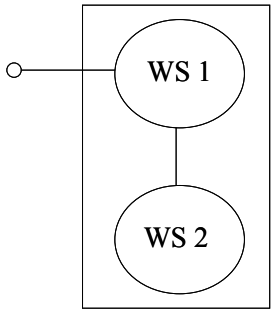


Figure 5. An Example of Visualized Business Process and Its representation in BPEL

By looking at SOC in terms of the roles of participants, the following three 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 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 composes a set of services as a new service and administers the registry. Figure 6 shows the three roles. These roles are currently manually implemented. Along with the development of new engines using semantic Web services for service publishing, discovery, and composition, these roles can be automatically done in the near future.

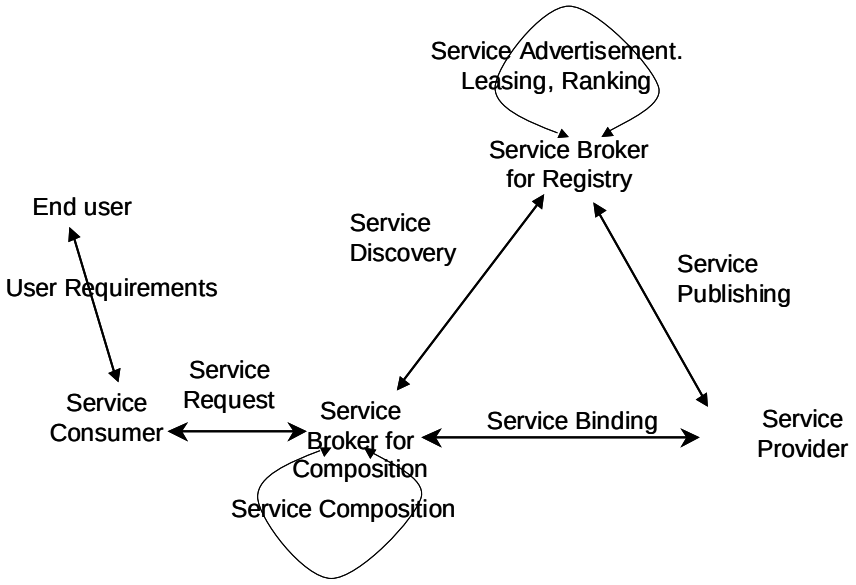


Figure 6. Three Participants in Service-Oriented Software Project: Service Consumer (SC), Service Broker (SB), and Service Producer/Provider (SP)

Modeling software systems has been broadly accepted in designing and implementing object-oriented applications in and relational databases. Several design methodologies have been proposed, and some of them have been incorporated with Computer-Aided Software Engineering (CASE) tools. Among them, OMG's Unified Modeling Language (UML) and IBM's Rational CASE tool and Rational Unified Process (RUP) have received strong industry attention. The RUP adopted Kruchten's 4+1 view model to support object-oriented application development. The 4+1 view model describes the architecture of software-intensive systems based upon multiple and concurrent views [23]. Kruchten's model was later incorporated into a famous CASE tool, IBM's Rational Rose Modeler. The 4+1 view model proposed for object-oriented computing was the dominant computing paradigm at that time. When Kruchten's 4+1 view model was proposed, the 'software as services' concept had not been available. Now, with the advent of web services concepts and technologies, there is a strong demand for models for developing and integrating service-oriented software intensive system. Figure 7 shows the 4+1 view model, in which a software system can be viewed in terms of logical/physical, and static/dynamic perspectives and the perspectives share a Use Case View (UV). The 4+1 view indicates that Design View (DV) and Process View (PV) are logical compared to Implementation View (IV) and Deployment View (LV). Like wise design view and implementation view are static compared to process view and deployment view.

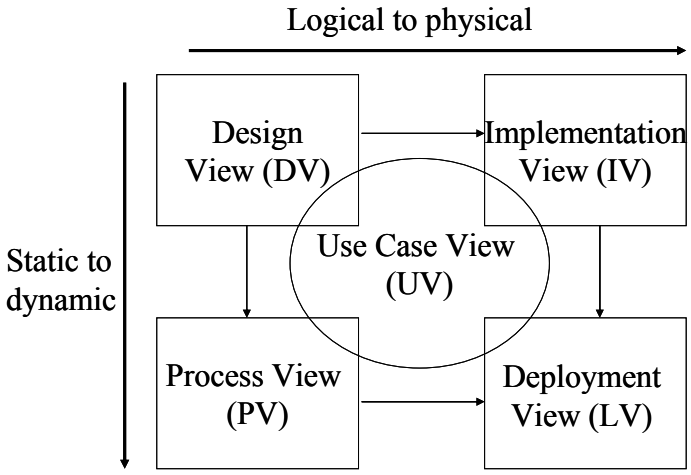


Figure 7. The 4+1 View

In any project, listing all of the tasks and human resources is a very important step. The RACI (Responsible, Accountable, Consulted, Keep Informed) chart is well used to accomplish the goal [24]. From the reference [22], it is quoted that "The guidance through the guide is task-based and modular, and each chapter relates to the various stages of the product development life cycle and the various roles involved. These roles include architects, developers, system administrators, and security professionals". The RACI chart explains four types of responsibilities: responsible (the role responsible for performing the task), accountable (the role with overall responsibility for the task), consulted (people who provide input to help perform the task, and keep informed (people with a vested interest who should be kept informed). Following chart (Table 1.) is an example of RACI chart from [22].

Table 1. A Typical RACI Chart [22]

Tasks	Architect	System Administrator	Developer	Tester	Security Professional
Security Policies		R		I	A
Threat Modeling	A		I	I	R
Security Design Principles	A	I	I		C
Security Architecture	A	C			R
Architecture and Design Review	R				A
Code Development			A		R
Technology Specific Threats			A		R
Code Review			R	I	A
Security Testing	C		I	A	C
Network Security	C	R			A
Hosts Security	C	A	I		R
Application Security	C	I	A		R
Deployment Review	C	R	I	I	A

The RACI chart is analyzed vertically and horizontally [26]. RACI Approach [26] outlines key questions to analyze. For example, when the chart has lots of Responsibility (R) on a particular role, then one may ask that one role is on top of so much. When too many Accountable (A) are in the vertical, this role can become a bottleneck. Horizontal analysis is on a particular task. When there are no R's then, the task may not complete since no responsible role exists. If there are too many A's, then there could be lots of confusion since there is no single designated decision maker.

3.2. The SoSR Methodology with Its Properties

The SoSR methodology consists of reverse software engineering and service-oriented forward software engineering, which is shown in Figure 8. The reengineering process of the legacy system to the target system starts with modernization requirements on a legacy system from a user. The requirements describe what features of a legacy system will be modernized to a target system(s) using the SOC paradigm. Through the reverse software engineering process, a visual model for the legacy system is constructed by analyzing the given legacy system and the modernization requirements. And then, based upon the model for the legacy system and the given modernization a requirement, a target system is generated through the service-oriented forward software engineering process.

Based upon the three-service-participants model, the reengineering process of the legacy system to the target system is explained in terms of the 4+1 view model that has been broadly adopted in object-oriented modeling: design, implementation, process, deployment views sharing use case view. The 4+1 view for each participant is more deeply explained in terms of tasks and different roles by using Responsible, Accountable, Consulted, Keep Informed (RACI) chart. Also, the 4+1 view for each participant is modeled in UML. In order to demonstrate how the SoSR methodology can be applied to a modernization of a legacy system, business information systems in retail business are reengineered. This methodology can help software developers and

system integrators to reengineer the tightly coupled legacy information systems to the loosely coupled, agile, and service-oriented information systems.

With the 3-layered architecture, service consumer is interested in implementation of user interfaces for presentation logic and invocation of business logic. Service broker is interested in composition of business processes and creating interface so that service consumer can invocate. Service Provider is interested in creating interface to handle database.

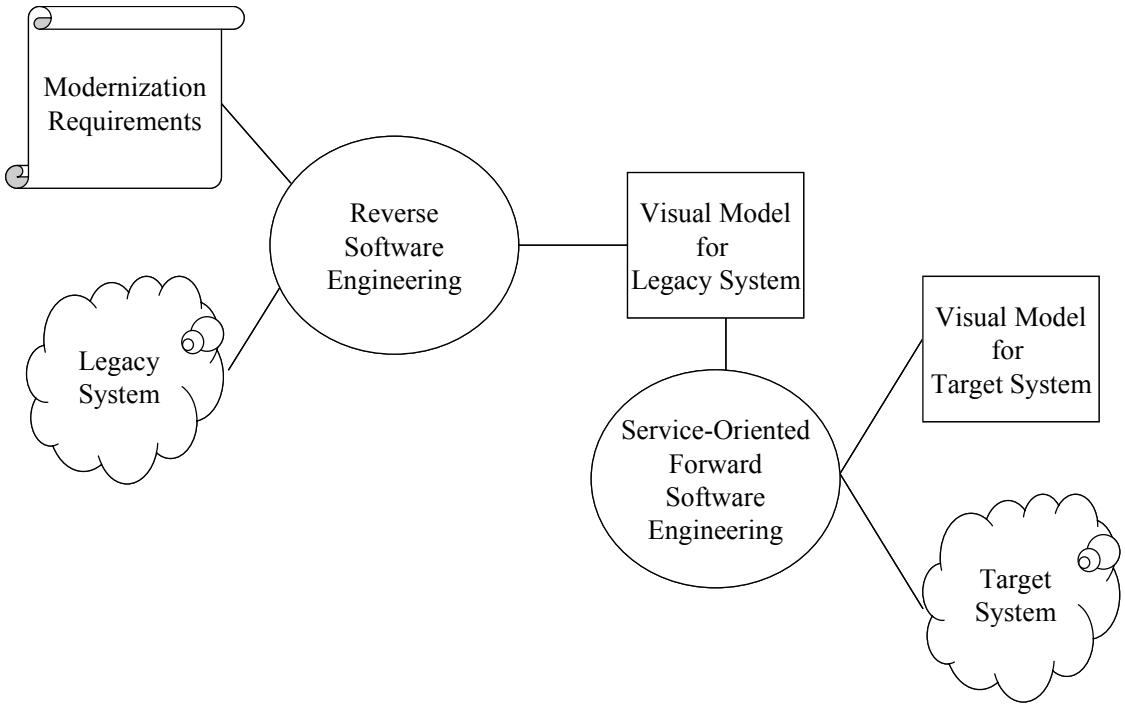


Figure 8. Service-Oriented Service Reengineering

The SoSR methodology consists of two sub-methodologies: reverse software engineering and service-oriented forward software engineering. Since we assume that no SOC was employed for the development of the legacy system, a reverse software engineering is 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 software developer who will conduct several sub-roles such as analysis, design, implementation, testing, and deployment, etc. In Figure 9, a cell shows Reverse Software Engineering (RSE) vector, RSE[i] for one of the five views and its content is a RACI chart and a visual model. Based upon the RACI charts, a software reverse engineering process is conducted. As the results of the reverse engineering, the legacy system is documented into a visual model. The visual model for the legacy system is used for next step – service-oriented forward software engineering.

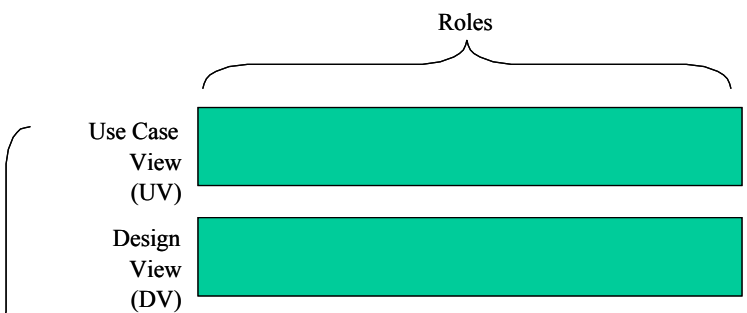


Figure 9. RSE[j] Vector, where j ϵ {UV, DV, IV, PV, LV}

At the Service-Oriented Forward Software Engineering (SoFSE) phase, three different roles such as service producer, broker, and consumer of the project participants are used. For each role, five RACI charts are described for the 4+1view. Total fifteen RACI charts are employed for this service-oriented forward software engineering. As the results of the service-oriented forward software engineering, the target system is documented into a visual mode. Figure 10 describes that each cell crossing each service stakeholder type and each view has an RACI chart and a visual model.

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 main building block for integration and a set of Web services, a composite service, to represent a business process is 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. Three service participants are following: service consumer, service provider, and service broker. Also, RACI chart is using tasks and roles to analyze a system design to support that SoSR methodology is role-specific.

Lastly, the SoSR methodology is model-driven. With n-tier architecture, deployment views are created for each service party. Also, SoSR is creating visual models for each service party and model view.

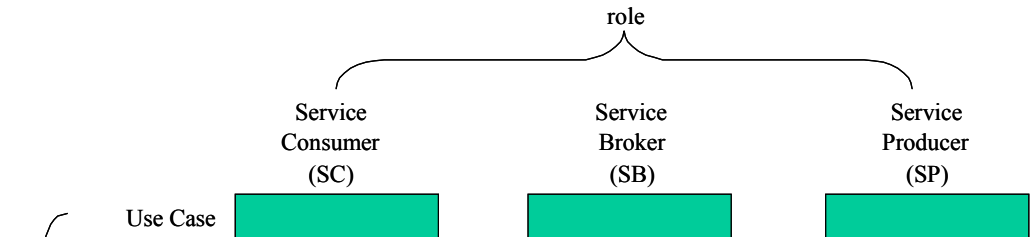


Figure 10. SoFSE[i, j] Matrix, where i ϵ {SC, SB, SP} and j ϵ {UV, DV, IV, PV, LV}

4. Legacy System

GMPO (Gran Mercado at Port Orchard) Retail Store Inventory System was developed from October 2002 to July 2003. GMPO system is designed to support daily Point Of Sale (POS) at a retail store. Based on daily transactions and updated inventory, a purchasing manager of GMPO generates a list of orders and uses the hard copies of purchasing orders to submit to supplier by hand.

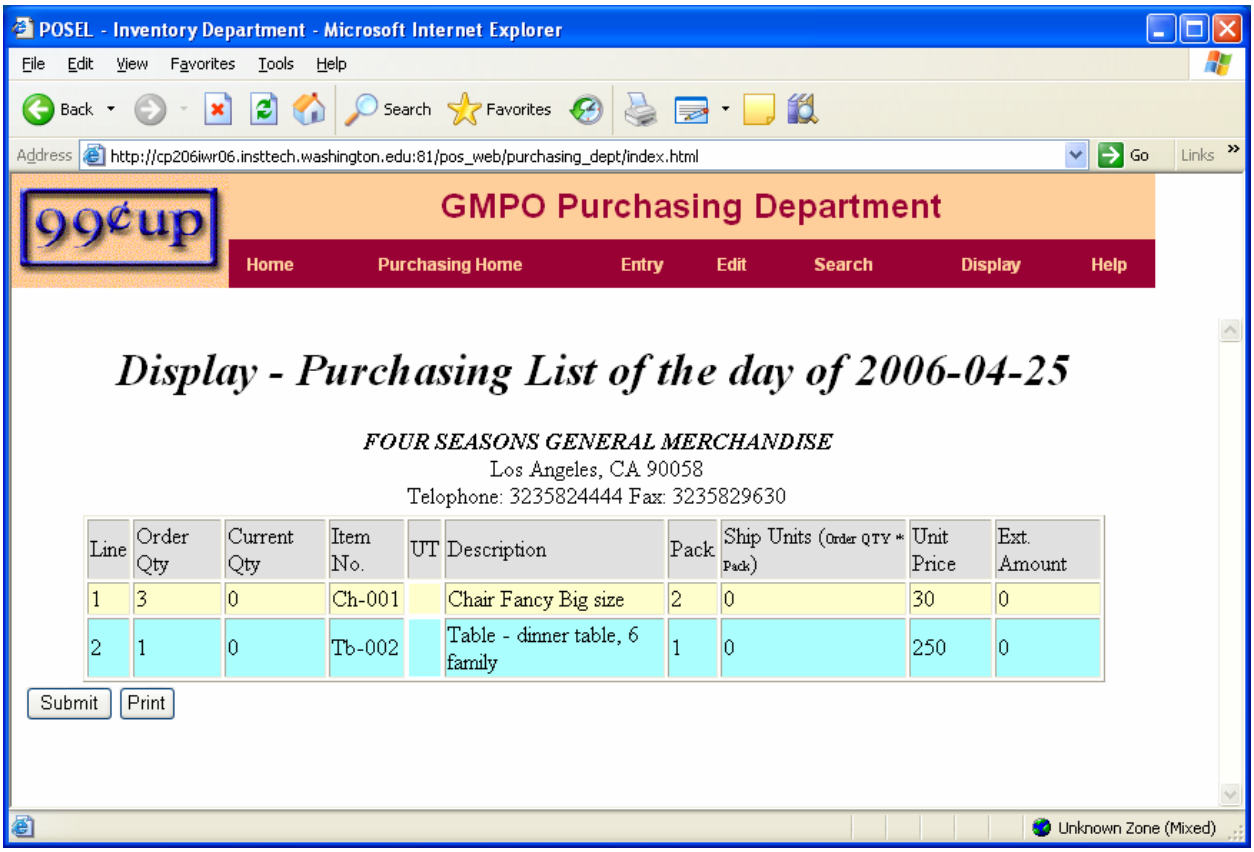


Figure 11. Report to Print for Ordering

Figure 11 is a simple accumulated ordering form to print out for ordering list. Java Server Page (JSP) is used to achieve business process at server side. The presentation interface, a business component, sends decision data to a business component in another server side to handle business processes such as validating the total of the estimated prices. And then the business layer sends validated and processed data for the system to another business component to commit the transaction by saving data to database.

The purchasing manager of GMPO returns with invoices and updates inventory by checking with real eyes while checking when the requested products arrive. An inventory manager is managing the whole process and the purchasing manager checks against product arrived based on the ordering list.

This process can be problematic; since a purchasing manager has to assume that suppliers always will have products retailers need and prices are always the same or are at lower price. Supplier cannot promise to support to deliver the products on time and/or on requested cost and quantity. When the purchase manager can access information such as availability and cost from suppliers, the purchasing process can increase productivity. When there are many suppliers for the same item, the purchase manager can compare the items at the GMPO's home site.

Suppliers can provide goods on time when the ordering list is requested to their database. Suppliers also can reduce cost to produce by consolidating items from other retailer when demands are predicted. Enabling Business Information System (BIS) with other business organization can improve the whole retail-supplier chain.

The GMPO legacy system consists of three departments: Inventory, Purchasing, and Sales. Sales occur at either at a cashier's desk or away from a cashier's desk due to business-to-business negotiation. This project is interested in the Purchasing department. Purchasing department decides how much is needed to buy to selling well and/or continuous selling of products and researching market to find new items. Figure 12 is an interface after GMPO's login interface on Retailer's SoSR BIS system.

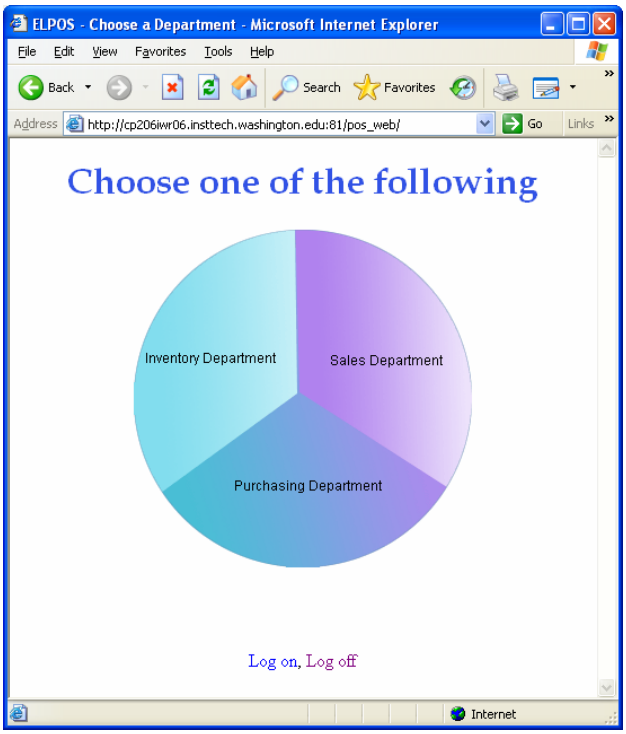


Figure 12. User Interface of After Login page of Retailer System

The purchasing manager specifies which supplier to work on to generate ordering list in Figure 13. Followed by the selection, the purchasing manager fills out the form. The order number can be automatically generated by system or can be typed in by the user. When the item is ordered frequently, the item info can be filled out by putting item number first.

Figure 13. Creating Purchase list – Choose supplier name

Figure 14 shows that the sub total of the ordering is generated on the fly. Figure 15 is an interface to display items for selected purchasing list. The purchasing manager at GMPO wants to automate the process to send purchasing list to suppliers so that suppliers can prepare the goods and the process of procurement can be smoother. In summary, the current system is tightly coupled and not agile. Thus, sending purchasing list to suppliers electronically is very difficult.

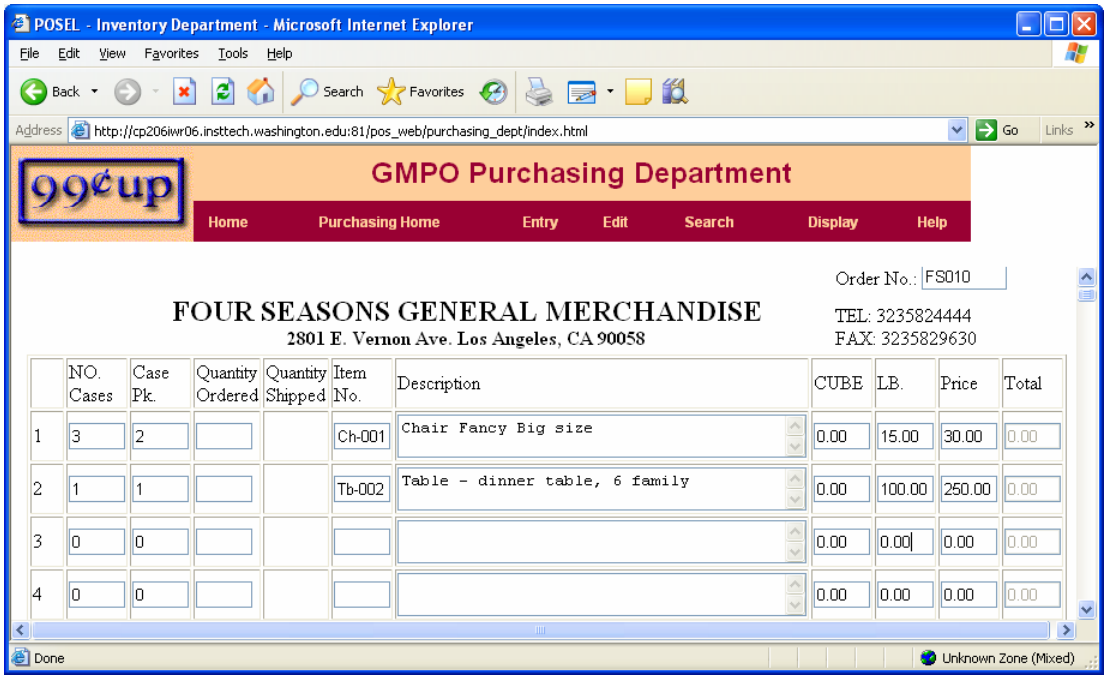


Figure 14. Creating Purchase list – Fill up order list

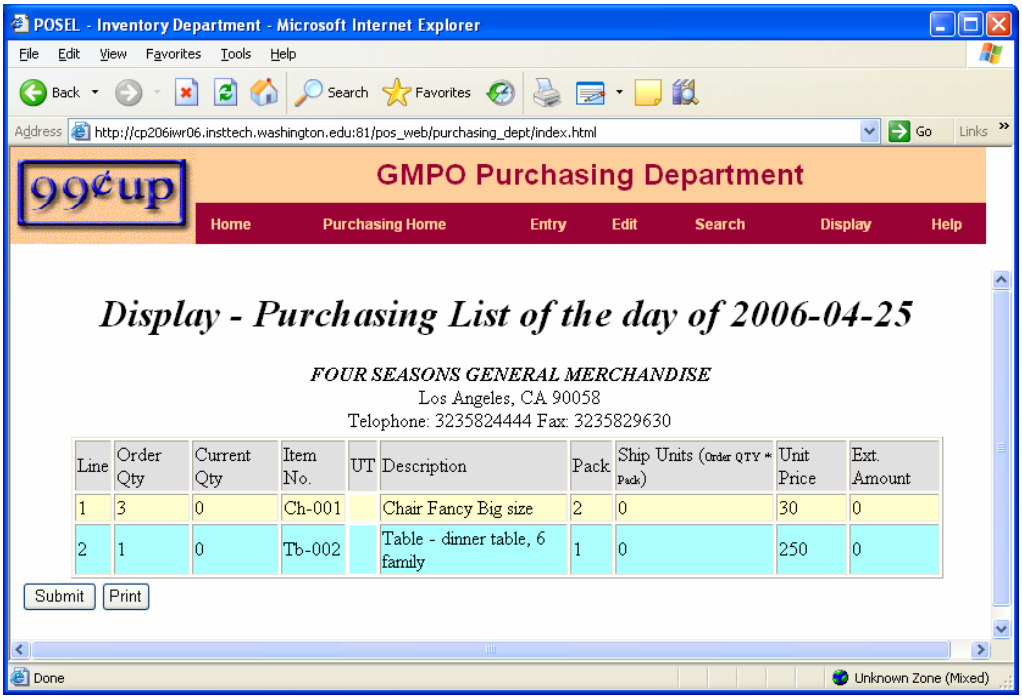


Figure 15. Purchase list

5. Target System

5.1 Retailer SoSR BIS

Retailer needed to keep all the existing business processes and enhance existing and re-occurring business components to ease of reuse and share. Also retailer needed to add a functionality to send a purchasing list to a supplier. This feature needed to complete with retailer-supplier SoSR BIS design due to agreements between the organizations. The interface to display purchase order list is the same as legacy as in Figure 15.

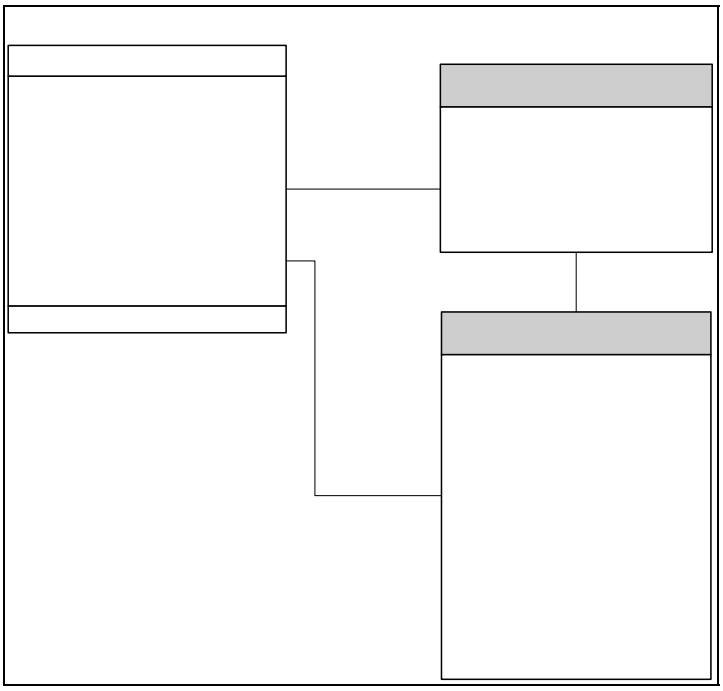


Figure 16. Design View of get_purchase_list class at Retailer's side

In the legacy system, the major functionalities need to be tightly coupled. However, in SoSR Retail BIS, the Web service that contains the business process can be any platform since Web services are loosely coupled. Figure 16, get_purchasing_list is a Web service that accesses two database tables and returns order form and other process that will send the information to supplier(s).

Figure 17 depicts a typical 3-layerd design with Web service. Web service is employed as middleware between business component and database [10].

get

-driver

url

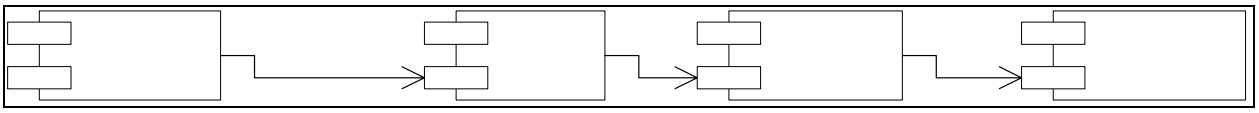


Figure 17. A Typical 3 Layered Design with Web Service

Figure 18 is an implementation view to explain ordering process in GMPO system with Web service. JSP and Web service are used to achieve business process at server side.

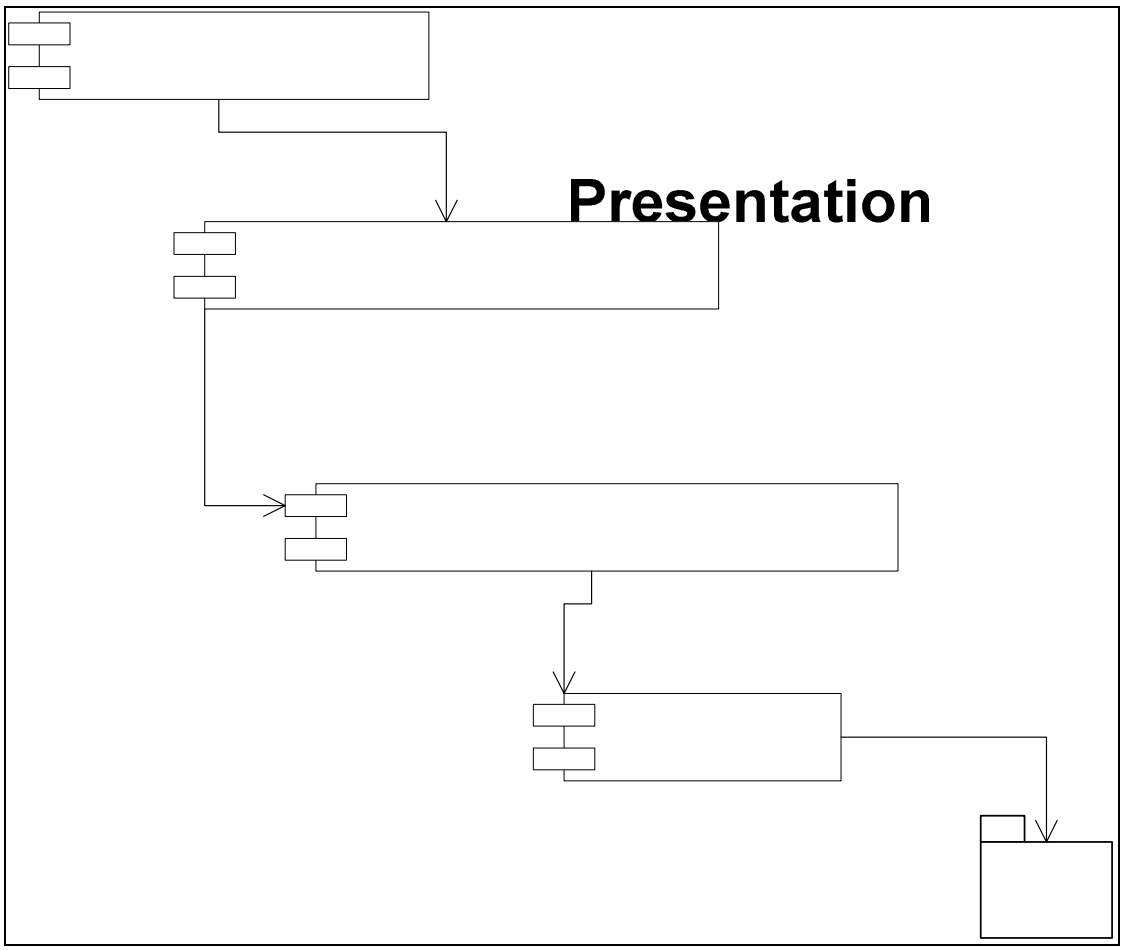


Figure 18. 3-Layered Design Applied to Saving Order List Process in Legacy System

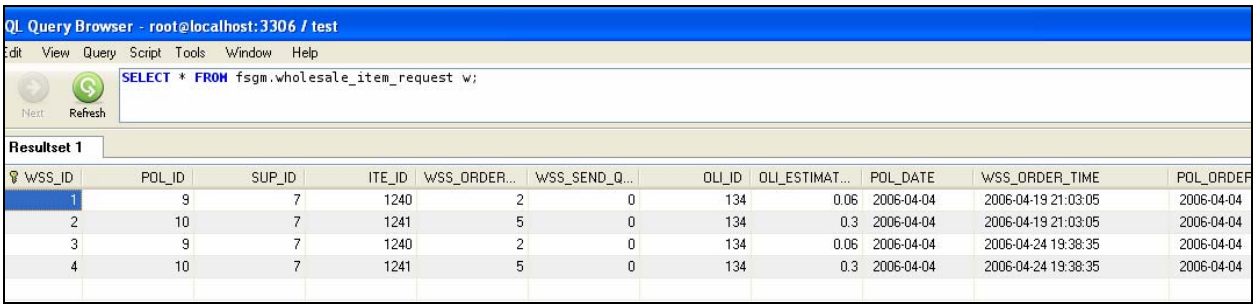
5.2. Supplier SoSR BIS

A system for Supplier was planed to develop. However the implementation did not happen for GMPO BIS legacy system. Thus, a system for Supplier is designed and developed in this project. Retailer sends a purchase order list to Supplier and Supplier saves the purchase order list to database as requested purchase list.

entry::purchasing_list_

The system design of Supplier is done very similar to Retailer. The database design of Supplier side was not done and design of database was necessary to set up the environment for SoSR project. The database at Supplier side needed to add two more tables to handle purchasing request from retailers. Purchasing requests consist of header information containing purchasing request number, requester information, and total amount to purchase and line items to specify item number, description, estimated price, and amount.

A Web service at Supplier side saves purchasing list in XML and unparsed to save to database. Figure 19 shows that the information went into Supplier's DB. Supplier will retrieve the requests and takes action to deliver the goods to Retailer.



WSS_ID	POL_ID	SUP_ID	ITE_ID	WSS_ORDER...	WSS_SEND_Q...	OLI_ID	OLI_ESTIMAT...	POL_DATE	WSS_ORDER_TIME	POL_ORDEF
1	9	7	1240	2	0	134	0.06	2006-04-04	2006-04-19 21:03:05	2006-04-04
2	10	7	1241	5	0	134	0.3	2006-04-04	2006-04-19 21:03:05	2006-04-04
3	9	7	1240	2	0	134	0.06	2006-04-04	2006-04-24 19:38:35	2006-04-04
4	10	7	1241	5	0	134	0.3	2006-04-04	2006-04-24 19:38:35	2006-04-04

Figure 19. Supplier's wholesale item list

5.3. Retailer-Supplier SoSR BIS

Retailer-Supplier SoSR BIS generally explained in use case diagrams. Figure 20 shows use case diagram to explain how the roles and the systems are integrated in conceptual view. Purchase Manager at Retailer side initiates the process and Sales Manager at Supplier side replies to the request via other Web services. Web services for replying part needs to be done. However, the use case diagram explains how two systems are related and what services are involved.

Purchasing manager at Retailer sends PO to Supplier and Retailer stores the received information to Supplier's database. And then Sales manager at Supplier cooperate with Inventory manager at Supplier to check their items in inventory to decide how the procurement can be done. Sales manager later, when the checks are done, corresponds to Purchasing manager at Retailer with detail information of availability of request items. Purchasing manager then will use the confirmed information and authorize to deliver the items to Retailer.

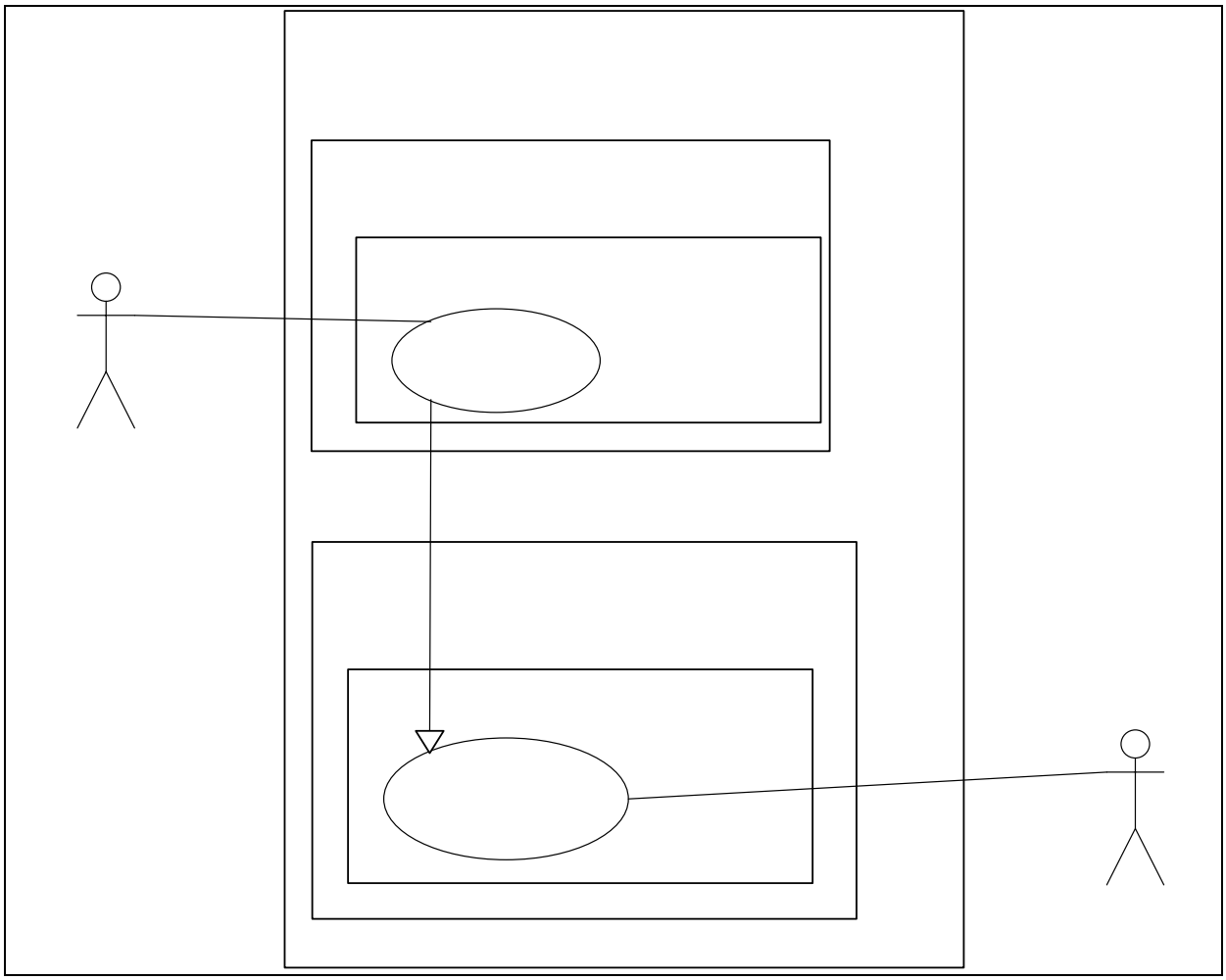


Figure 20. Use case: Retailer sends an order invoice to Supplier

Oracle BPEL Designer creates a composite Web service. The created Web service has access to two databases at Retailer and Supplier. Oracle Business Process Execution Language (BPEL) Project Manager (PM) Server provides interface to test the Web service unit. Figure 21 shows that value 135 is entered to specify purchase invoice id from the retailer.

*

Purchase Manager

Figure 21. Transaction initiator with order invoice ID

Figure 22 shows that the Web service returns a message back to acknowledge that the process succeeded. The Web service at Supplier side sends a message "good" to let Web service at Retailer side know that the inserting process for purchasing list is completed successfully. The message between organizations can be set between them. Setting specifications that universally understood acknowledgement is a good idea so that the level of interoperability can increase.

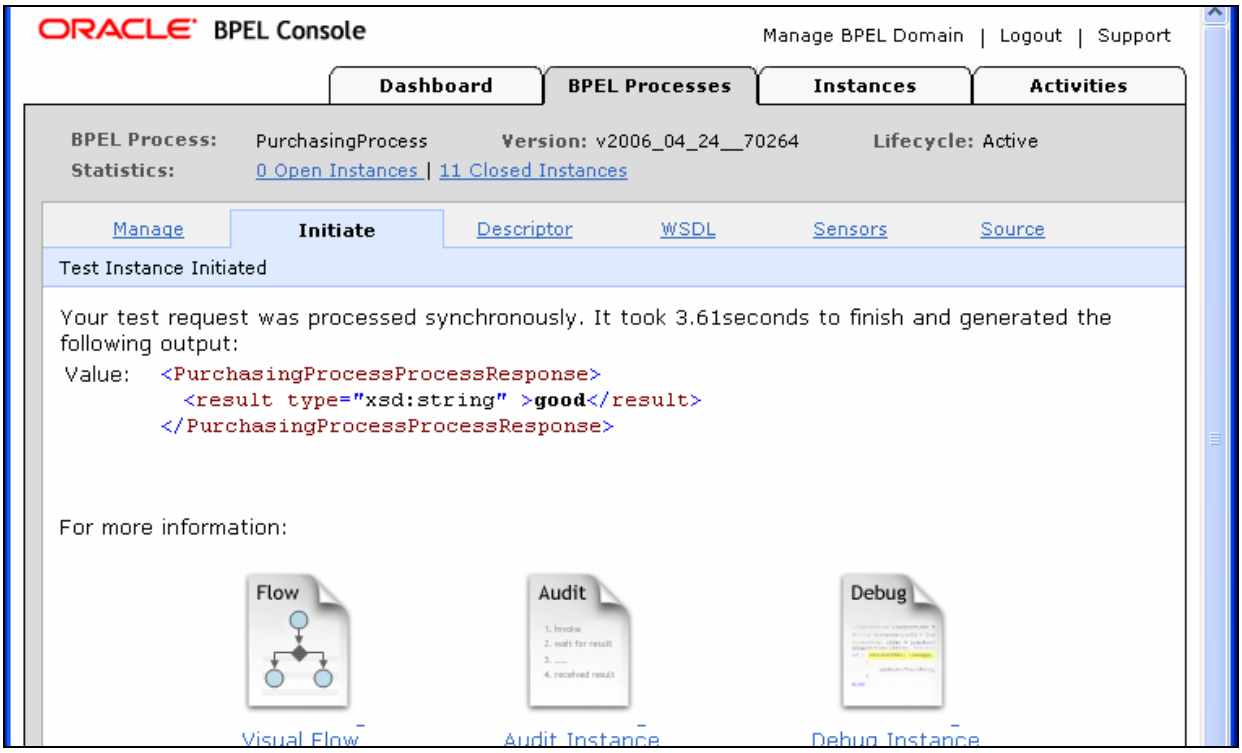
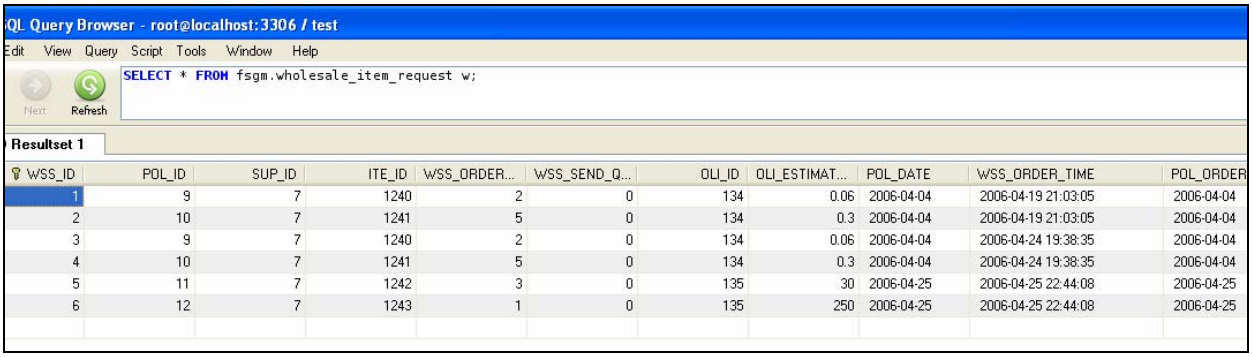


Figure 22. Result in BPEL Console

Figure 19 is the content in a table, wholesales_items_request and Figure 23 is after running the Web service to insert purchasing list from Retailer. You notice that OLI_ID 15 is inserted to the table after the transaction.



WSS_ID	POL_ID	SUP_ID	ITE_ID	WSS_ORDER...	WSS_SEND_Q...	OLI_ID	OLI_ESTIMAT...	POL_DATE	WSS_ORDER_TIME	POL_ORDER
1	9	7	1240	2	0	134	0.06	2006-04-04	2006-04-19 21:03:05	2006-04-04
2	10	7	1241	5	0	134	0.3	2006-04-04	2006-04-19 21:03:05	2006-04-04
3	9	7	1240	2	0	134	0.06	2006-04-04	2006-04-24 19:38:35	2006-04-04
4	10	7	1241	5	0	134	0.3	2006-04-04	2006-04-24 19:38:35	2006-04-04
5	11	7	1242	3	0	135	30	2006-04-25	2006-04-25 22:44:08	2006-04-25
6	12	7	1243	1	0	135	250	2006-04-25	2006-04-25 22:44:08	2006-04-25

Figure 23. Result in MySQL Query Browser

A Web service, SaveWholeSaleOrder supports the Supplier to restore the purchase request from Retailer. SaveWholeSaleOrder is a composite Web service to save the requested purchasing list to database at Supplier. Figure 24 is image of BPEL design for the composite Web service. The composite Web service does not reside at Supplier's SoSR BIS; however, another Web service, SaveWholeSaleOrder, on the right hand side is at Supplier's SoSR BIS.

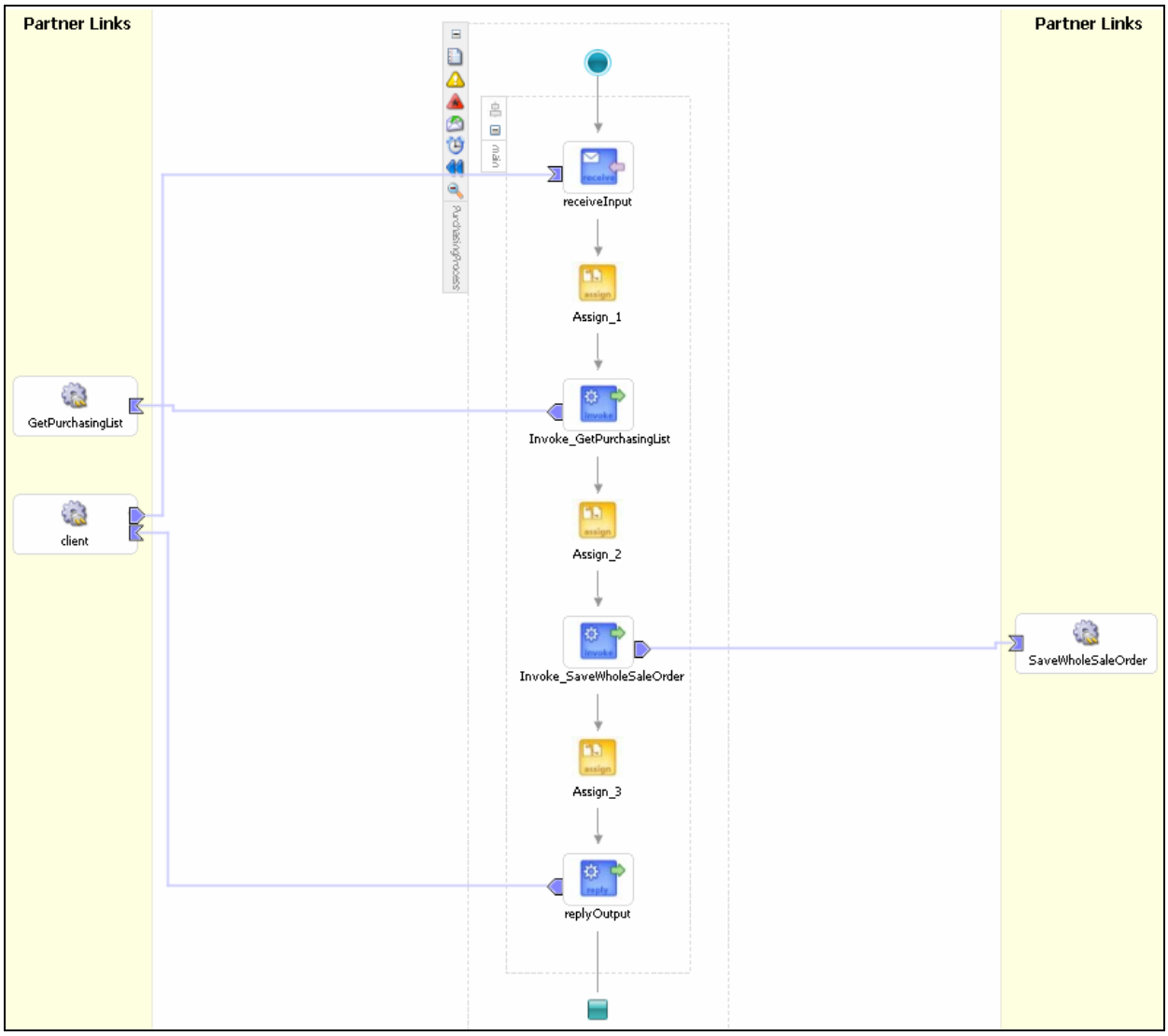


Figure 24. BPEL (Business Process Execution Language) Design view of composite services

6. Analysis of GMPO's SoSR Model

6.1. Legacy System

The system has a 3-layered component-based architecture. Web interface communicates with business functions at server side and a business process stores user's decisions to database by having connections to data interface. Figure 25 depicts a typical 3-layerd design.

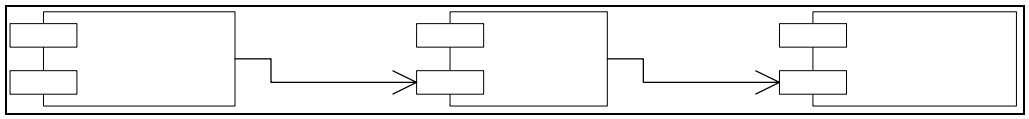


Figure 25. A Typical 3-Layered Design

Figure 26 elaborates the 3-layered design in Figure 25 by explaining ordering process in GMPO system.

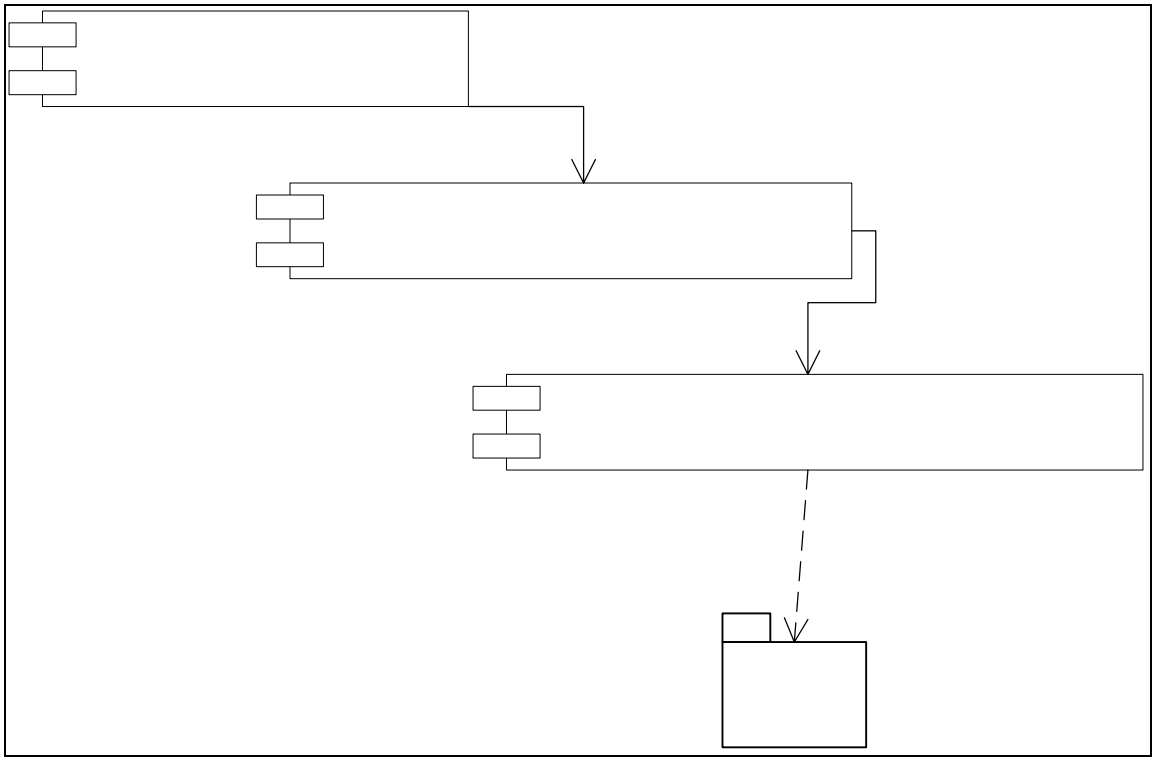


Figure 26. 3-Layered Design Applied to Saving Order List Process in Legacy System

A JSP page, purchasing_list_generator.jsp, sends purchase information from an end user to purchasing_list_generator_to_database.jsp to display the gathered information to confirm. When the information is confirmed, purchasing list is saved to Retailer's database by purchasing_list_generator_to_database_real_final.jsp.

Table 2. RACI Chart of Implementation View on Legacy System

		Roles							
		Business Analyst	Architecture and Design Reviewer	Project Manager	Resource Scheduler	Code Developer	Data Architect	Component Deployer	Development Tester
IV	Analyze dependency of components	I	R	A	C	C	C		C
	Manage uptime of components	I	I	A	C	I	C	R	C
	Register local components to the local system			A		I		R	I
	Analyze availability of components	I	I	A	R		C	I	C

The analysis of Table 2 follows. A project manager is accountable for all the tasks and performers (R) are distributed to an architecture and design reviewer, a component deployer, and a resource scheduler. An architecture and design reviewer is responsible to analyze dependency of components. A component deployer is responsible to manage uptime of components and to register local components to the local system. A resource scheduler is responsible to analyze availability of components so that other roles can contact the resource scheduler and find out the availability of components.

6.2. Target SoSR BIS Systems

6.2.1. Service Provider

Service provider is mostly interested in creating internal components and converting them into Web services. Design view of save_wholesale_order at Supplier's side in Figure 27 explains the relationship between Web service and the database tables with class diagram. A Web service associates with two tables, wholesale_inv_desc_entry and wholesale_item_request. The Web service updates purchasing invoice header information to wholesale_inv_desc_entry and line items of purchasing invoice information. This Web service becomes a provider in the role-based model.

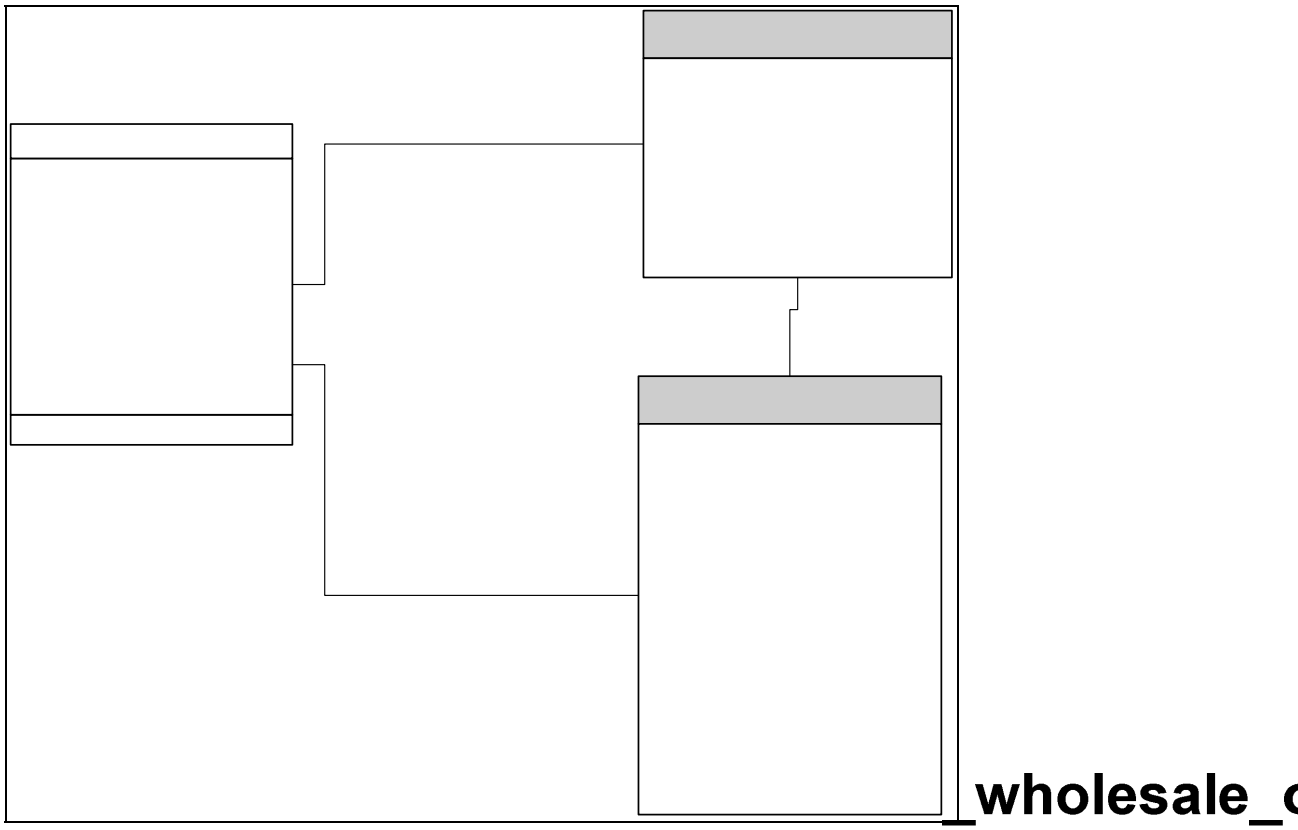


Figure 27. Design View of save_wholesale_order at Supplier's side

Figure 28 shows an implementation view of saving the purchasing list at Supplier. The component, save_wholesale_order.jws, saves purchasing information into Supplier's database and the list becomes request list for the Supplier.

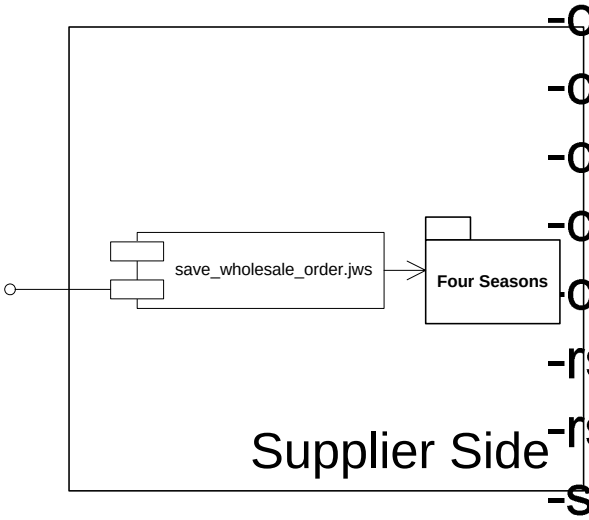


Figure 28. Implementation View of save_wholesale_order at Supplier's side

The analysis on Table 3 follows. Data architect is responsible for database design with consults with architecture and design reviewer and business analyst. Project manager is accountable for database design. Code developer is responsible of designing business and database components. Service provider needs to develop simple component level business functions. Security administrator, code reviewer, data architect, architecture and design reviewer, and development tester are taking a consulting role on design business and database components. Project manager has the 'accountable (A)' role on the task. Code developer can convert existing reusable business and database components into Web services. The Web services will be simple Web service; thus, a role for BPEL design is not required. This task is taking many resources to achieve. Besides business analyst and Web service Registrar (they have keep informed role), rest of the roles have one of the A, R, and C responsibilities. When all the testing is done, Web service registrar registers the converted Web services to Web service registries.

Table 3. RACI Chart at Design View for Service Provider

		SP									
		Business Analyst	Project Manager	Security Admin	Security Tester	Code Reviewer	Code Developer	Data Architect	Architecture and Design Reviewer	Web Service Registrar	Development Tester
DV	Design Database	C	A				I	R	C		
	Design Business /Database Components		A	C		C	R	C	C		C
	Convert reusable Business /Database Components into Web Services	I	A	C	C	C	R	C	C	I	C
	Publish Web Services to Service Registry	I	A	C	C		I	I	I	R	

Table 4 is RACI chart at component view for Service Provider. Web service master is having similar roles as Webmaster. Webmaster makes sure uptime of components and makes the resource available to end-users. Web service master is also having the same role with Web services. Project manager has the 'accountable (A)' role through out the tasks: analyze dependency of components, manage uptime of components, register local components to the local system, and analyze availability of components. Resource scheduler is responsible to decide all the human resources and facilities are well distributed and there is no extreme bottlenecked resource. Project manager is an apparent bottlenecked resource.

Table 4. RACI Chart at Implementation View for Service Provider

		SP								
		Business Analyst	Web Service Master	Architecture and Design Reviewer	Project Manager	Resource Scheduler	Code Developer	Data Architect	Web Service Registrar	Development Tester
IV	Analyze dependency of components	I	I	R	A	C	C	C	I	C
	Manage uptime of components	I	R	I	A	C	I	C	I	C
	Register local components to the local system		R		A		I		I	I
	Analyze availability of components	I	I	I	A	R		C	I	C

6.2.2 Service Broker

Among the roles in Service Broker party, BPEL designer and BPEL design reviewer are new roles from Service Provider and Service Consumer. Broker is heavily compositing simple Web services; thus, new roles BPEL designer and BPEL design reviewer are needed.

Figure 29 is an activity diagram to explain how broker may choose right Supplier based on parameters of Consumer name (or ID) and Supplier Name (or ID). Consumer initiates to send purchasing list to Supplier; however, Consumer does not send directly; rather, sends the information to Broker and Broker, with given information by Consumer, then forwards the information to Supplier. The design of this system assumes to have multiple suppliers to deal.

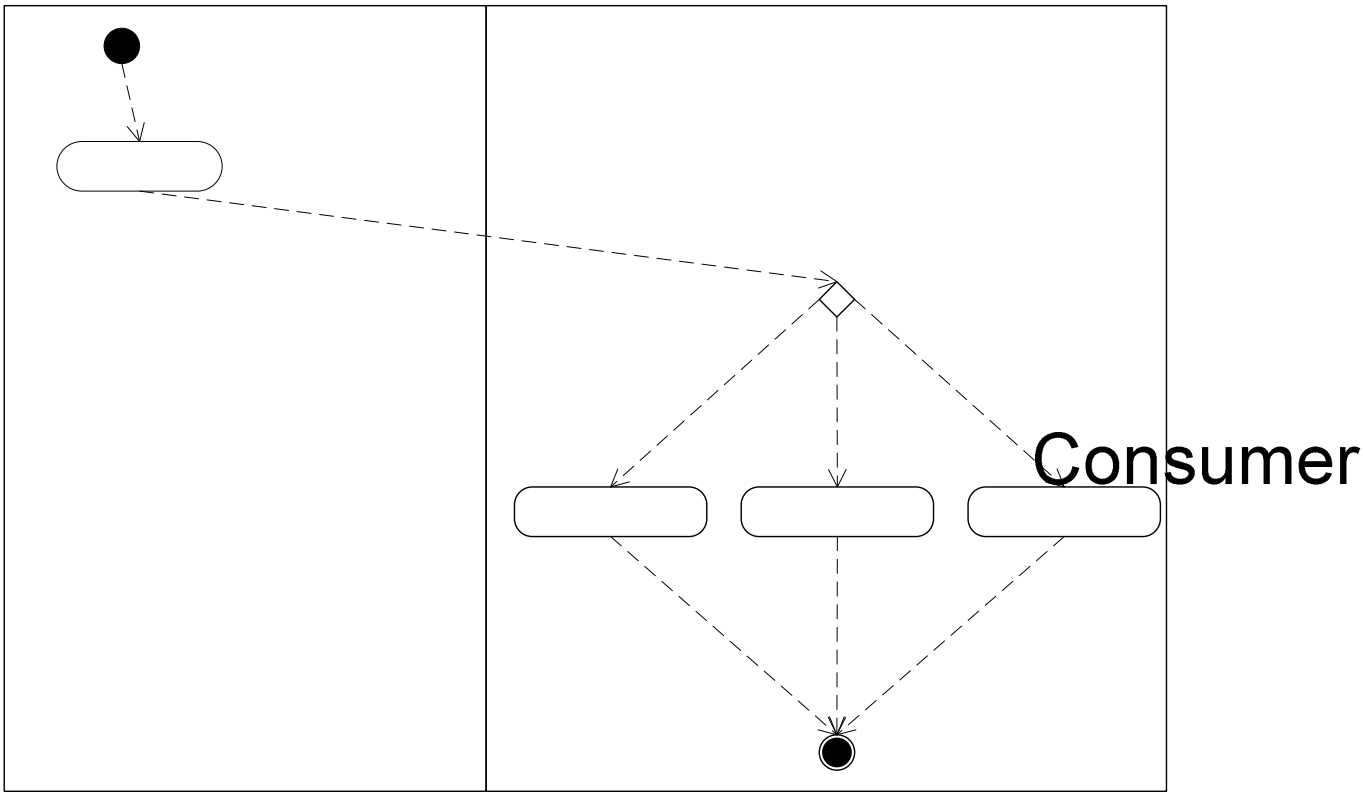


Figure 29. Activity Diagram for sending PO to Supplier process

Table 5 shows RACI chart at process view for Service Broker. A project manager is in the middle of many tasks by being accountable. The project manager is responsible for specifying activities at time and defines roles because a business analyst is accountable for these tasks.

Table 5. RACI Chart at Process View for Service Broker

		SB						
		Business Analyst	Project Manager	Resource Scheduler	Security Admin	BPEL Designer	Web Service Searcher	WS Tester
P V	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		

6.2.3 Service Consumer

Figure 30 depicts use case diagram for Service Consumer. The purchase manager calls a Web service, 'Send PO to Supplier', and later the purchase manager retrieves delivery invoice from Supplier by calling another Web service, 'Request Invoice to Supplier'.

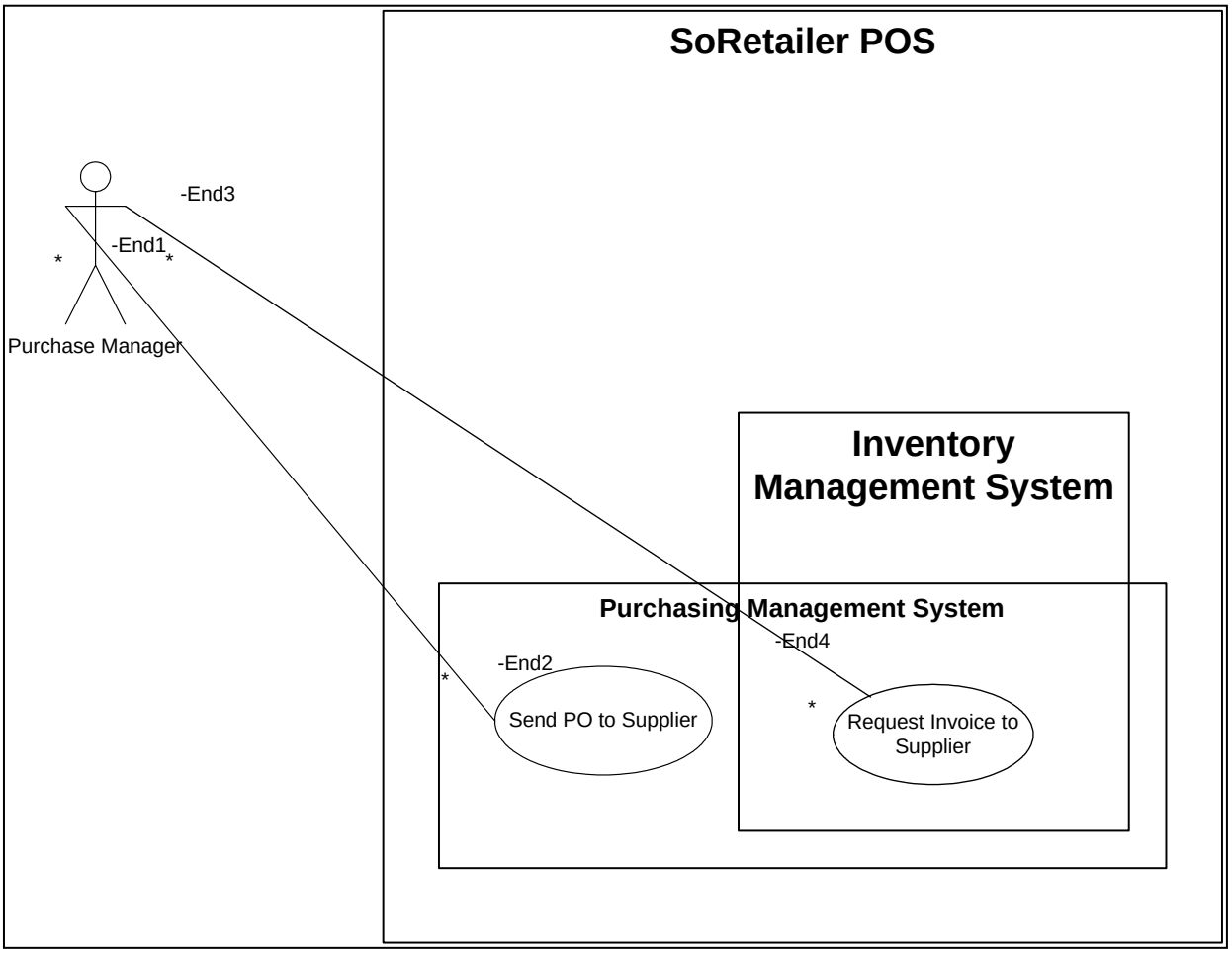


Figure 30. Use Case Diagram for send PO to Supplier process

Table 6 shows RACI chart at use case view for service consumer. Business analyst is accountable for gathering business requirements and managing business information flow. Project manager is accountable for checking availability of staffs and coordinating resources. Project manager is responsible for managing business information flow. Resource scheduler is responsible for checking availability of staffs and coordinating resources. Note that B2B (Business to Business) liaison is responsible to configure processes with other organizations.

Table 6. RACI Chart at Use Case View for Service Consumer

		SC					
		Business Analyst	Architecture and Design Reviewer	Project Manager	Resource Scheduler	Code Developer	B2B Liaison
UV	Gather Business Requirements	A	C	C		I	R
	Check Availability of Staffs	C	I	A	R		
	Coordinate Resources	C	I	A	R	I	
	Manage Business Information Flow	A	C	R	C	I	I

In Figure 31, a completed example of Provider, Broker, and Consumer are explained. Get_purchasing_list and save_wholesale_order are providers, and PurchasingProcess is Broker. Another Web service, send_purchasing_list_to_supplier is a consumer. A Web service, PurchasingProcess in Figure 31, is composite Web service and it is also called a 'food of Web'.

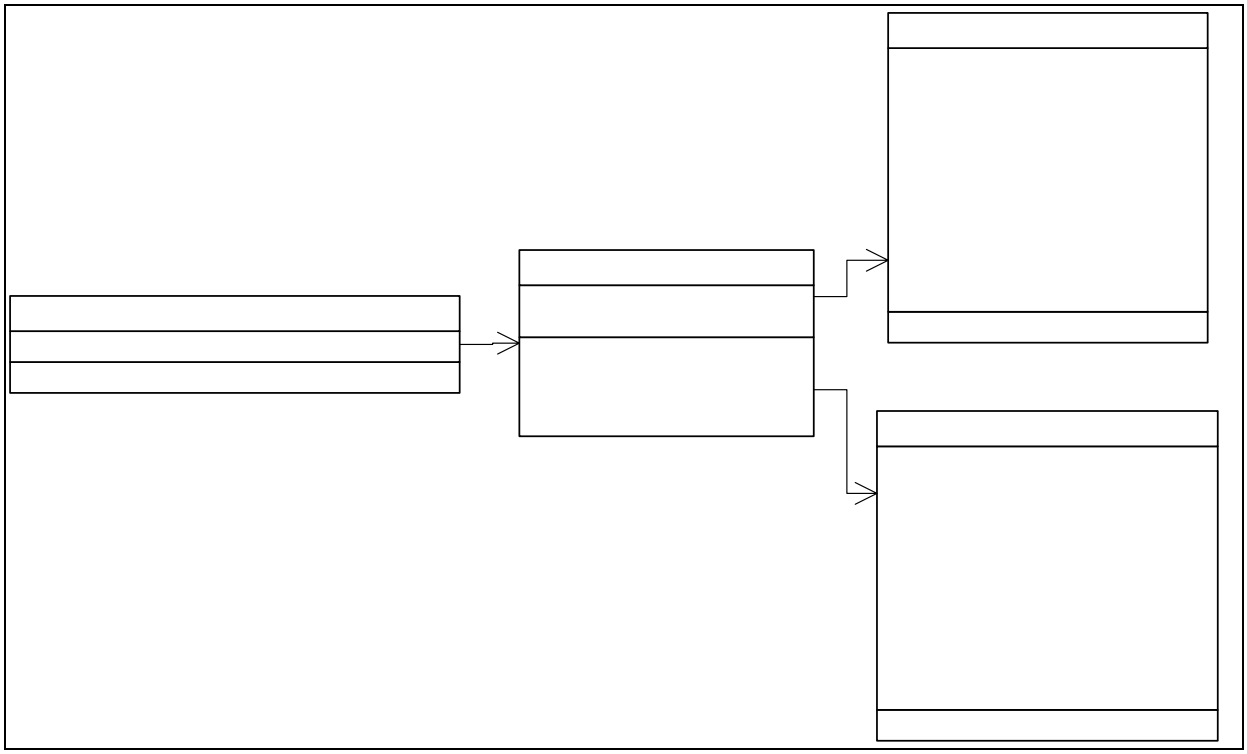


Figure 31. Design View of business process for sending purchasing list to Supplier

7. Conclusions and Future Researches

7.1 Conclusions

The SoSR methodology can help software developers and system integrators to reengineer the tightly coupled, rigid, and non-service-oriented legacy information systems to the loosely coupled, agile, and service-oriented information systems. This SoSR methodology brings several benefits for modernizing a legacy system to a target system using the SOC paradigm. First of all, the scope of a project participant is very clearly defined. When a person joins a software modernization project, the person wants know his or her scope that shows the guidelines that who will do what tasks in terms of which view. The participants of a service-oriented software modernization project can find their scopes in terms of one of three service stakeholder types such as service consumer, broker and producer and one of 4+1 views such as use case, design, implementation, process, and deployment views. Since the roles and tasks of a participant's scope are described through a RACI chart for the combination of a specific service stakeholder type and a view, the participant knows the scope in the middle of the modernization process. Also, since the diagrams in a visual model is related to the combination of a specific service stakeholder type and a view, the participant knows which diagrams need to be referenced or created within his scope.

Secondly, concrete artifacts are provided for the project participants. As soon as the participant knows his scope, the RACI chart is used to guide the reverse or forward software engineering to visualize, specify, construct, and document a specific scope of the project participant. For example, if a service stakeholder type of participant is service provider and he is interested in process view, the participant needs a role such as service composer and his task is to discover available services and compose them. He will describe the service composition in a UML activity diagram. Especially, the inclusion of a business process engine for executing the composite services to existing application and database servers shows how SOC affect future information system design, deployment, and integration.

Thirdly, this SoSE methodology provides agile guidelines. After the scope of a participant is chosen, the RACI chart can be adjusted according to the project styles. The tasks and roles in a RACI chart for a specific scope can be added, omitted, or changed. The cross cells between roles and tasks need to make sure who are 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 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, the visual model is generated as one of artifacts of each software engineering.

Lastly, popular approaches that have been employed to develop better software systems are naturally integrated into this SoSR methodology. SOA describes the three different types of service stakeholders: service consumer, service broker, and service producer. 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. Multi-tier architecture is used to explain the deployment view of each service stakeholder type. The 4+1 view has been used for model-driven object-driven development.

7.2 Future Researches

In doing this project, it is found that there are not enough related examples for SoSR methodology. The examples in this project are not enough for analysis. The SoSR community needs to grow by developing more examples of SoSR solutions, and proper responsibilities will be assigned to proper roles.

Automatic service discovery and service composition need to cope with SoSR methodology. In this project, we assume that service consumers already knew service providers and brokers. For the enhanced and robust solution, it needs to be allowed that a consumer can blindly ask for a list of broker agent to bring integrated solution and then the broker agent finds providers and other brokers and integrate on the fly. To make this happen, a registry for broker agents needs to be made.

Other services or systems need to be made as well. A service that validates found-services needs to be made. This service is validating the found-services based on criteria such as accuracy, speed, and uptime. Another service that needs to be made is negotiation agent [2, 5]. This service negotiates between a consumer and a broker agent and between brokers and providers. To create profit from providing services, this negotiation process is needed. One example of criteria of negotiation would be range of payment versus range of accepting payment.

Security expert needs to look into SoSR models and analyze, and add more tasks and the responsibilities will be adjusted. By having more examples and bigger and more communities applying this SoSR approach for software reengineering on their existing legacy systems, the missing pieces regarding security can be found and the SoSR model will be enhanced. Security on Web service is one of the functionality in WS-*. Furthermore, many enhancements can be made by coping SoSR methodology with WS-*.

References

[1] Bieberstein, Norbert, et al. (2005). Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap. IBM Press.

[2] Brereton, O.P. The Software customer/supplier relationship. Accepted for publication by Communications of the ACM. Downloaded from: Service-oriented [online]. Date of article is not known. http://service-oriented.com/publications/SoftwareDemandandSupply.pdf. Accessed May 20, 2006. 1-9.

[3] Carpenter, Jeff and Bieber, Guy. Introduction to service-oriented programming (Rev 2.1). Downloaded from: Open Wings [online]. Date of article is not known. http://www.openwings.org/download/specs/ServiceOrientedIntroduction.pdf. Accessed May 27, 2003. 1-13.

[4] Chung, Sam and Lee, Yun-Sik. (2000). Reverse software engineering with UML for Web site maintenance. First International Conference on Web Information Systems Engineering WISE. 2000. Vol. 2. P. 2157.

[5] Elfatatry, A. and Layzell, P.J. Negotiating in service oriented environments. Accepted for publication by Communications of the ACM. Downloaded from: Service-Oriented [online]. Date of article is not known. http://service-oriented.com/publications/Negotiation.pdf. Accessed May 20, 2006. 1-6.

[6] Enterprise Solutions Competency Center: U.S. Army PEO EIS & Software Engineering Center – Belvoir. Responsibility Charting (RACI). Date of article is not known. http://www.army.mil/escc/docs/RACI_Approach.pdf. Accessed May 20, 2005.

[7] Erl, Thomas (2004) Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services. Prentice Hall.

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

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

[10] Lea, Doug and Vinosk, Steve. (2003). Middleware for web services. Internet Computing, IEEE, 7 (1), 28 – 29.

[11] Meier, J, et al. (2003). Improving Web Application Security: Threats and Countermeasures. Microsoft

[12] Seacord, Robert C., Plakosh, Daniel and Lewis, Grace A. (2003). Modernizing Legacy Systems. Addison Wesley.

[13] Turner, Mark, Budgen, David, and Brereton, Pearl. Turning Software into a Service. IEEE Computer. October 2003. Vol. 36, No. 10. pp. 38-44.

[14] Wikipedia contributors, "Extreme Programming," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Extreme_Programming&oldid=55575789 (accessed May 29, 2006).

[15] Wikipedia contributors, "Unified Process," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Unified_Process&oldid=55222239 (accessed May 29, 2006).

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

[17] Wikipedia contributors, "Rational Software," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Rational_Software&oldid=55820113 (accessed May 29, 2006).

[18] Wikipedia contributors, "Scrum (development)," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Scrum_%28development%29&oldid=55379452 (accessed May 29, 2006).

Appendix

A1. RACI Chart at Process View for Service Provider on target system

		SP						
		Business Analyst	Project Manager	Resource Scheduler	Code Developer	Data Architect	Web Service Registrar	Development Tester
PV	Collaborate Processes		A	R				C
	Specify activities at time	A	R					C
	Set Input/Output parameters		A		R		I	C
	Define Roles	A	R			I		

Mostly business analyst and project manger are having major roles in process view by having A and R's. With resource scheduler's performance to schedule resources, process view can have four major tasks: collaborate process, specify activities at time, set input/output parameters, and define roles. Code developer is specifying input and output parameters and as project manager approves the setup since project manager controls all processes.

A2. RACI Chart at Deployment View for Service Provider on target system

		SP								
		Business Analyst	Web Service Master	Architecture and Design Reviewer	Project Manager	Resource Scheduler	Code Developer	Data Architect	Web Service Registrar	Deployment Tester
LV	Register to WS Registry	I	C	I	A		I		R	I
	Check Security	I	C	C	A	C	C		I	R
	Design Deployment Plan	I	I	C	A	R	C	C	C	I
	Test Deployed WS		I	I	A	I	I	I	I	R

Deployment is significant, because the end user can access to Web services when the services are deployed. Deployment tester is having important role in this phase by checking security and testing deployed Web services. Resource scheduler also takes an important role by preparing deployment plan. Web service registrar is responsible to register Web services to Web service registries.

A3. RACI Chart at Use Case View for Service Provider on target system

		SP								
		Business Analyst	Web Service Master	Architecture and Design Reviewer	Project Manager	Resource Scheduler	Code Developer	Data Architect	B2B Liaison	Team Lead
U V	Gather Business Requirements	A		C	C		I		R	R
	Check Availability of Staffs	C		I	A	R				C
	Coordinate Resources	C		I	A	R	I			C
	Manage Business Information Flow	A	I	C	R	C	I	C	I	C

Internally project manager is taking an accountable (A) role; however, business analyst is having a significant role to gather business requirements from end user and to manage business information flow. Team leads are best staffs know how the business work internally in detail; thus, they provide what inputs may require and what outputs are available to provide. Resource scheduler checks availability of staffs and coordinates resources.

A4. RACI Chart at Deployment View for Service Broker on target system

		SB								
		Security Admin	Deployment Engineer	BPEL Design Reviewer	Project Manager	Resource Scheduler	BPEL Designer	Security Tester	Web Service Searcher	Deployment Tester
L V	Check Security	R	I	C	A	I	C	C	I	C
	Design Deployment Plan	C	R	C	A	C	C	C	I	I
	Test Deployed Solution	I	C	I	A	I	I	C	I	R

A5. RACI Chart at Use Case View for Service Broker on target system

		SB					
		Business Analyst	BPEL Design Reviewer	Project Manager	Resource Scheduler	BPEL Designer	B2B Liaison
U V	Gather Business Requirements	A	C	C	C	C	R
	Check Availability of Staffs	C	I	A	R	I	I
	Coordinate Resources	C	I	A	R	I	I
	Manage Business Information Flow	A	C	R	C	R	I

Use case view in Service Broker is similar to Service Consumer's. However, Service Broker another distinctive task called, 'manage business information flow'. 'manage business information flow' is a task to design BPEL process and BPEL designer designs and business analyst authorize the business information flow. Project manager is another responsible role to

manage business information flow. When two roles are having responsible roles, the communication may not be smooth. However, Project manager is overlooking all process and BPEL designer is a role to do real configuration. BPEL designer needs to know all business process related to Web services. BPEL designer has much of access to resources so that the configuration of Web services can go smoothly. B2B liaison also has an important role to gather business requirements from other businesses that are providing Web services. B2B liaison needs to be informed how the services are configured so that service resources are well managed kept.

A6. RACI Chart at Design View for Service Broker on target system

		SB										
		Web Service Registrar	Business Analyst	Project Manager	Security Admin	Security Tester	BPEL Design Reviewer	BPEL Designer	Web Service Analyst	WS Negotiator	Web Service Searcher	Web Service Tester
DV	Find Published WS		I	A						I	R	
	Analyze Found WS		C	A	C	C	C	C	R		C	C
	Check Requirements to consume found WS		I	A	C	C	C	C	C	R	I	I
	Implement Composite WS with found WS's	I	I	A	I	C	C	R	C	I	I	C
	Monitor downtime of found WS			A			I	I	I	I	I	R
	Publish Web Services to Service Registry	R	I	A	C	C	I	I	I	I	I	C

A7. RACI Chart at Implementation View for Service Broker on target system

		SB								
		Business Analyst	Security Admin	BPEL Design Reviewer	Project Manager	Resource Scheduler	BPEL Designer	Web Service Searcher	Security Tester	WS Tester
IV	Analyze dependency of components	R		C	A		C	C	C	C
	Manage uptime of WS	C		I	A	R	I	I		C
	Review Integration of WS	I	I	R	C	C	A	C	C	C

A8. RACI Chart at Process View for Service Consumer on target system

		SC						
		Business Analyst	Project Manager	Resource Scheduler	Code Developer	Data Architect	Web Service Searcher	WS Tester
PV	Collaborate Processes		A	R				C
	Specify activities at time	A	R					C
	Configure Input/Output parameters		A		I		R	C
	Define Roles	A	R			I		

A9. RACI Chart at Deployment View for Service Consumer on target system

		SC								
		Security Admin	Deployment Engineer	Architecture and Design Reviewer	Project Manager	Resource Scheduler	Code Developer	Security Tester	Web Service Searcher	Deployment Tester
LV	Check Security	R	I	C	A	I	C	C	I	C
	Design Deployment Plan	C	R	C	A	C	C	C	I	I
	Test Deployed Solution	I	C	I	A	I	I	C	I	R

A10. RACI Chart at Design View for Service Consumer on target system

		SC									
		Business Analyst	Project Manager	Security Admin	Security Tester	Code Reviewer	Code Developer	Web Service Analyst	WS Negotiator	Web Service Searcher	Web Service Tester
DV	Find Published WS		A						I	R	
	Analyze Found WS	C	A	C	C	C	C	R		C	C
	Check Requirements to consume found WS		A	C	C	C	C	C	R	I	I
	Implement to use found WS		A		C	C	R	C	I	I	C
	Monitor downtime of found WS		A			I	I	I	I	I	R

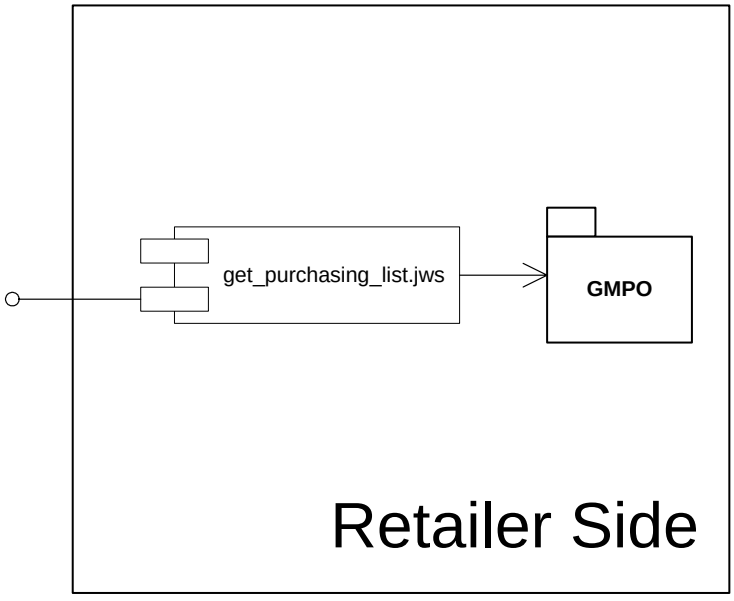
At design phase, Web service searcher, Web service negotiator, Web service tester, and Web service analyst are having significant roles. Web service searcher finds available services and Web service analyst evaluates found Web services with Web service testers and Web service negotiator. When the negotiation is harder or costly; then, Web service analyst may not consider using the specific service and find for alternative services and evaluation process reoccurs on found services. Code developer does not need to do hardcore development for developing complex business processes, but needs to use evaluated Web service accordingly.

A11. RACI Chart at Implementation View for Service Consumer on target system

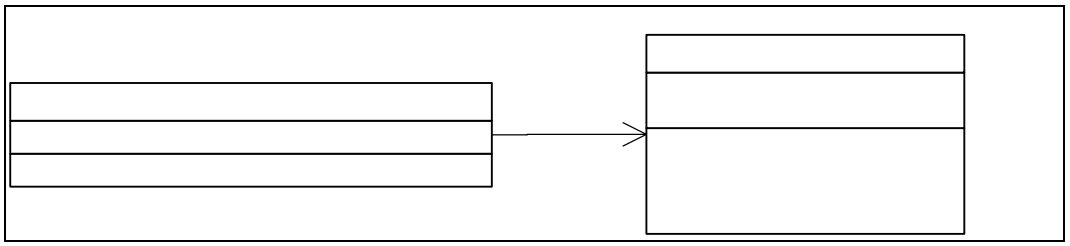
		SC								
		Business Analyst	Security Admin	Architecture and Design Reviewer	Project Manager	Resource Scheduler	Code Developer	Web Service Searcher	Security Tester	WS Tester
IV	Analyze dependency of components	R		C	A		C	C	C	C
	Manage uptime of WS	C		I	A	R	I	I		C
	Configure integration of WS	I	I	C	A	C	R	C	C	C

At design phase, Web service searcher, Web service analyst, Web service negotiator, and Web service tester are doing the same task as in Service Consumer. However, Service Broker has BPEL designer and BPEL design reviewer. When Web services are evaluated, BPEL designer will integrated. BPEL designer uses a highly abstracted interface to configure processes between Web services to create complex Web services. A Web service solution can be created by compositing existing Web services. Project manager authorizes the artifact of compositing Web services.

A12. Component Diagram (IV) of Web service, get_purchasing_list.jws on target system

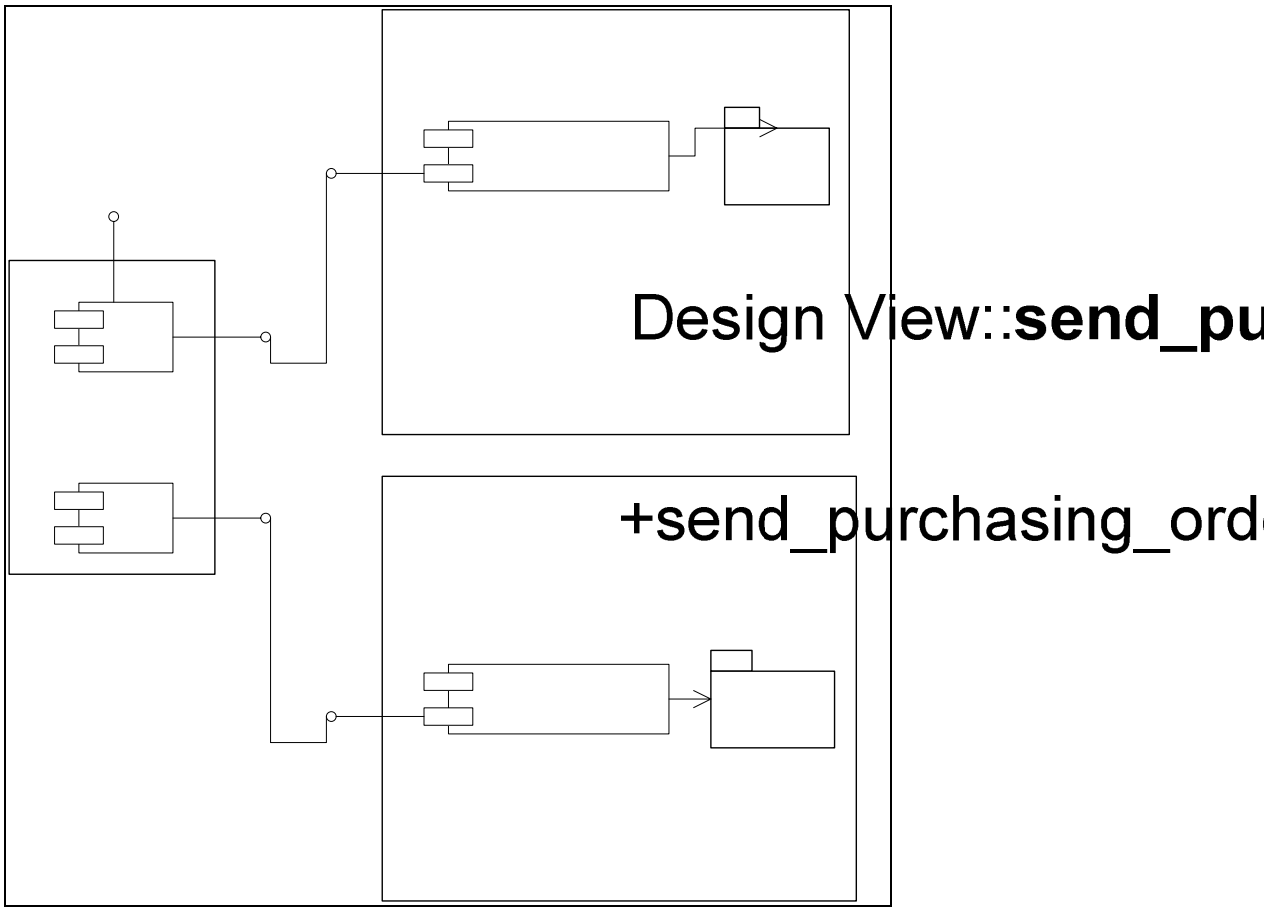


A13. Design View of calling composite Web service for Service Broker on target system

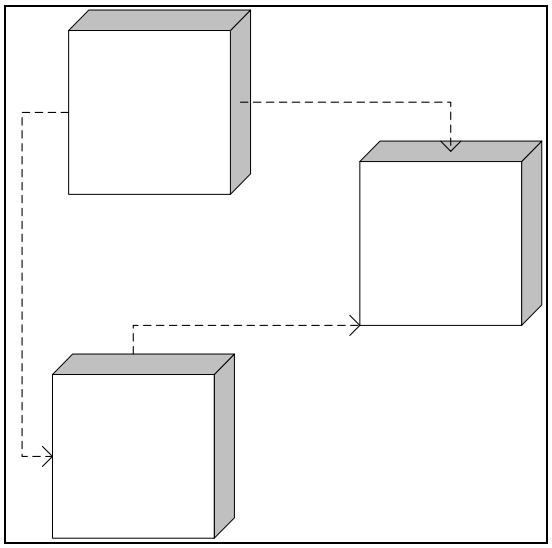


A Web service send_purchasing_list_to_supplier consumes another Web service, PurchasingProcess. Web service, send_purchasing_list_to_supplier, becomes a consumer and another Web service, PurchasingProcess, becomes a provider.

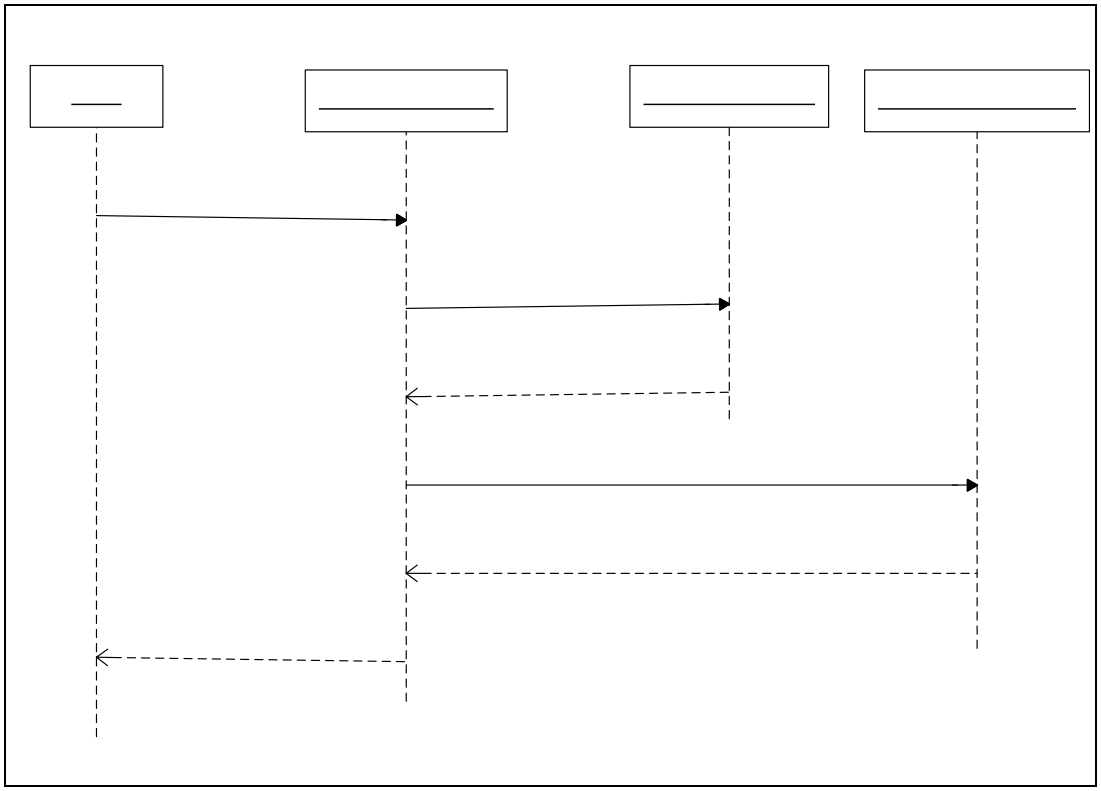
A14. Component Diagram (IV) of a Composite Web service for Service Broker on target system



A15. Deployment View for Three Participants on target system



A16. Sequence Diagram View of business process for sending purchasing list to Supplier on target system



A17. Component Diagram (IV) of Composite Web service with Consumer with a caller component on target system