Service-Oriented Software Reengineering Methodology for Composite Services

Craig Niiyama[1], Sam Chung[1], Donald Chinn[1], Sergio Davolos[2]
Computing & Software Systems
Institute of Technology
[1]University of Washington, Tacoma
{cniiyama, chungsa, dchinn}@u.washington.edu

[2]Milgard School of Business
University of Washington, Tacoma
sergiod@u.washington.edu

Abstract - **The purpose of this paper is twofold: 1) to apply a current reengineering methodology called Service-Oriented Software Reengineering (SOSR) to the modernization of a legacy application into a service-oriented application, and 2) to extend the SOSR methodology to include composite services. There is a need for new software methodologies to address the effects of composite web services to software reengineering projects. One of the methodologies called SOSR is still in its early stages and has yet to be fully tested against applications designed and implemented with a business process engine. It has not been applied to applications using composite web services. For this purpose, the reverse software engineering portion of the current SOSR is applied to a service-oriented system in which composite services are heavily employed. Based upon the results of reverse engineering, the SOSR methodology is extended to include guidelines for a service broker who composes a set of services, deploys, and executes the composite services. And then, we verify the extended SOSR methodology by reengineering a legacy system not using services by following the methodology. The guidelines of a service broker are described in terms of roles of a service broker over time, tasks within 4+1 views, the descriptions of the tasks in UML (Unified Modeling Language), and the relationships between roles and tasks, which are represented by an RACI (Responsible, Accountable, Consulted, and Informed) chart. The results of this research will help a software developer to reengineer a legacy system to a target system using composite services.**

I. Introduction

In Service-Oriented computing (SOC) [11] the software developer uses web services as fundamental building block components to their software development process. Services are platform and network-independent operations that clients or others services invoke. SOC developers use service composition to create applications on top of SOC's native capabilities such as standard web service interface language and interaction protocol. Developers can compose available web services to create new applications to solve complex business problems [6]. Service composition accelerates Rapid Application Development (RAD), service reuse, and complex service consumption.

The goal and need of creating new business processes composed of multiple web services to solve complex business problems is behind the development of many new Web Services Business Process Execution language (WS-BPEL or BPEL) based tools [4]. BPEL is an emerging industry standard language of how web services are arranged and composed to solve complex business problems and has the full backing of industry leaders such as Microsoft [20], Oracle [21], Sun Systems [22], IBM [23], as well as the open source community [24]. In fact, Microsoft, Oracle, and IBM as well as many other companies and open source projects are vying for positions in this new technology and actively developing tools to utilize it [17]. However, many of them are at their infant stages[1].

Service-Oriented Software Reengineering (SOSR) methodology was proposed by Chung et al [7] to support software developers to reengineer a legacy software system into a service-oriented software system(s) based on the SOC paradigm [7]. The SOSR methodology is based upon service orientation, layered and n-tier architectures, roles and its specific tasks using RACI (Responsible, Accountable, Consulted, and Informed) charts, and UML (Unified Modeling Language) modeling. However, there is a need for improving this methodology and for examples that can be used by software developers to accommodate composite web services based application development. The SOSR methodology is still in its early stages and has yet to be fully tested against applications designed and implemented with BPEL engines. It has not been applied to applications heavily using composite web services.

In this paper, firstly, we applied the SOSR methodology to a well known system called a loan application process system

that was developed by Oracle to demonstrate Oracle's Business Process Engine called Oracle BPEL Process Manager [21]. Since the loan application process system invokes a composite service that consists of three services, the SOSR methodology can be extended to provide guidelines for service brokers who will compose available services into a composite service and publish the composite service. The extended SOSR is described in terms of roles of a service broker over time, each role's tasks within 4+1 views, the descriptions of the tasks in UML, and the relationships between roles and tasks, which are represented by an RACI chart. Secondly, the extended SOSR considering the effects of composite services is applied to a software reengineering project that reengineers a legacy application into a service-oriented application using composite services heavily and is verified. The results of this research will help a software developer to reengineer a legacy system to a target system using composite services by providing guidelines for the software developer as a service broker.

This paper is organized as follows. In Section II we discuss the related technologies, Service-Oriented Architecture (SOA), and WS-BPEL. In Section III, we revisit the related software reengineering methodology, SOSR. We then present in Section IV a case study of applying the reverse software engineering portion of SOSR methodology to a service-oriented system using a composite service. Then, in Sections V, the SOSR is extended. And then, the Extended SOSR is applied to a reengineering project in Section VI and its effect is discussed. In Section VII, we present our conclusions and future work.

II. RELATED WORKS

A. *Service-Oriented Architecture (SOA) and Related Technologies*

Service-Oriented Architecture (SOA) will provide the basis for the next generation of distributed software systems [11]. It has already proven itself in the industry through an XML-based instantiation, web services. SOA enables the design of loosely coupled software for integration with other software systems, which is achieved when the functional and non-functional behavioral characteristics of services are published in a standardized and machine readable format [9]. This interface definition hides the implementation of the language-specific service such that SOA-based systems are independent of development technologies and platforms (such as Java, .NET etc). Services written in C# running on .NET platforms and services written in Java running on Enterprise Edition platforms, for example, can both be consumed by a common composite application. Applications running on either platform can also consume services running on the other as Web services, which facilitates reuse.

Furthermore, SOA gives hope of bridging Information Technology (IT) and business by providing a mechanism for defining business services and operating models in terms of actual business requirements. These services are able to adapt quickly and efficiently to the needs and requirements of the business environment. The purpose of using SOA as a business mapping tool is to ensure that the services created properly represent the business view and are not just what IT thinks the business services should be. At the heart of SOA planning is the process of defining architectures for the use of information in support of the business, and the plan for implementing those architectures [15]. Every service should be created with the purpose to bring value to the business in some way and should be traceable back to the business architecture.

The key components of SOA namely are web services and Web Service Description Language (WSDL) [13]. Web services are software components that are published through an interface agreement that defines the tasks, data and behavior of the service and the messages they accept and return [11]. This interface is platform independent and the technology used to exchange the messages is also platform independent. This provides the added benefit of code reuse and RAD. Web services are able to be used recursively saving both time and money and are able to be updated, adapted, and changed to meet the ever changing needs of business.

In order for web services to be located, WSDL was developed. WSDL is an Extensible Markup Language (XML) format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information [14]. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. In addition, other systems interact with the web service in a manner prescribed by its description using Simple Object Access Protocol (SOAP) messages, typically conveyed using Hypertext Transfer Protocol (HTTP) with an XML serialization in conjunction with other Web-related standards. Web Service technologies (XML, SOAP, and WSDL) provide the means to describe, locate, and invoke a web service.

B. *Web Services Business Process Execution Language (WS-BPEL or BPEL)*

One of unique characteristics of web services is that it allows business and IT to build a closer relationship. This is due to the fact that web services represent a higher level abstraction, business process, with respect to traditional middleware objects and are often used to support business-to-business interactions [12]. Furthermore, business processes and business needs are able to be directly translated into business process applications through composite web services. Composite web services are compositions of web services that model a business process in direct response to a business need. These can be recursively created on the fly and can be changed as the business

[1] Among them, Oracle's Business Process Engine is mature to be tested and is employed in this paper.

environment changes [13]. In order to compose web services in an efficient, timely, and accurate manner that supports RAD, there has grown a need for tools that support this process. The BPEL tools create new web processes that are composed of multiple web services that solve business problems [4].

BPEL is an emerging industry standard of how web services are arranged and composed to solve complex business problems. There are two ways that BPEL can be used to create web services: orchestration and choreography [2]. In orchestration, a central process takes control over the involved web services and coordinates the execution of different operations on the web services involved in the operation which is done with the given requirements of the orchestration. The involved web services do not know (and do not need to know) that they are involved with other services and that they are a part of a higher business process. Only the central coordinator of the orchestration needs to know this, so the orchestration is centralized with explicit definitions of operations and the order of invocation of web services [2]. In contrast, Choreography does not rely on a central coordinator. Rather, each web service involved in the choreography knows exactly when to execute its operations and whom to interact with. Choreography is a collaborative effort focused on exchange of messages. All participants of the choreography need to be aware of the business process, operations to execute, messages to exchange, and the timing of message exchanges.

From the perspective of composing web services to execute business processes, orchestration provides a more flexible approach compared to choreography as one knows who exactly is responsible for executing the whole business process. Also, alternative scenarios can be provided when faults occur. BPEL follows the orchestration structure. Choreography is covered by other standards such as Web Services Choreography Interface (WSCI) and Web Services Choreography Description Language (WS-CDL) [5].

III. Service-Oriented software reengineering (SOSR) Revisited

In order for businesses to fully make use of these SOA and the benefits that a business centric application develop lifecycle provide, there is a need for the reengineering of legacy software. Most legacy software is tightly coupled and not designed with "software as a service" methodologies in mind. Many legacy software systems need to be converted to ones using SOA and methodologies in regards Service-Oriented Software Reengineering (SOSR) are being researched [7]. Fortunately, due to the nature of web services and the interoperability between heterogeneous systems, SOC makes reengineering more of a possibility than previous architectures (i.e. Object-Oriented and Structured Computing). However, there needs to be solid methodologies for this to take place. There has been some work done in the area of reengineering of legacy software towards SOA but due to the emerging nature of the technology there is much more work to be done.

In order to understand SOSR, we need to understand a reverse engineering and design recovery taxonomy proposed by Chikofsky and Cross [10]. This article provides a framework for examining reverse-engineering technologies by defining important terms and identifying common objectives. Reverse engineering is rapidly becoming a recognized and important component of future Computer Aided Software Engineering (CASE) environments. Because the entire life cycle is naturally an iterative activity, reverse-engineering tools can provide a major link in the overall process of development and maintenance. As these tools mature, they will be used in all phases of the software lifecycle. They will be a permanent part of the software development process. To meet their true potential, CASE environments are being applied to the problems of maintaining and enhancing existing systems. This is where reverse-engineering approaches come into play. This article clarifies some of the confusion in reverse-engineering principles by defining six terms: forward engineering, reverse engineering, re-documentation, design recovery, restructuring, and reengineering. The term "reverse engineering" has its origin in the analysis of hardware – where the practice of deciphering designs from finished products is regularly done. Reverse engineering is regularly done to improve on one's product as well as analyzing competitor's products. This article sets the foundations for reengineering. It defines the key terms in this domain and sets the stage for future research. Reengineering is becoming a key part of the software development lifecycle. Software isn't static as the needs of the customers are not static. In this ever changing environment it is important that software be designed in such a way as to facilitate easy reengineering. SOSR extends the concept of reengineering in term of SOC.

SOSR heavily uses the 4+1 View Model of software development architecture of Kruchten [1]. This article presents a model for describing the architecture of software-intensive systems, based on the use of multiple concurrent views. The use of multiple views allows one to address the concerns of various stakeholders separately in terms of the functional and non functional requirements. The views are designed using an architecture-centered, scenario-driven, iterative development process. The description of software architecture can be organized around four views: logical view, development view, process view, and physical view, and then illustrated by a few selected use cases, or scenarios that becomes the fifth view. The article goes on to describe each of these views in detail. The "4+1" view model has been used with success on several large projects. It allows the various stakeholders to find what they want to know about the software architecture. Systems engineers approach it from the physical view, then the process view. End-users, customers, and data specialists approach it from the logical view. Project managers and software configurations staff see it from the development view. This article provides the groundwork for future work in this area. Other models of software architecture use this paper as a reference of the "4+1" view of software architecture. This model also provides the framework for software reengineering

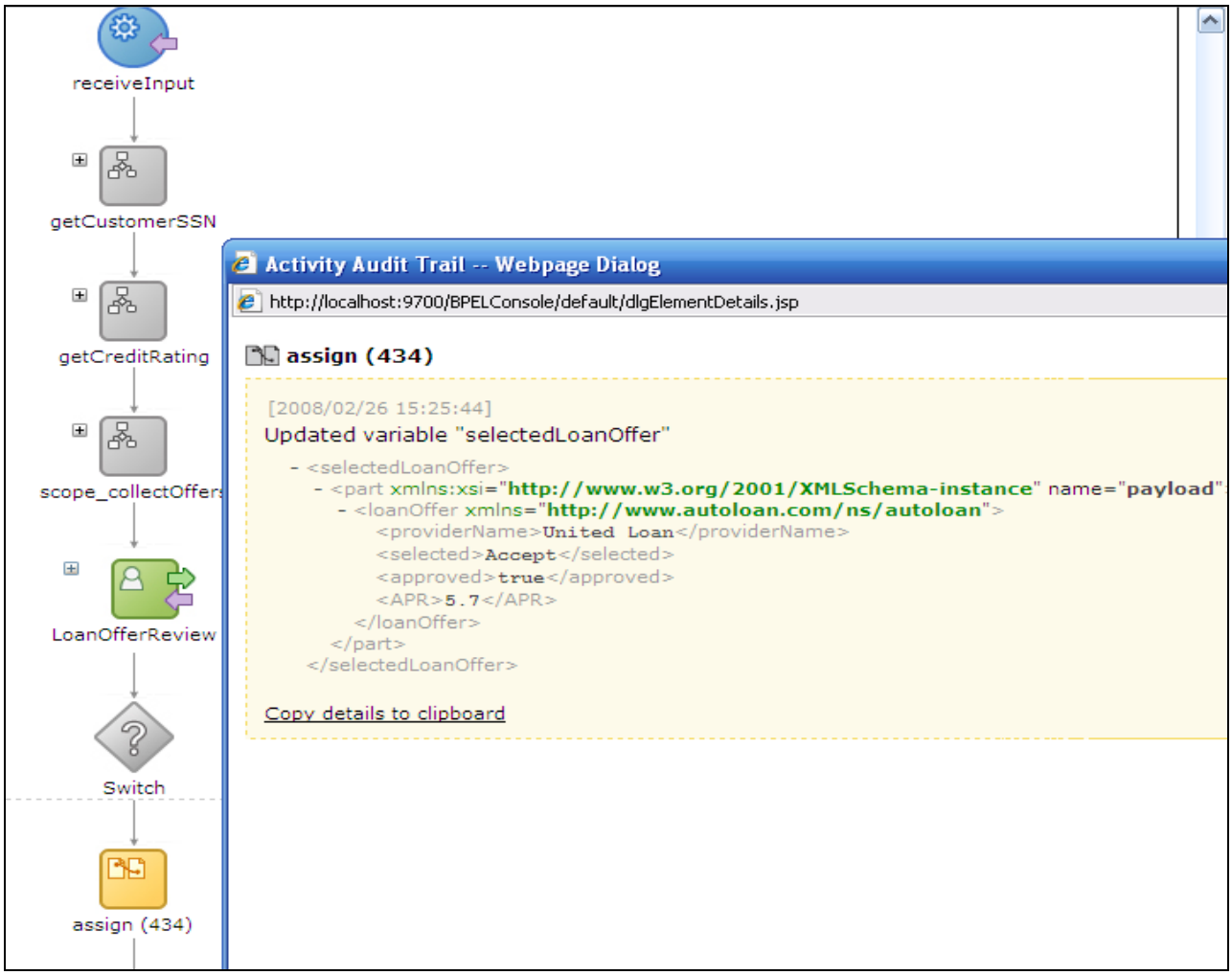

Fig. 1. Designing a composite service called 'LoanFlowPlus' for the loan application process

models. This model can be improved by taking into account new reengineering methodologies that include web service composition.

Finally, Chung et al (2007) propose a SOSR methodology for the reengineering of legacy software into a service-oriented system [7]. SOSR is architecture-centric based on the following architectures: 3-layered architecture for design, n-tier architecture for deployment, and SOA for integration. SOSR methodology uses services as the main building block for application development. Roles and tasks play are a key part of SOSR. RACI charts are used to analyze a systems design and assign specific tasks and roles to different phases. Also, the SOSR methodology produces visual models for each service participant and model view.

This paper provides key information about new software design methodologies developed around the new paradigm of web services. The reengineering of legacy systems is an important issue as technology continues to change. One can take legacy systems and create web services out of the important components. Sometimes that can be cost effective and sometimes it is not cost effective. SOSR takes this into consideration.

The SOSR methodology consists of reverse software engineering and forward software engineering. The reengineering process begins with defining the modernization requirements and targeting the components that will be reengineered. Then a visual model of the legacy system is created. Based upon the visual model, the legacy system is modernized into a target system. Finally, a visual model for the target system is designed.

Another important part of SOSR is the application of RACI charts to the process. When starting a project, it is important to identify roles and tasks as well as to list all the tasks for each role (Booch et al., 2005). The RACI chart explains the following roles: Responsible, Accountable, Consulted, and Informed [18]. These roles are assigned to different participants and for specific tasks. The RACI chart that does not include the service broker as web services was not a viable option at the time of RACI inception. This chart can be analyzed both horizontally and vertically. When there is a high concentration of the same letters on a given row or column then there is a potential for a bottleneck. The charts that have been proposed in SOSR have not fully taken into account composite services and the roles and job descriptions that need to be created.

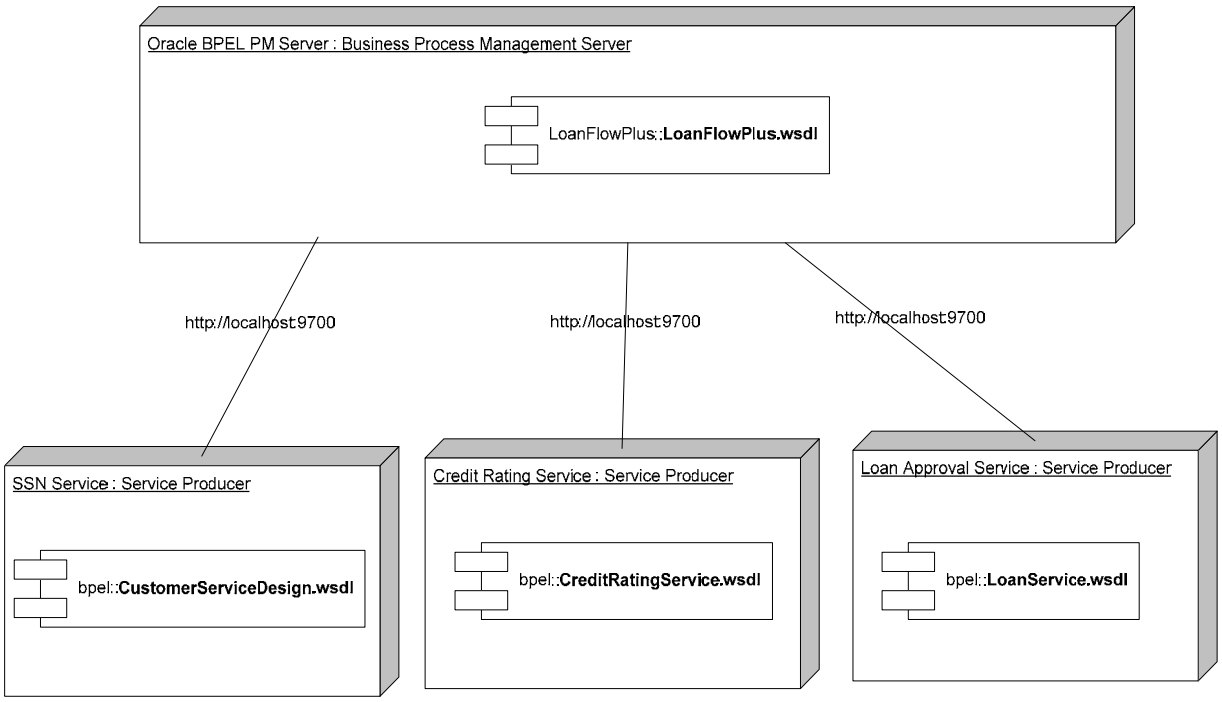

Fig. 2. A deployment diagram for the loan application process system

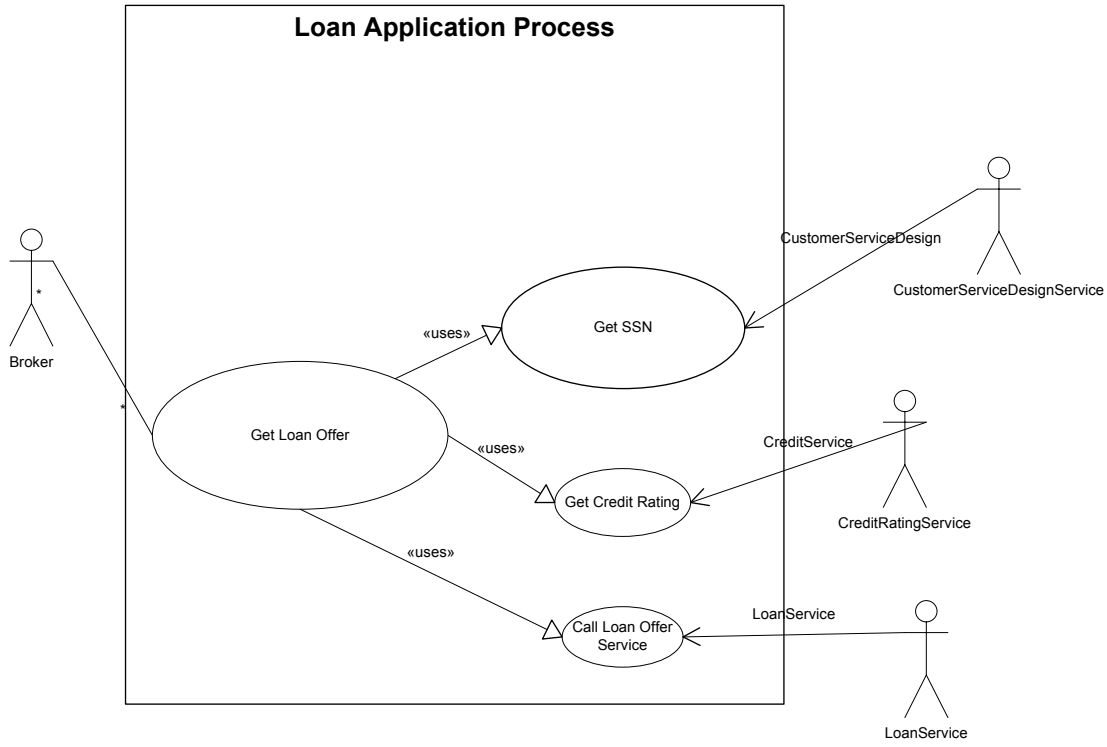

Fig. 3. A use case diagram for the loan application process system

Chung et al (2007) provides the foundation for SOSR and propose key concepts but didn't go very deep into the area of composite web services. Instead, they mentioned a future work needs to be done in the area of composite web services and business processes and how SOSR relates to this issue.

IV. REVERSE ENGINEERING OF LEGACY APPLICATION – LOAN APPLICATION PROCESS SYSTEM

We applied the reverse engineering portion of SOSR methodology to a well known system called a loan application process system that was developed by Oracle [21]. This loan

application process system demonstrates how a composite service is designed and invoked by using Oracle's Business Process Engine called Oracle BPEL Process Manager. The loan application process consists of a sequential business process that takes a loan application document as input then it returns a selected approved loan offer as the final output.

First of all, the user enters personal information on a webpage to start the loan application process. Next, the information is sent to a composite service called 'LoanFlowPlus'. The composite service consists of three simple services: 'CustomeServiceDesign', 'CreditRatingService,' and 'LoanService.' The 'CustomeServiceDesign' returns the Social Security Number (SSN) of the applicant from an Enterprise Java Bean (EJB) (accessed as a service through the native EJB Web Services Invocation Framework (WSIF) binding). The SSN is then used to retrieve the credit rating of the applicant by calling a synchronous 'CreditRating' web service to request a credit rating for that customer. Once the credit rating is obtained it sends the completed loan application to a loan processing service called 'LoanService' that finishes that application process and provides an interest rate offer and delivers it to the customer. The selected and approved offer is returned as a result of the flow. The customer can then accept or reject the offer from the process. The automation of the application process saves time and energy as well as reduces the amount of human error that can occur if done manually. Figure 1 shows how the composite service in BPEL that consists of three services is designed for the loan application process by using Oracle Business Process Manager.

Based upon SOSR, if we reverse-engineer the loan application process system, we need to generate the following artifacts: 1) a visual model with 4+1 view and 2) a RACI chart. However, since the reverse engineering in SOSR [1] does not consider the roles of a service stakeholder, we need to consider the reverse engineering in terms a specific service stake holder. Since the role of a service broker is to compose services, we generate the artifacts in terms of a service broker.

The service broker's role is sub-classified into administrator, programmer, designer, developer, and analyst. The first step is to understand the current deployment of the application as a system administrator. The deployment view is represented by using a deployment diagram, which is shown in Figure 2: The 'LoanFlowPlus' composite service is composed of three simple services that are provided by three service producers, respectively.

Once the system's current deployment is analyzed, then the implementation view is created for a (Service-Oriented) programmer followed by the design view for a designer, the process view for a developer, and the use case view for an analyst, respectively. Figure 3 shows the use case view of the loan application.

V. THE EXTENDED SOSR

Based upon the experiences of reverse engineering the loan application process system, the SOSR methodology is extended to provide guidelines for service brokers who compose available services into a composite service and publish the composite service.

A. Service-Oriented Reverse Software Engineering Matrix

In SOSR, the artifacts of reverse software engineering are a UML visual model and an RACI chart that is generated in terms of 4+1 views dimension. Since we assumed that a legacy system did not use the SOC paradigm in the current SOSR, the service stakeholder dimension was not considered. The reverse software engineering is represented in a vector called Reverse Software Engineer (RSE) vector:

Definition 1: For all i $\in$ *{UV, DV, IV, PV, LV}*, RSE[i] = {M, C}, where UV = Use Case View, DV = Design View, IV = Implementation View, PV = Process View, and LV depLoyment View, M= UML Visual Model, C = RACI chart.

Example 1: The content of RSE[i], where i = LV, shows a visual model that shows a deployment view with a deployment diagram and a RACI chart that a system administrator is responsible for the deployment problem, a tester approves on the deployment work before the system is deployed, a system integrator is consulted, and others such as programmer, designer, and system analyst are kept informed.

However, the previous example in Section IV, the loan application process system, uses the SOC computing paradigm; we need to define the reverse software engineering in terms of two dimension – 4+1 views and service stake holders. The Service-Oriented Reverse Software Engineering matrix, SoRSE, is defined as follows:

Definition 2: For all i $\in$ {UV, DV, IV, PV, LV} and j $\in$ {SC, SB, SP} , SoRSE[i, j] = {Mj, Cj}, where UV = Use Case View, DV = Design View, IV = Implementation View, PV = Process View, and LV depLoyment View, SC = Service Consumer, SB = Service Broker, SP = Service Producer M = UML Visual Model, and C = RACI chart.

Example 2: The content of SoRSE[i, j], where i = LV and j = SB, shows a visual model that shows a service deployment view with a deployment diagram and a RACI chart that a system administrator is responsible for the service deployment, a tester approves on the deployment work before the composite service is deployed, a system integrator is consulted on service composition , and others such as service-oriented programmer, service designer, and system analyst are kept informed.

Therefore, the number of artifacts after SoRSE is equal to the number of service consumers, brokers, and producers. For the loan application process system, the SoRSE generates at least 5 models and charts: 1 for a service consumer, 1 for a service broker, and 3 for three service producers. Table 1 shows a comparison of RSE and SoRSE.

TABLE 1
A COMPARISON OF RSE AND SORSE

	Number of Models	Number of RACI Charts
RSE	1	1
SoRSE	Nc + Nb + Np	Nc + Nb + Np

Nc = Number Consumers, Nb = Number of Brokers Np = Number of Producers.

B. View Sequence in SoRSE and SoFSE

We visualize, document, and specify the 4+1 views by using UML diagrams. Table 2 shows the relationships between UML diagrams and each view.

TABLE 2
THE RELATIONSHIPS BETWEEN UML DIAGRAMS AND 4+1 VIEWS.

View	UML Diagram
Use Case View	• Use Case Diagram • An activity diagram per a use case
Design View	• Class Diagram • An activity diagram for a method
Implementation View	• Component Diagram
Process View	• Sequence Diagram • Collaboration diagram
Deployment View	• Deployment Diagram

However, the SOSR does not explicitly describe which view will be chosen over time during the reverse or forward software engineering process. In the Extended SOSR, we explicitly describe the orders of which view takes place, which is shown in Table 3. For example, SoRSE starts with the deployment view of the system. The deployment view shows the nodes that form the system's hardware topology on which the system executes. This view takes into account the distribution, delivery, and installation of the parts that make up the physical system. The existing system is analyzed as it exists and presented in a deployment diagram.

Secondly, one looks at the implementation view which encompasses the components and files that are used to assemble and release the physical system. This view primarily addresses the configuration management of the system's releases, made up of loosely coupled components and files that can be assembled in various ways to make a running system.

Thirdly, one looks at the design view which includes the classes and interfaces that make up the system. This view primarily supports the functional requirements of the system, meaning the services that the system should provide to its end users.

Fourthly, the process view is created next which takes into account the threads and processes that show the systems concurrency and synchronization mechanisms. This view addresses the performance, scalability, and throughput of the system.

Finally, the user looks at the use case view which encompasses the use cases that describe the behavior of the system as seen by its end users, analysts, and testers. This view doesn't specify the organization of a software system; rather, it exists to specify the forces that shape the system's architecture.

TABLE 3
VIEW SEQUENCE IN SORSE AND SOFSE

	SoRSE	SoFSE
Step 1	Deployment View	Use Case View
Step 2	Implementation View	Design View
Step 3	Design View	Process View
Step 4	Process View	Implementation View
Step 5	Use Case View	Deployment View

C. UML Modeling Guideline for a Service Broker

Next SOSR is enhanced by adding guidelines for the creation of UML diagrams for each view for the Service Consumer, Service Broker, and Service Producer [3][19]. These guidelines can be used by Software Developers, Architects, and Project Managers in the creation of SOSR modeling. These models are the blueprints of the SOSR software development methodology. In this paper, we focus on the roles of Service Broker using composite services. For each view the following modeling guidelines are proposed:

Deployment View
1) Use the UML deployment diagram
2) Node Instances for physical locations of components from the service broker perspective.
3) Components represent the top level components the user interacts with
4) Create directory structure of component's physical location in Implementation View and place the component in the correct place.
5) The '.wsdl' files should be shown and the location information of where they physically reside.
6) Show the type of connection between Nodes

Implementation View
1) Use the UML component diagram
2) Components represent the components and files that are used to compose the physical system
3) Components that the broker interacts or are used to achieve the broker's goals are represented. These include the '.wsdl' components that are used.
4) The process flow of the application is represented by arrows.

Design View
1) Use the UML class diagram
2) Create new folders called L1 Presentation, L2 Business, and L3 Data Access representing the 3-layered architecture of the application.
3) Create the classes with methods and attributes that the service broker interacts with or achieves the broker's goals

Process View

1) Use the UML sequence diagram
2) Use the "Object life line" components and show the "classifiers" from the design view
3) Use arrows to represent the process direction and method calls
4) Show the processes of the application from starting from the BPEL composite service and the order of the services that it calls.

Use Case View

1) Use the UML use case diagram
2) Represent the user's of the application as Actors and how they interact with the application.
3) Draw arrows representing the direction of the flow of interaction
4) The left side represents the front end users while the right side represents the back end users
5) The front end user is the broker and back end users are the service providers that the broker invokes.
6) The middle represents the actual BPEL process
7) Create an activity diagram to show the flow of the process.

D. RACI Chart for a Service Broker

RACI chart is enhanced to include Service Broker. The RACI chart explains the following roles: Responsible, Accountable, Consulted, and Informed. These roles are assigned to different participants and for specific tasks. Table 4 shows a typical RACI chart that does not include the service broker as at the time of RACI inception web services was not a viable option [18].

TABLE 4
A TYPICAL RACI CHART (MEIER ET AL., 2003)

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

By considering the sub-roles of a service broker, the RACI is revised. We add two new job titles such as Web Service Engineer (WSE) and Web Service Test Engineer (WSTE). The WSE's job includes creating or choosing of the web services employed by the project. Also, the BPEL process design, construction, and implementation can be done by the WSE. It has been decided that the WSE handles the BPEL process because BPEL is itself a web service: The BPEL composition creates a composite web service with its own WSDL file. The outside world only sees the WSDL file and doesn't care how it is implemented.

However, the WSE is concerned with the implementation and for this reason another job description was created called the WSTE. WSTE is tasked with the testing of web services. Since BPEL creates a web service and is composed of many other web services, it becomes difficult to track and find software bugs. BPEL services can be composed of other BPEL service which can become quite complex and exponentially difficult to test. The question is asked who is responsible for spurious results returned by the Web service. Is it the Web

TABLE 5
A TYPICAL RACI CHART FOR A SERVICE BROKER

Activities	View	PM	SA	UD	AR	WSE	WSTE
BPEL Architectural Design	Des	I		I	RA	C	I
Research Web Components	All	C	C	I	C	RA	I
Design Test Cases	All	I	C		I	C	RA
Architectural implementation	Des/Imp	I			C	RA	I
Implement needed services	Des/Imp	I	C		C	RA	I
Publish web services	Dep/Imp	I	R		C	A	I
Implement BPEL Process	Imp/Des	I	C			RA	
Deploy BPEL Process	Pro/Des	I	R			A	
Implement Test	All	I				I	RA
Perform Integration Test	All	I				I	RA
Evaluate Test	All	I	I			C	RA

ROLES-PM: PROJECT MANAGER, SA: SYSTEMS ANALYST, UD: USER INTERFACE DESIGNER, AR: ARCHITECT, WSE: WEB SERVICE ENGINEER, WSTE: WEB SERVICE TEST ENGINEER

service provider itself or is it the person who uses the web service? How well does one know whether the application was tested or not? The beauty of web services also becomes its greatest possible weakness. The abstraction and black box aspect of web services also can be dangerous in the fact that one does not know what is inside in a web service and how it works. This is where the WSTE, who is a white box tester and black box tester, looks at the inner workings of the web service and tests it to make sure it performs to specifications. Table 5 shows a RACI chart that includes the service broker.

VI. Reengineering of Legacy Application – Drug Recognition Evaluation System

In this section, the Extended SOSR is applied to the Washington State Patrol's Drug Recognition Evaluation (DRE) System. The Washington State Patrol department currently has a Microsoft Access based Windows form application that keeps track of drug evaluations performed by DRE officers. Drug evaluations are performed on drivers that are suspected of driving under the influence of drugs. A series of tests as well as personal information are documented on a hardcopy of a DRE form. The hard copy form is then sent to an office administrator of the Impaired Driving Section. Also, a blood sample is sent to the toxicology lab with the suspect's personal information. After the lab tests are performed at the toxicology lab, the hardcopy results are sent by mail to the office administrator of the Impaired Driving Section. The office administrator then enters all the information into the DRE MS Access application. This total process take at least three months and sometimes an indefinite amount of time as the paper trail gets broken and information lost. This inhibits data and information to be reported in a timely, accurate and usable fashion. This process needs to be automated and is perfect for a reengineering project using SOSR to be undertaken.

The DRE application exists on a shared drive and is unstable with limited functionality. Significant reports are not able to be generated and accessed over the internet. This application is being reengineered using ASP .NET and VB .NET technologies. The .NET technologies provide a stable and high performance environment that allows more data to be processed and a significant amount more information to be reported. These reports are available over the web to many different parties. This greatly enhances the ability of the patrol department in its fight against impaired driving.

This project is chosen in this research because there are many similar applications that exist in work places around the globe. With the advent of MS Access, an explosion of standalone desktop applications has occurred. As business needs change and growth occurs, many of these applications need to be reengineered to web based distributed systems that are performance driven and highly scalable.

The first phase in the reengineering of the DRE application is analyzing the existing legacy system and applying RSE. The Service-Oriented Reverse Software Engineering vector developed in the SOSR states that all i ϵ *{UV, DV, IV, PV, LV}*, RSE[i] = {M, C}. This results in a vector since it only has one participant. The artifacts produced are a UML visual model and RACI chart that are both created in terms of the 4 + 1 views.

In 4 + 1 modeling this one model contains 5 views. Using the view sequence for RSE developed in the SOSR, one starts with the deployment view to analyze the current state of the application. Figure 4 shows the deployment view that was developed for the legacy DRE software application. As shown in the diagram this application is a MS Access based windows form application that exists on a shared drive. This is highly unstable and shows that there could be performance and security issues involved. Also, the deployment view shows that the presentation, business and data access layers are not separated thus violating practices of good software design.

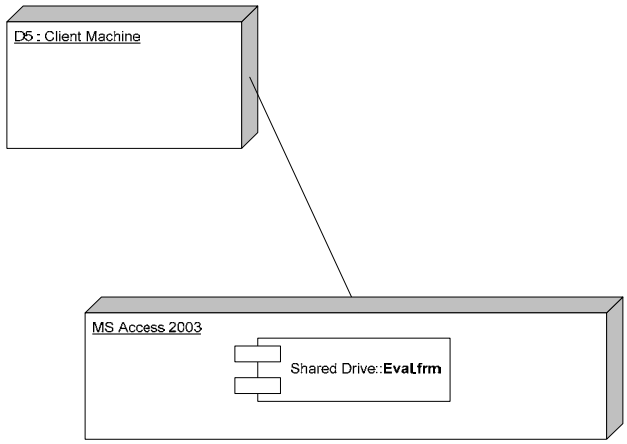

Fig. 4. DRE Legacy System Deployment View

Next, the following views are created in this respective order: implementation view, design view, process view and use case view. Once the model is developed, it can be analyzed to find ways of improvement and reengineering. For example, by looking at the use case diagram in Figure 5, one can discover that there are users involved in the business processes that are not even using the application. These types of manual processes are a good sign that reengineering needs to be done.

During the reverse engineering process RACI charts were created to define the roles and tasks of the stakeholders involved. Table 6 shows the RACI chart of the legacy system in terms of design view. The roles and task involved are typical of legacy application software development.

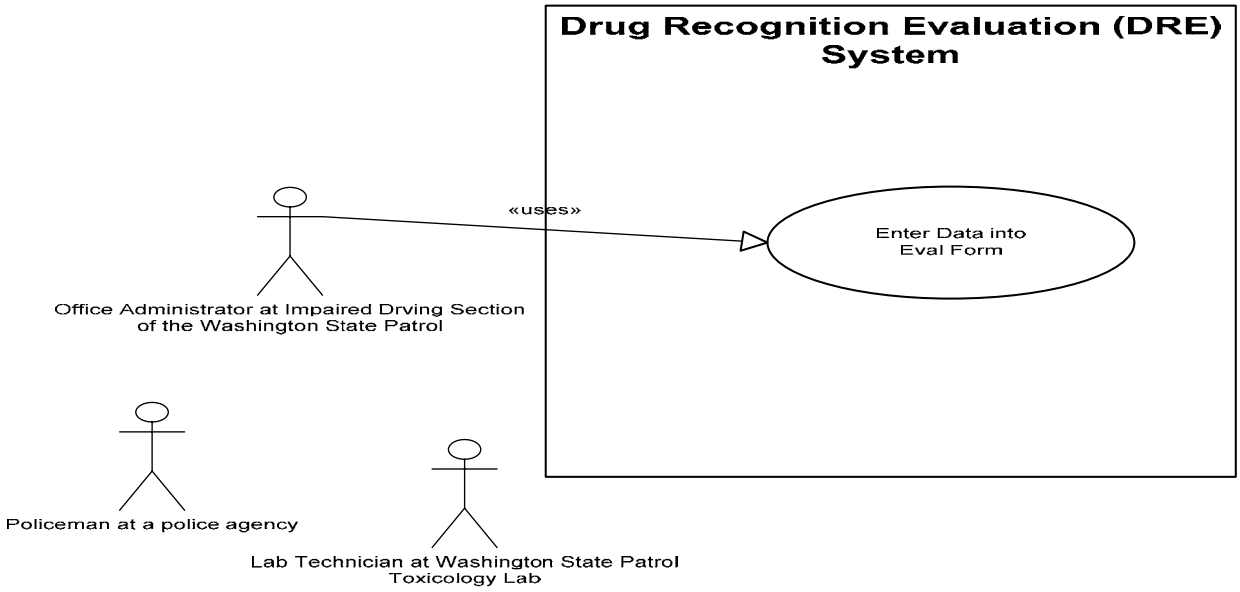

Fig. 5. DRE Legacy System Use Case Diagram

TABLE 6
RACI CHART OF DESIGN VIEW OF LEGACY SYSTEM

Activities	PM	SA	AR	SD	TE
Analyze Application	I	C	RA	C	I
Design Deployment Structure	I	RA	C	C	I
Design Database Structure	I	C	RA	C	I
Design Form UI	C		R	RA	C
Design API of Classes	I		C	RA	I
Design Methods	I		C	RA	I
Design Test Case	I		I	C	RA

ROLES-PM: PROJECT MANAGER, SA: SYSTEMS ANALYST, AR: ARCHITECT, SD: SOFTWARE DEVELOPER, TE: TEST ENGINEER

Once analysis is done on the legacy application, deliverables need to be developed for the new application in accordance with Extended SOSR. This involves creating the 4 + 1 view models and the RACI charts for the new application. The Service-Oriented Forward Software Engineering Matrix developed in Extended SOSR states for all $i \in \{UV, DV, IV, PV, LV\}$ and $j \in \{SC, SB, SP\}$, $SoFSE[i, j] = \{Mj, Cj\}$, This results in a matrix as there are three service stakeholders namely the service consumer, service broker, and service producer. For the DRE project, the result is a total of 4 models; one for the service consumer, one for the service broker, and two for the service producers.

User case studies reveal that the first area of input comes from the DRE officer who conducts the evaluation on the suspect and enters in the initial information in a form which generates a unique rolling log number. This will be done by officers from around the State and many different law enforcement agencies, not just the Washington State Patrol.

The second area of input is from the State Toxicology Lab which provides the Drug results. The drug results of the blood tests will be accessible through a web service using the rolling log number. The final area of input will be the data entry of the rest of the form which will take place in Olympia or Seattle. Web services will be implemented to allow the data entered in by the field DRE officers and the toxicology results from the State Toxicology Lab to be inserted into the final form. This process is ripe for BPEL to orchestrate the web services into a cohesive process.

This paper focuses on the service broker for BPEL composite services. The service broker generates four models each with 4+1 views. Using the view sequence for SoSFE developed in Extended SOSR, the first model developed is the use case view. Figure 6 shows the use case view that is created in terms of the broker using the guidelines created in Extended SOSR for the DRE application. This use case view will be used by the analyst to discover services from published services that could be used for the new application. In terms of the DRE application the analyst discovered that there was a web service under development by the toxicology lab that published toxicology results as a web service. The input for this web service was a rolling log number and the output was the drug result for the suspect.

Secondly, the process view is created in terms of the broker using the guidelines developed in Extended SOSR. The process view is used by the designer to describe discovered services and to compose a composite service using available services. For the DRE application one can see in this view that the service consumer initiates the process by calling the BPEL DRE process. The DRE process then calls the 'RollingLogService' and the 'ToxWebService' respectively.

These need to be constructed in this order as the unique license number generates the rolling log number which in turn is used by the 'ToxWebService' to return the drug result. The BPEL process returns back to the consumer the toxicology result for a subject suspected of driving under the influence of drugs. This process is shown in Figure 7.

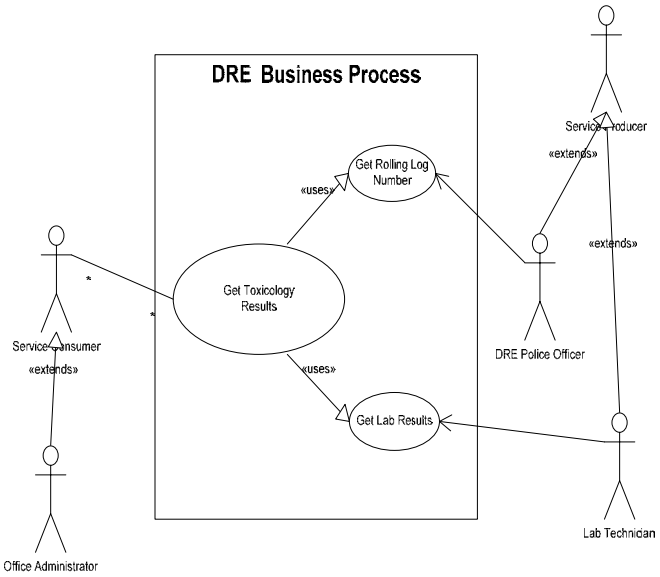

Fig. 6. DRE application use case view of service broker role

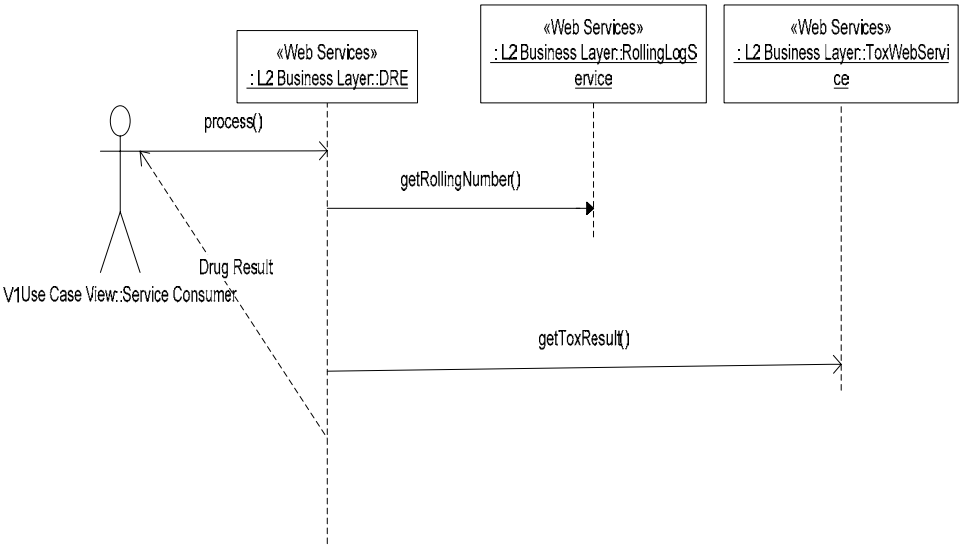

Fig. 7. DRE Application Process view of Service Broker Role

Thirdly, the design view is created for the broker using the guidelines in Extending SOSR. The design view is used by the designer to describe services and design composite services. Figure 8 shows the design view that was created for the DRE application. The design view shows that one method called process needs to be created for the BPEL composite service. The process method will start the BPEL process. The 'RollingLogService' has one method called 'getRollingNumber' that will retrieve a rolling log number given a unique license number. This web service called 'RollingLogService' will need to be designed and implemented as it was discovered in the use case view stage by the analyst that there weren't any web services available to accomplish this task. On the other hand, the analyst was able to determine that there was a service that could be used to return Toxicology results from the Toxicology Lab. According to the application documentation the 'ToxWebService' has one method called 'getToxResults.' This method returns toxicology results

returned from the 'getRollingNumber' method. The input and output for this web service will need to be integrated into the BPEL composite service.

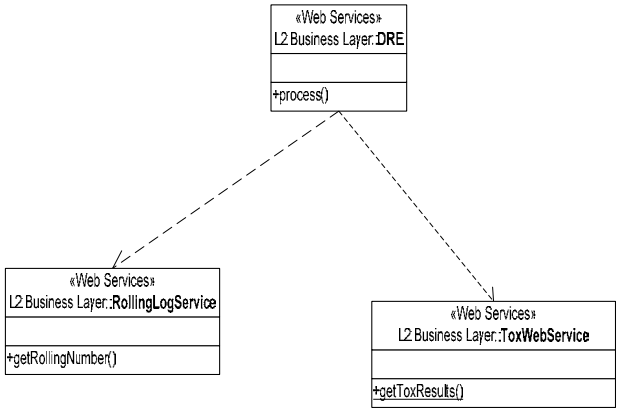

Fig. 8. DRE Application Design view of Service Broker Role

Fourthly, the implementation view is created for the broker according to the guidelines in the Extended SOSR. This view is used by the BPEL programmer to implement the application. Figure 9 shows the implementation view created for the DRE application. The programmer uses the implementation view to implement the program using the BPEL designer. This view shows the process that needs to be created and the web services that need to be called in the BPEL composite service. The process that is created is called the DRE.bpel. The BPEL Programmer uses Oracle JDeveloper to design the BPEL composite service and call the appropriate web services in the correct order in accordance with the business process shown in the implementation view.

Finally, the deployment view is created for the broker according to the guidelines in the Extended SOSR. Figure 10 shows the deployment view created in the DRE application. The system administrator uses the deployment view to deploy the composite service. The DRE composite service is deployed on the Oracle BPEL PM Server. The 'RollingLogService' and the 'ToxWebService' are also deployed on the 'localhost.'

The RACI chart for the broker is another deliverable in

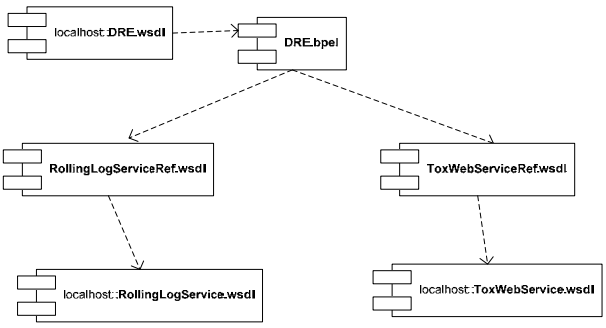

Fig. 9. DRE Application Implementation view of Service Broker Role

Extended SOSR reengineering methodology. Table 7 shows the RACI Chart for the design view for the DRE application. This RACI chart shows the roles and the tasks associated with the design view for the service broker. The table shows that the Architect and the WSE play a crucial role in the design of the BPEL composite service as well as the design and integration of the component web services. The WSTE also creates test plans for the resulting composite service and the component web services.

A successful DRE application was developed in accordance

TABLE 7
RACI CHART OF DESIGN VIEW OF DRE SYSTEM:

Activities	PM	SA	AR	WSE	WSTE
Analyze Process	I	I	RA		I
Research Web components	I	C	RA	C	I
Design Needed Web Services			R	A	I
Design API of Process			C	RA	I
Design Methods			C	RA	I
Design Test Case			I	C	RA

Roles-PM: Project Manager, SA: Systems Analyst, AR: Architect, WSE: Web Service Engineer, WSTE: Web Service Test Engineer

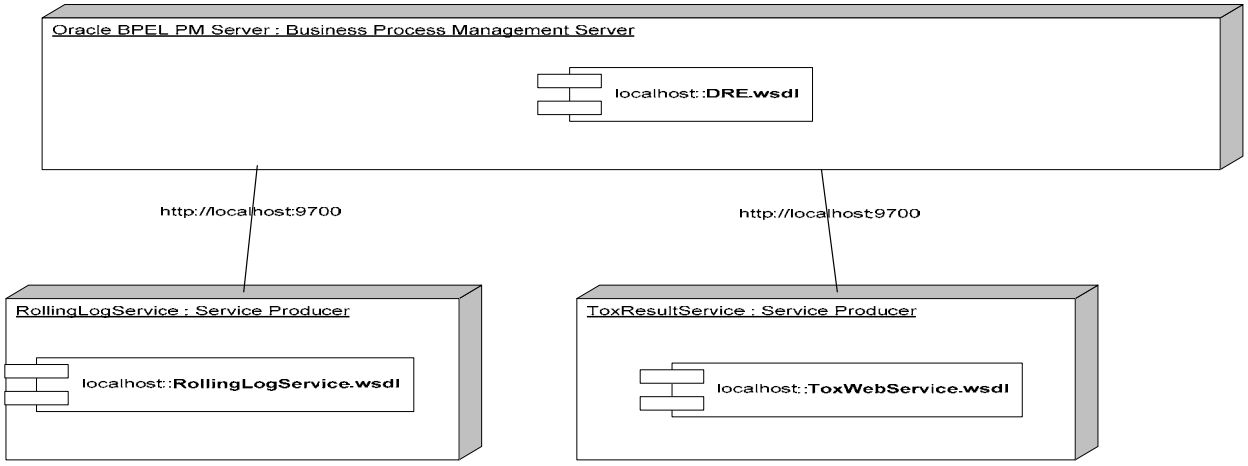

Fig. 10. DRE Application Deployment view of Service Broker Role

with extended SOSR and a BPEL process was implemented to orchestrate this process. The Figure 11 shows the final automated BPEL process that was created. This process eliminates the bottlenecks and human error that occur when hardcopies of documents are transcribed by humans and sent by mail. The automated process saves both time and money with results that can be updated virtually instantaneously.

In the final BPEL composite service, first a user enters a unique license number. The process then calls the first web service to retrieve the unique rolling log number for this license number. Then this rolling log number is used to retrieve the drug results from the toxicology lab via web service. Finally, the toxicology drug result is returned to the BPEL process for that particular subject and populates the evaluation form that will be finished by an office administrator in the Impaired Driving Section of the Washington State Patrol.

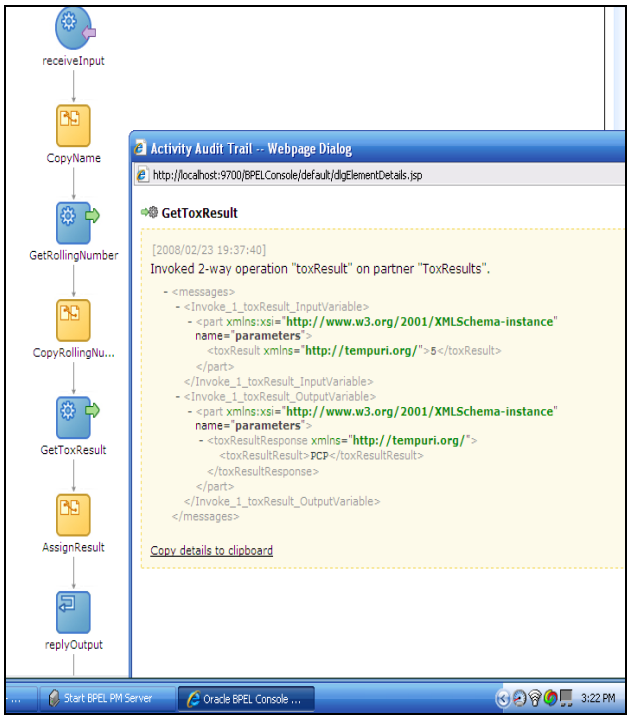

Figure 11: DRE BPEL composite service

VII. CONCLUSIONS AND FUTURE RESEARCH

Software design composed of loosely coupled web service components is an emerging software development paradigm. BPEL engines provide an environment to develop software as a set of Business Processes comprised of web service components. This type of loosely coupled architecture provides a flexible, efficient, and time effective way of software development. Business process needs drive business decisions and BPEL process engines meet these needs head on with the help of Extended SOSR

In this paper, we first applied the SOSR methodology to the modernization of a legacy application that does not use web services and gained some key experiences. Based upon those experiences, we extend the SOSR methodology to include guidelines for a service broker who composes a set of services as a composite service(s), deploys, and executes the composite services. The results of this research will help a software developer to reengineer a legacy system to a target system using composite services.

The Extended SOSR has several strengths compared to the current SOSR. First of all, it proposes Service-Oriented Reverse Software Engineering in terms of both 4+1 views and three service stakeholders. Secondly, the sequence of views in reverse and forward engineering processes is clearly defined. Therefore, the UML modeling sequence is also very clear. Thirdly, guidelines of modeling for a service broker are also presented. Lastly, the sub-roles of the service broker and their tasks are managed through a proposed RACI chart.

However, the Extended SOSR needs to be further researched and improved. Firstly, this Extended SOSR needs to be applied to more complex and diverse composite services and the outcomes need to be discussed. Secondly, view correspondences need to be researched in addition to the view sequence. Thirdly, the sub-roles of each service stakeholder need to be explored to bring unified efforts for RACI charts. Finally, a comparison of business process engines needs to be done as BPEL development products mature.

REFERENCES

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

[2] *Konrad Pfadenhauer, Schahram Dusdar, and Burhard Kitt. Anane, Rachid & Tsai (2005) "Comparison of Two Distinctive Model Driven Web Service Orchestration Proposals" Vienna University of Technology, Vienna, Austria*

[3] *Harrison, William & Barton, Charles & Raghavachari, Mukund (2000) "Mapping UML Designs to Java" ACM ISBN 1-58113-200-x/00/0010*

[4] *Hull, Richard & Su, Jianwen (2004) "Tools for Design of Composite Web Services" Presented at SIGMOD 2004 June 13-18, Paris, France. ACM ISBN 1-58113-859-8/04/06*

[5] *A. Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, C. K. Liu, S., Thatte, P. Yendluri, and A. Yiu, editors. "Web Services Business Process Execution Language (WS-BPEL) Version 2.0", 2005.*

[6] *N. Milanovic, M. Malek,"Current Solutions for Web Service Composition." IEEE Internet Computing, Nov.-Dec. 2004, Volume: 8, Issue: 6, p. 51-59*

[7] *Sam Chung, Joseph Byung Chul An, Sergio Davalos (2007). Service-Oriented Software Reengineering: SOSR. The mini-track "Technology and Strategies for Realizing Service-oriented Architectures with Web services" of the 40th Annual Hawaii International Conference on Systems Sciences (HICSS 2007). January 3-6, 2007. Waikoloa, Big Island Hawaii.*

[8] *Dan W. Henricks and Sam Chung. (2006). Latency and Bandwidth Comparison of Distributed Middleware Technologies for Broadband Communication Systems: RMI, Mobile Objects, and Web Services. The Second International Symposium on Broadband Communications (ISBC 2006). September 10-14, 2006. Moscow & St. Petersburg, Russia. (3 pages)*

[9] *Booth, D., Champion, M., Ferris C., McCabe, F., Newcomer, E., & Orchard, D. (2003). Web services architecture (WSCA 1.0), W3C Working Draft. http://www.w3.org/GR/we-arch/*

[10] *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*

[11] *Weerawarana, S., vurbera, F., Leymann, F., Dtorey, t., Ferguson, D. (2005). Web Services Platform Architectures. Upper Saddle River, NJ: Prentice Hall.*

[12] *Casati, F., Shan, E., Dayal, U., and Shan, M. 2003. Business-oriented management of Web services. Commun. ACM 46, 10 (Oct. 2003), 55-60.*

DOI= http://doi.acm.org.offcampus.lib.washington.edu/10.1145/944217.944238

[13] Curbera, F., Khalaf, R., Mukhi, N., Tai, S., and Weerawarana, S. 2003. The next step in Web services. Commun. ACM 46, 10 (Oct. 2003), 29-34. DOI= http://doi.acm.org.offcampus.lib.washington.edu/10.1145/944217.944234

[14] Mukhi, N. K., Konuru, R., and Curbera, F. 2004. Cooperative middleware specialization for service oriented architectures. In Proceedings of the 13th international World Wide Web Conference on Alternate Track Papers &Amp; Posters (New York, NY, USA, May 19 - 21, 2004). WWW Alt. '04. ACM Press, New York, NY, 206-215. DOI= http://doi.acm.org.offcampus.lib.washington.edu/10.1145/1013367.1013401

[15] Shan, T. C. and Hua, W. W. 2006 "Service-Oriented Solution Framework for Internet Banking" International Journal of Web Services Research; Jan-Mar 2006; 3, 1; ABI/INFORM Global, p.29

[16] James Gordon, Sam Chung, Sergio Davalos (2006) "Comparisons of Model-Driven Web Services Composition and Execution Engines."University of Washington Tacoma, Capstone Project Paper.

[17] Scientific Research Corporation "2007 Comparative Research Methodology Process For BPEL" http://www.active-endpoints.com/documents/documents/1/2007-comparative-assessment-for-BPEL.pdf

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

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

WEBSITES

[20] Microsoft. Biztalk Retrieved March 4, 2008. http://www.microsoft.com/biztalk/default.mspx

[21] Oracle. Process Manager Retrieved March 4, 2008 http://www.oracle.com/technology/products/ias/bpel/index.html

[22] Sun Microsystems. Retrieved March 4, 2008 http://developers.sun.com/jsenterprise/nb_enterprise_pack/reference/techart/bpel.html

[23] IBM. Retrieved March 4, 2008 http://www.ibm.com/developerworks/library/specification/ws-bpel/

[24] Active Endpoints, Open Source Project. Retrieved March 4, 2008 http://www.activevos.com/community-open-source.php