J. Basic. Appl. Sci. Res., 2(12)12246-12254, 2012

ISSN 2090-4304
Journal of Basic and Applied Scientific Research
www.textroad.com

From Modules-to-Services: A Conceptual SOA Migration Approach Based on QoS

Mehrosh Khalid, Farooque Azam, M. Aqeel Iqbal

Department of Computer Engineering,
College of Electrical and Mechanical Engineering,
National University of Sciences and Technology (NUST), Islamabad, Pakistan

ABSTRACT

Even though Service-Oriented Architecture (SOA) has turn out to be accepted in the past few years, the vast majority of legacy architectural systems at a standstill are not modularized to work in an SOA environment. The rich quantity of information that businesses have to manipulate has generated a substantial raise in the complication of the legacy systems that accumulate this information. Whereas heading towards a service-oriented architecture working environment can facilitate in managing this raise, as well as along with, it is vital to conserve the venture of several years of fine-tuning and debugging of the legacy possessions as much as achievable. Transforming legacy architectures toward service-oriented architectural frameworks is a tough job being complex due to the unavailability of suitable approaches and tools. In this paper, a step-wise migration approach is projected to transform legacy architecture into an corresponding architecture composed of a set of services and the implementation of that architecture. The proposed framework, in its every step, takes into consideration part of the existing legacy application and back-track it to its origin to visualize the legacy architectural environment, enabling migration into services in the similar working business environment. In the proposed migration framework, much emphasis is put on the testing, since after each migration step the artifacts are tested to ensure QoS of the new service-oriented application being developed and architectural conformance to the original version.
KEY WORDS: SOA, QoS, Horseshoe Model, Legacy, Services, Reverse Engineering, Transformations, Forward Engineering.

I. INTRODUCTION TO SOA

Service-Oriented Architecture (SOA) is a way of scheming, mounting, deploying, and administering systems that are described by reusable services and service patrons. The services symbolize reusable system functionalities, all the way through customary interfaces; service consumers create applications or systems that make use of those services [1].

Service-oriented architecture (SOA) can be considered as an architectural build for easy-to-use association of distinguished components in reaction to varying needs in businesses. SOA emphasizes the trade of information amongst foremost software assets and the means by which these software assets can be reused. A vast majority of benefits of using SOA that lead businesses to step towards migrating legacy system to SOA, includes less functional coupling amongst modules, generalization of core logic, agility, flexibility, reusability, autonomy, statelessness, discoverability and compact costs. The key here to adapt SOA is to progress towards how businesses communicate so that the objectives of the venture can be more willingly accepted [2].

The proposed framework is based on SOA migration horseshoe model [1] and builds QoS measures in the transformation process.

II. Soa building blocks

What is a Service?

Services are reusable gears that symbolize business responsibilities. A Service can be disseminated worldwide across businesses and organizations and can be easily re-configured to map to new business requirements. Services act reusable by means of businesses that many different business processes can use the same service. [3].

SOA is a manner of building up systems being comprised of services that are instantiated in a customary way. In SOA architectural style, an SOA-based system is comprised of services, applications that utilize those services, and an SOA infrastructure that bridges applications to services [3].

SOA Operations:

SOA-based system streamlines three main operations:

1. **Service discovery** -> Services are searched within the service repositories with preferred features. The service repository may be a single simple service directory where services are organized according to their type or it can be a more composite

***Corresponding Author:** Mehrosh Khalid, Department of Computer Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology (NUST), Islamabad, Pakistan, mehrosh.khalid@yahoo.com

registry in which services are categorized by pre-defined domain-specific knowledge bases with QoS information. All the details about available and accessible services are stored in a service registry.
The foremost challenges of discovering services are the suitable portrayal and the protection of the service registries.
2. **Service composition -> *Services*** are incorporated into applications to endow with segments of required functional requirements. The main risks associated with service composition are input/output conversions and transaction management.
3. **Service invocation ->** It deals with the invocation of services and then the resultant service code is executed. There are many different ways by which services can be called upon:

- By service consumer directly
- By discovery services that locate and return the location of required services to service consumer to be invoked.
- By service brokers employed by service consumers through which the demands for services are passed to one or more service discoveries.

The foremost risks associated with service invocation are trading with service availability and encompassing exhaustive exception handling techniques to deal with situations when the required services are not more present and accessible.

SOA Framework:
A Service Oriented Architectural framework comprises of:

1. Services
2. Applications that search for and make use of services
3. An SOA infrastructure that helps communicates applications with services at an extreme abstract level

As depicted in Figure 1, in this perspective, standard means of communicating applications with the services are being facilitated by the SOA Infrastructure. Every application calls for the services by a similar mechanism. An interface, being provided by every service, is invoked by means of a proper data format and protocols being understood by all the prospective users of that service.
In this context expertise of the following people are required in their respective domain:

a) **Infrastructure Developers** emphasize on providing a well-established and secure infrastructure having standards, common services and development tools.
b) **Applications Developers** focus on the detection, composition and static or dynamic instantiation of services.
c) **Service providers** work for the portrayal and granularity of services so that applications can simply trace and utilize them with up to standard Quality of Service (QoS).

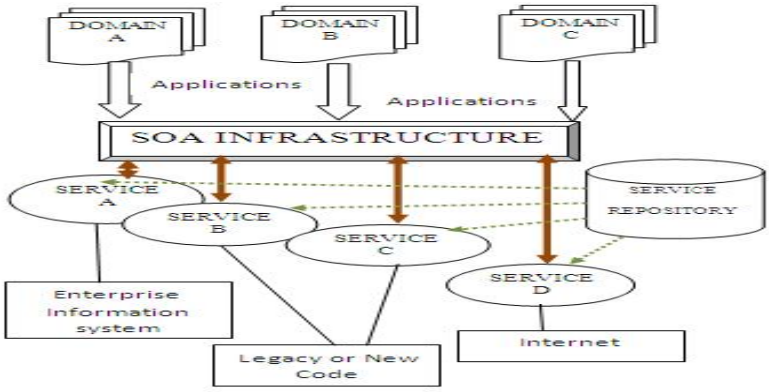

Figure-1: SOA Framework

The task categorization as discussed in [2] is summarized as:

TASKS		
Infrastructure Developers	**Application Developers**	**Service Providers**
✓ Selecting standards for implementation ✓ Developing a set of general infrastructure services for discovery, communication, security, etc. ✓ Developing binding approaches to satisfy potential service users. ✓ Developing tools for development. ✓ Developing documents to support the infrastructure	✓ Discovering appropriate services for applications ✓ Analyzing service description documents. ✓ Instantiating discovered services in applications ✓ Testing services to ensure that application perform accurately by using those services.	✓ Identifying requirements of potential service users ✓ Developing code that manipulates with the services ✓ Publishing the service ✓ Developing service initialization code and operational procedures.

Table-1: Task Categorization

III. Need for migration to soa

Widespread objectives for embracing the SOA are to eradicate redundancy, bring together novel functionality from existing services, adapt systems to varying needs, and leverage legacy investments.

Eradicating Redundancy

It is frequently discovered that in conventional systems, often the same functionality is found across systems. For example, many different applications might support to deal with their customer databases. A service with corresponding functionality can be put into action and used by all the different applications. This renders into cost-effectiveness for the reason that changes and maintenance is done with only one service, and many different applications will use the same service, that have need of this core functionality with no modifications.

Bringing Together Novel Functionality

Consider an Online Shopping application that deploys a group containing related services to execute its functionality. If the business makes a decision to step into another new business, then many of the existing services already being used by the online shopping application would be on hand, such as customer management and transaction management. The business could progress more rapidly with new innovations.

Adapting Systems to Varying Needs

The capacity to adapt to varying needs is for the reason that in an SOA approach, applications access services in a customary approach all the way through the chosen SOA infrastructure. Hence, the coding logic underneath the services can be modified as considered necessary without having a consequence on existing applications, provided the service interface is being unchanged.

Leveraging Legacy Investments

As a final point, an SOA modeling approach is an appealing decision to depict functionality in legacy systems. In SOA approach, SOA infrastructure and service interface provides access to the legacy functionality and not the application. This leads to the consequence that the legacy platform diversity is crystal clear to the applications.

IV. Existing soa migration approaches

Renovating legacy system applications with SOA services permits systems to continue for the most part unaffected despite the fact that legacy functionality reveals to a huge figure of users by means of well-defined service interfaces. Migration of a legacy system to SOA, e.g. wrapping Web Services, could be somewhat clear-cut. Though, distinctive features of legacy systems as programming language, system architecture and platform may surprisingly confuse the job. This situation may arise while migrating to an extremely challenging SOA by depicting the rich content of the legacy system and creation of services [4].

Some SOA migration approaches as revealed from the literature study are discussed as under:

SEI at CMU, as result of a project, developed a preliminary approach known as Service-Oriented Migration and Reuse Technique (SMART). This approach deals with the identification and analysis problems in being faced in the SOA migration process [5]. It assists businesses evaluate legacy systems to conclude whether their functionality can be realistically interpreted as services in an SOA. SMART collects extensive information on the subject of legacy components, the target SOA, and prospective services. It seems that this approach is platform/vendor/tool independent and it focuses on integrating the services only at middle tier. But, this approach does not map QoS issues.

Salvaging & Wrapping Approach (SWA) anticipated by Sneed [6] entails a tree-step practice for generating Web services from legacy code. These are salvaging the legacy code, wrapping the salvaged code and producing the code available as a Web service. After code re-engineering automatic code extraction is performed by using a tool, and afterwards the legacy code is wrapped in the rear of an XML shell to present Web services to the service clients. It can be a helpful approach but it does not deal with the QoS issues.

Ziemann et al [7] portray a business determined legacy-to-SOA migration approach (EMDMA) by means of introducing an simple process model among the business functions and the legacy system. Afterwards, legacy business process models are converted to the SOA process model. EMDMA illustrates consideration to the actuality that many of the characteristics of legacy systems like functional granularity, security, reliability, etc. are not considered effectively. This approach places important consideration to the dire need for exploring how particular features of legacy systems (reliability, scalability, policies) can be conformed to SOA environments.

Oracle proposes Oracle Modernization Framework (OMF) as the migration approach [8]. The OMF migration technique is strongly supported by a tool suite, “ORACLE SOA SUITE” [9]. The proposed framework first allows the businesses to have a thorough analysis of their application and then help businesses to choose amongst the transformation techniques as re-architecting, enablement, re-hosting, automated migration or COTS replacement.

Winter and Ziemann in 2007 also proposed a model “SOA-Migration Horseshoe” [10][1]. Their approach incorporates software reengineering principles and techniques with business process modeling and focuses on using reverse engineering methodologies that help extort a Legacy Enterprise Model from the legacy code. Then, forward engineering principles are used to identify services and map them. Ultimately, the mapped services are wrapped up to make components.

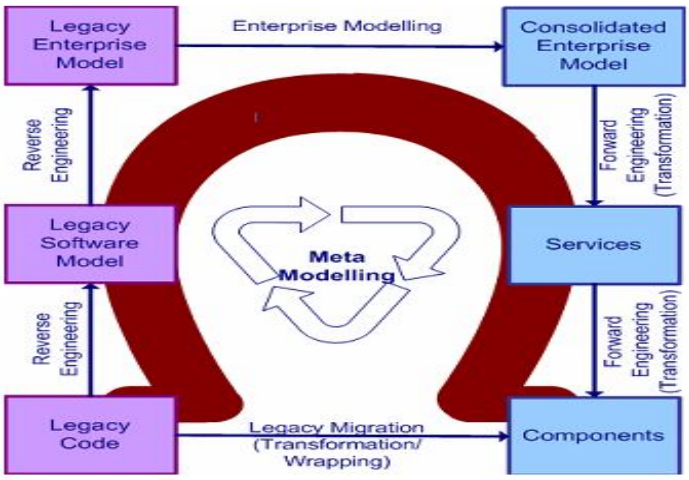

Figure-2: SOA-Migration Horseshoe [10]

V. Challenges of soa migration:

On hand approaches primarily claim context sensitive solutions for migrating legacy applications to service orientation. In fact, they suggest simply back-end integration based on the reverse engineering and architectural reconstruction of source codes. Such context sensitive endeavors, however, are not realistic enough for enterprise scale legacy applications and they cannot be widespread effortlessly [4].

Consequently, the resettlement of legacy systems into SOA environments is a foremost exploratory challenge by means of a considerable cost-effective impact. Regardless of key research efforts in the field of legacy reengineering and SOA, contemporary approaches barely emphasize on the technical migration itself and do not adequately reveal business desires particularly QoS [4].

The perfect reengineering method would be one that implements the complete SOA-Migration Horseshoe. The dilemma is that at hand there are approaches and automated tools that implement only portions of the horseshoe. Vital locale of desirable study is the development of a real process to put into practice the horseshoe and the development of tools (or suites of tools) to maintain that process. Furthermore, the computerization of this process would be worth examining, although extremely difficult [1].

VI. PROPOSED MODEL

As discussed in the previous section, the major challenges of SOA migration strategies are mainly lack of adequately measuring QoS (Quality of Service) of the services [4] and implementing the complete SOA-Migration Horseshoe [1].

The SOA-Migration Horseshoe model as depicted in Figure 3 is used to identify the phases, processes and artifacts of the model to implement it in its full essence.

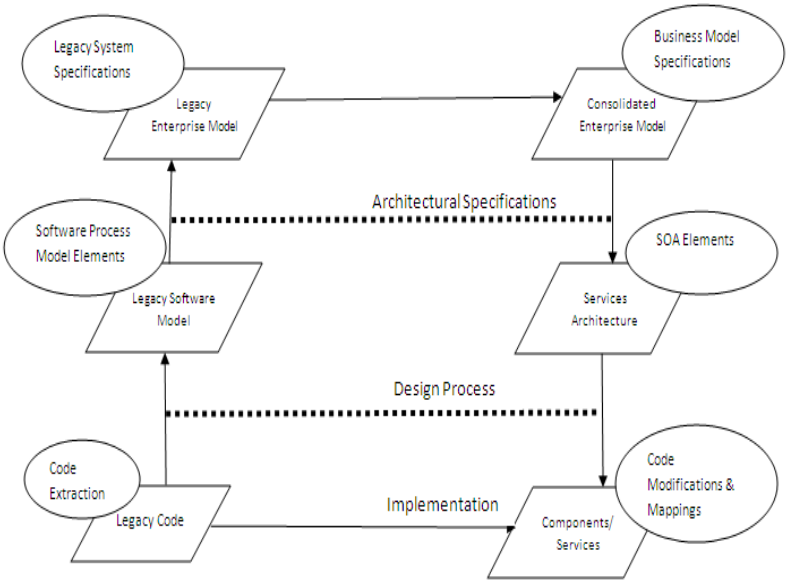

Figure-3: Proposed Life Cycle Model of SOA-Migration Horseshoe

The proposed life-cycle model consists of mainly three phases Architectural Specifications, Design Process and Implementation phase. Basically the Horseshoe model works as

Reverse Engineering -> Transformations -> Forward Engineering

Reverse Engineering: From the code units we back track the architectural elements and the legacy model requirements specifications to get an idea of the working environment and needs of the model

Transformations: Basically the process of forward engineering and transformations go parallel, as at each step we will transform artifacts of the phase into the required SOA environment. The Legacy System specifications will be transformed into the business process model specifications as a result of the Architectural Specifications phase. Then as an output of the design phase the Legacy model elements will be transformed into the SOA elements. Finally, after the implementation phase extracted code portions will be mapped into the services coding units.

Forward Engineering: While in reverse engineering, we gained an in-depth knowledge of the existing legacy system, in this step we will start from specifications and map legacy elements into SOA models through design to implementation.

Quality of Service Requirements:

Sam Guckenheimer introduced the term "Quality of Service Requirement" as an alternative to 'nonfunctional requirement' or 'quality attribute' for Agile Software Development. Nonfunctional requirements, quality attributes, and quality of service requirements are the synonyms for the restrictions being imposed on functional requirements such as performance, security, and platform. Many other types of quality of service requirements certainly exist but much attention is given frequently to performance, security and platform. The significant facet of a quality of service requirement is that it does not illustrate functionality but it restricts other functionality. Quality of service requirements holds the maximum influence on the architecture. A good architecture will smooth the progress of easier delivery of quality of service requirements [12].

The legacy to SOA migration introduces a lot of well-known advantages (e.g., reusability and flexibility) but to achieve it successfully, a substantial amount of testing effort is needed. Testing is required to be applied at each phase and for each artifact produced as a result of different phases in SOA migration Horseshoe model, to ensure Quality of Service at every step.

The legacy to Service Oriented Architectural migration that is being proposed is extremely unique from the existing migration approaches explored customarily in the literature. Numerous methodologies [1, 2, and 4] suggested revealing some parts of the original (legacy) system as Web services. This can be achieved by introducing a new software layer called "wrapper". The wrapper helps representing some system functionalities to be accessible and reusable for other software systems and consumers. Via implementing a wrapper, the system is supplemented by way of an extra piece of software but its inner architecture remains unchanged. Basically, the architectural migration requires altering the internal configuration of the original system leading towards an "absolute" transformation of the original system [11]

The proposed model instead of using the wrapper technology, fully transforms the legacy architectures into service-oriented architectures using the SOA migration horseshoe model. To achieve Quality of Service, it is recommended that at extensive testing must be introduced throughout the migration process.

Step-Wise SOA Migration Approach:

In order to implement the full SOA migration Horseshoe model, I propose a detailed and comprehensive step-wise approach. Figure 4 depicts all the steps as being implemented in the SOA Migration horseshoe model. The dotted line being ended with a diamond symbol shows mapping transformations from the legacy components to the SOA components. The diamond symbol represents that at each step if the mappings are not accurate; it is possible to back-track to the associated originator process in the legacy architectural system for adaptations and enhancements.

- **Extraction of Code Segments:**

  This step focuses on the detailed study of the code that will lead towards identification of the candidate code that can be transformed into services. It means that this step will extract the code aimed to implement a particular functionality that can be mapped in form of individual service. This step needs an extensive understanding of the programmable code and all the depending programming units.

- **Unit Testing:**

  After the extraction of the function modules, each module is tested exhaustively to ensure its correctness. Unit testing is performed at this stage to make sure that modules can work independently or not. Different non-functional properties such as performance, reliability and integrity are assured at this stage.

- **Identification of Legacy System Elements:**

  To get a deeper insight into the legacy system, it is required to gain in depth knowledge of the legacy system architecture. If the supporting documents like design documents, are not being provided, then from the understandability of the code, it will be easy for the architects and developers to identify the system elements, their relationships and dependencies.

- **Integration Testing:**
  After the legacy system elements have been identified, integration testing is performed to check the dependencies amongst the various system elements. This step is must required to make sure that what kind of relationships exists between the different system elements.
- **Understanding Legacy System Specifications:**
  When the legacy system elements have been properly tested, it is then required to back-track the system specifications to visualize that for what kind of particular requirements the legacy code was written. This step will be quite helpful to idealize the application environment, the need of the particular legacy application and its advantages and faults to the intended audience.
- **System Testing:**
  Consequently, it is required to perform a rigorous system testing to ensure all the identified modules conform to the architectural elements according to the required legacy system environment. It will help the architects to find out that whether the particular functional modules extracted in the first step accurately conform to the user requirements or not.

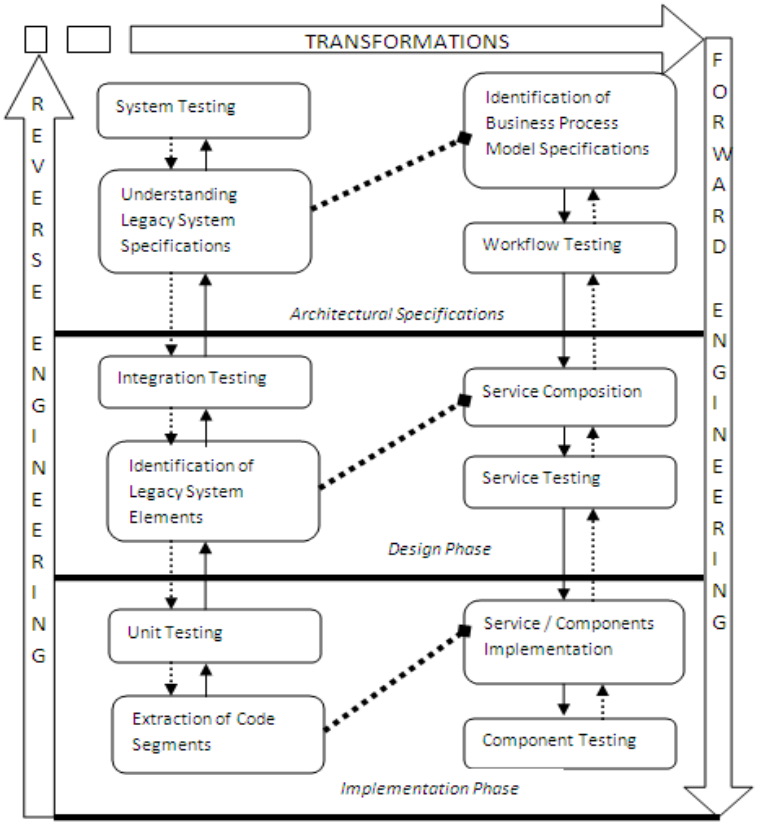

Figure-4: Step-wise approach towards SOA Migration Strategy

Till now, the steps were performed as a process of reverse engineering, that with the help of the code, architects will be able to identify the legacy application specifications and requirements. At every step, an exhaustive testing strategy is applied to make sure the accuracy of each step.

The further steps will now transform the legacy system architecture to service-oriented architecture.

- **Identification of Business Process Model Specifications:**
  As an end result of all the steps of the reverse engineering process, the legacy system architectural specifications had been mapped and tested properly. If the testing results are error free, similar business process model specifications are modeled and analyzed in the context of the Service-Oriented architectures. This process will lead to identify the service producers and service consumers and the need for the creation of services.
- **Workflow Testing:**
  After the identification of the requirements of the business process model, work flow testing is performed to ensure the proper flow of information between service producers, service consumers and service repositories. This testing analyzes the performance, security and availability sequencing and business logic requirements. If the testing will

generate some kind of non-compliances, it is possible to go-back and streamline the business process model specifications accordingly.

- **Service Composition:**
  When the workflow testing ensures accuracy of the workflows amongst the components of the SOA framework, service components are created which will be mapped from the legacy software model processes. If some problems occur we can back track to the legacy software process model to ensure compliances.
- **Service Testing:**
  To ensure proper service composition service testing is performed to ensure proper working of the services and their compliance with the SOA standards, ensuring mappings of service compositions to security, performance and reliability. This phase will also ensure the accuracy of service-level interactions. Service Level testing must conforms the adherence of the services to only the requirements of the ongoing project, but also to the business and operational requirements of other consumers of the same service.
- **Service / Component Implementation:**
  Now at the end, by getting all the required information, this step will transform the legacy code into the services code. As the model progresses through all the steps, there are very rare chances to re-write the original code. Because the model has identified the legacy architectural elements and environment specifications which are mapped into the business process model elements and specifications, the legacy code can be easily transformed in the form of services.
- **Component Testing:**
  Component testing corresponds to the unit testing strategy that will ensure the proper working of each and every component individually. The testing is performed to ensure the successful compilation of the code plus the assurance that the components and functions of the service work accurately as specified.

Proposed SOA Migration Framework:

As discussed in the previous section, the step-wise approach towards SOA migration is aimed at fully implementing the SOA migration horseshoe model [1]. Figure 5 illustrates the complete SOA migration framework.

As depicted in Figure 5, the legacy components and architectural elements are identified and analyzed to ensure either they are worth transformable. If so, then these architectural components and legacy codes are transformed into SOA components. Table 2 gives an overview of the phases of the SOA migration framework.

SOA MIGRATION FRAMEWORK	
PHASES	**DESCRIPTION**
Architectural Specifications	Aims to identify the system specifications, values and requirements
Design	Creates architectural components and produces a detailed architectural design
Implementation	Implements the design elements in programming languages
Testing	This phase spans throughout the framework and helps to ensure accuracy of activities at each step

Table-2: Phases of SOA Migration Framework

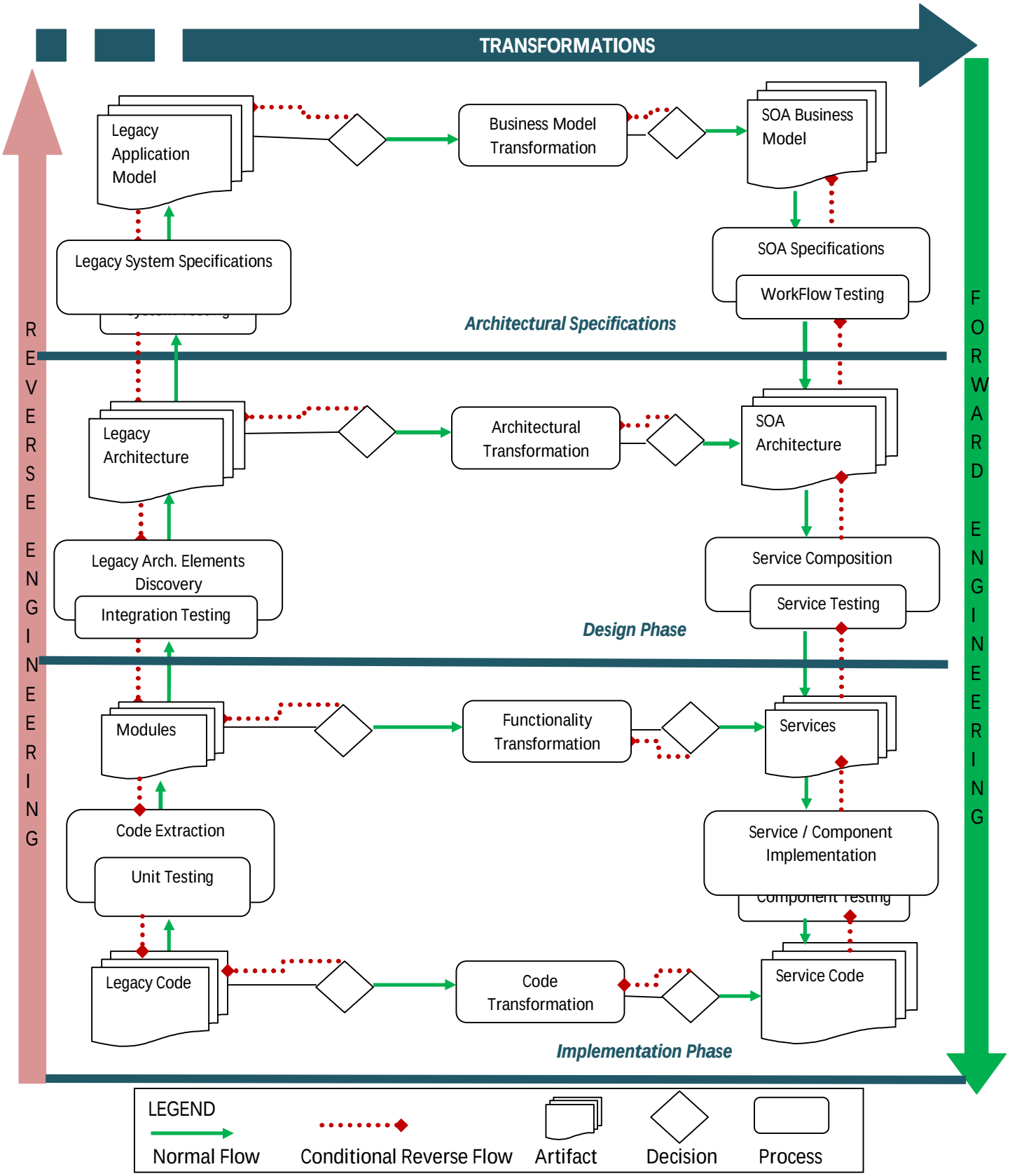

Figure-5: Proposed SOA Migration Framework

VII. Conclusion

In the past few years, many legacy-to-SOA migration approaches have been developed, each of which emphasizes on particular perspective of SOA migration. This paper aims to present a legacy-to-SOA migration framework that tries to fully implement the SOA migration horseshoe model [1] based on applying QoS. The proposed framework facilitates to characterize and isolate the properties of migration approaches in terms of processes it supports, artifacts produced as result of processes and decisions based on testing results. The proposed framework is based on a step-wise approach. Testing is central to the proposed approach to ensure Quality of Service as before heading towards next step, testing makes sure all the conformances and compliances. The proposed framework basically works on the same dimensions as though the SOA migration horseshoe model works, that are, Reverse Engineering (from legacy code, legacy architectures are extracted and tested), Transformations (Legacy Architectural elements are mapped to SOA working environment), Forward Engineering (following the SOA specifications, services are designed and implemented after being fully tested).

VIII. Future work

As future work, we will use the SOA migration framework in business organizations to elicit how businesses perform SOA migration in practice. Furthermore, we are planning to develop computerized tools that will automate SOA migration framework.

REFRENCES

[1]. Grace A. Lewis, Dennis B. Smith, Kostas Kontogiannis, “A Research Agenda for Service-Oriented Architecture (SOA): Maintenance and Evolution of Service-Oriented Systems”, March 2010 TECHNICAL NOTE CMU/SEI-2010-TN-003

[2]. Asil A. Almonaies, James R. Cordy, and Thomas R. Dean, “Legacy System Evolution towards Service-Oriented Architecture”, 2010, School of Computing, Queens University Kingston, Ontario, Canada, {almonaies,cordy,dean}@cs.queensu.ca

[3]. Grace Lewis and Dennis B. Smith, “Developing Realistic Approaches for the Migration of Legacy Components to Service-Oriented Architecture Environments”, Proceedings of the 2nd international conference on Trends in enterprise application architecture, 2006, Carnegie Mellon University, Software Engineering Institute, 4500 Fifth Ave., Pittsburgh, PA 15213, USA {glewis,dbs}@sei.cmu.edu

[4]. Semih Cetin, N. Ilker Altintas, Halit Oguztuzun, Ali H. Dogru, Ozgur Tufekci and Selma Suloglu, “Legacy Migration to Service-Oriented Computing with Mashup”, Cybersoft Information Technologies, Department of Computer Engineering, Middle East Technical University, ICSEA 2007

[5]. SEI, “SMART: The Service-Oriented Migration and Reuse Technique”, (CMU/SEI-2005-TN-029), 2005.

[6]. Sneed H.M., “Integrating Legacy Software into a Service Oriented Architecture”, CSMR’06, IEEE CSP, 2006, 3-14.

[7]. Ziemann, J., Leyking, K., Kahl, T. and Dirk, W.: "Enterprise Model driven Migration from Legacy to SOA", Software Reengineering and Services Workshop, 2006.

[8]. Oracle, "Oracle IT Modernization Series: The Types of Modernization", Oracle White Paper, 2006.

[9]. Mohammed, F., "Oracle SOA Suite", Sys-Con XML Journal, 2007.

[10]. Winter, A. & Ziemann, J. “Model-Based Migration to Service-Oriented Architectures.” Proceedings of the International Workshop on SOA Maintenance Evolution (SOAM 2007), 11th European Conference on Software Maintenance and Reengineering (CSMR 2007), Amsterdam, the Netherlands, March 20-23, 2007. IEEE Computer Society, 2007.

[11]. Alessandro Marchetto · Filippo Ricca, “From objects to services: toward a stepwise migration approach for Java applications”, Published online: 28 October 2009, © Springer-Verlag 2009

[12]. Randy Miller, “Quality of Service Requirements” Randy Miller's Blog, Software Development on the Difficult Projects, 28 Nov 2005