CHAPTER-1

Introduction

1.1 General:

Legacy Systems are those systems/ applications that have been developed long time ago and are still in use. These systems are often hard to improve and expand because there is a general lack of understanding of the system; the designers of the system have left the organization, so there is no one left to explain how it works \[1\]. Such a lack of understanding is further aggravated by little or no documentation. Software engineering, which stresses the need and importance of documentation, has evolved much latter. Also these systems are developed using old technology and are working on old platforms. e.g. 3GL COBOL systems working on mainframes. Despite these problems, organizations can have compelling reasons for keeping such a system, such as, rich functionality that implements main business logic (Payroll, Inventory, Financial Accounting etc.), the costs of redesigning the system are prohibitive because it is large, monolithic, and/or complex, system do still meet basic needs of the organization etc. etc.

Another dimension is the need that has emerged for these applications to work, both among themselves, as also with newer applications in an integrated manner. This is the result of realization that most systems do not exist in isolation, instead they must interact with other systems in some fashion. Very often, a strict transactional solution is required, whereas, sometimes a batch integration (e.g. for reports etc.) may be acceptable.

Reengineering is the complete overhaul of a process in order to achieve goals not being met using current system. Attempts have been made, notably in the last two decades to reengineer these legacy systems so as to move them to newer technology (e.g. Databases, client server architecture etc.). These attempts have mostly been from the point of view of getting the advantage of modern platforms, and better
documentation. Yet this task has proven to be quite illusive, with more than one trillion lines of COBOL code still being used. Multiple alternatives are being used for arriving at a workable solution.

We have attempted to view this problem as an EAI problem that is general enough to encompass several disparate applications including some legacy applications. EAI is a business computing term for plans, methods, and tools aimed at modernizing, consolidating, and coordinating the computer applications in an enterprise. Typically, an enterprise has existing legacy applications and databases and wants to continue to use them while adding or migrating to a new set of applications that exploit the internet, e-commerce, extranet, and other new technologies. EAI may involve developing a new total view of an enterprise’s business and its applications, seeing how existing applications fit into the new view, and then devising ways to efficiently reuse what already exists while adding new applications and data. Present EAI solutions are usually tightly coupled, proprietary, and are not easily amenable to changes. Service Oriented Architectures (SOA) have emerged recently and are considered to be an ideal solution of EAI issue with simple and extensible option.

In this thesis, we have looked at reengineering of legacy systems in the perspective of SOA based EAI. We have shown that using this approach, only limited changes need to be made to any legacy application that is still functioning satisfactorily. Persistent messaging, which first led to J2EE architecture and which has later evolved into modern SOA has been used as the backbone for the suggested architectures.

1.2 Layout of Thesis:

The thesis is divided into eight chapters.

Chapter 2 discusses Legacy Systems and Reengineering Issues. The legacy systems in general and their various definitions are discussed first, followed by gravity of the problem. Software renovation is discussed next. Reengineering in general and the
taxonomy follows then. Finally issues and challenges facing legacy systems is
discussed.

Chapter 3 discusses the Enterprise Application Integration (EAI) in general, followed
by its need, characteristics, and requirements. The chapter next presents logical EAI
architectures, followed by levels of integration. In the chapter next discussed are
various available EAI approaches. In the chapter then discussed is an XML based
framework for EAI. The chapter finally reasons out why EAI is the best approach for
accessing Legacy data.

Chapter 4 discusses J2EE in general. It starts with discussing J2EE architecture and
its connector architecture in detail. It then presents a discussion on Enterprise Java
Beans (EJB). Types of EJBs and architecture of EJBs discussed here. The chapter
next presents eXtensible Markup Language (XML). It discusses issues like XML and
B2B, XML within J2EE, XML for Integrating Legacy Data, and promise of XML.

Chapter 5 presents our first approach towards the problem solution. It presents the
approach in general followed by internationally accepted paper on the approach.
Finally, the chapter presents the experimental setup in detail that was used for the
approach. Detailed software design specifications that were used follow next.

Chapter 6 discusses the modern EAI frameworks, which are woven around Services
Based Architectures (SOA). The chapter discusses in detail Web Services, Simple
Object Access Protocol (SOAP), Web Services Description Language (WSDL), and
Universal Description and Discovery Integration (UDDI).

Chapter 7 presents our second approach towards the problem solution, which is
woven, around Services Based Architectures (SOA). It also presents another paper
written around the concept of Web Services.

Conclusions are listed in chapter 8.