CHAPTER 1

INTRODUCTION

This thesis work entitled “Development of Components with Effective interoperability among heterogeneous Platforms” is focusing on finding frequent itemsets and integrating heterogeneous components.

1.1 OVERVIEW OF COMPONENTS

The complexity and resource demands of present-day software systems create the need for more flexible solutions than those offered by conventional programming environments based on a succession of subroutine calls. One solution is component programming, based on encapsulating units of functionality and providing a meta-language specification of their interfaces [79].

Software engineering techniques are embracing “software component” model as an alternative to the development of large applications and systems. Software components, whether object-oriented or not, solve a number of application development problems [19]. Software components could be used as building blocks when developing new application software, thereby reducing the expense and risk of developing everything from the scratch.

A software component is an unit of composition with contractually specified interfaces and explicit context dependencies [89]. Software components are prefabricated, pre-tested, self-contained, and reusable
software modules. They bundle data and procedures that perform specific functions. Advantages of component software are reusability, economy, (user) modifiability, extensibility, quality and vendor neutrality [60] [50]. Reuse is also being proposed as a way of increasing quality of the software, as is described by Lim in [55].

Software component can be deployed independently and is subjected to composition by third parties. Benefits of Component-Based Development are [54] [9]:

- Quick deployment of critical software applications
- Simplification of large-scale software development
- Encapsulation of business services into reusable application logic
- Shortening of software development cycles
- Reduce the amount of new code to write
- Allow software applications to share functionality
- Make software applications more adaptable and easier to change
- Decrease software complexity
- Increase software reliability and overall quality and Increase software productivity by reducing costs.

Component Based Development

The need to develop software based on existing code rather than development from scratch, led to the emergence of component-based software. Component-Based Development (CBD) is concerned with the development of systems from reusable parts, that is, components [36].
Component based design is perceived as a key technology for developing advanced real-time systems in both cost and time effective manner. Component-based software development (CBSD) is highly promising in improving software development productivity and quality by developing application software using existing software building blocks or services instead of developing application software from scratch [63].

Building software systems with reusable components brings many advantages [59]. If the reuse concept is utilized on several levels of a system development, the development becomes more efficient, the reliability of the product is enhanced, and the maintenance requirement is significantly reduced. The levels of reuse are spread out from the reuse of source code and common libraries, through the reuse of large business components, up to the reuse of the standard products in the configuration of large systems. Designing, developing and maintaining components for reuse is a very complex process. This places high requirements not only for the component functionality and flexibility, but also for the development organization [89]. Component-based development aims at constructing software artifacts by assembling software components [34].

The Component Object Model (COM), CORBA/IIOP, Enterprise Java Beans (EJB) object models and more recently, .NET are tools that allow the creation of reusable software components. Each of these models works well provided that cross-platform integration is not required. Anyone who has tried to integrate programs across these technologies knows that doing so is time consuming and expensive [92].

This work focuses on two aspects. First one is on creation of an EJB component for finding frequent itemsets. Second one is on the integration
Chapter 3 of this work describes a component for mining frequent itemsets in Data mining. Data mining, the extraction of hidden predictive information from large databases, is a powerful new technology with great potential to help companies focus on the most important information in their data warehouses. Data mining tools predict future trends and behaviors, allowing business houses to make proactive, knowledge-driven decisions [53].

1.2 DATA MINING: AN OVERVIEW

Data Mining is a concept that is taking off in the commercial sector as a means of finding useful information out of gigabytes of data. In Business and Marketing domains large volumes of data are stored in centralized and distributed databases. In any Business application, data need to be analyzed for sales forecast, business planning and marketing trend etc. Data mining is very essential in any business application to take crucial decisions. Data mining research during the last eight years has led to the development of a variety of algorithms for finding frequent itemsets in very large transactional databases. Finding frequent itemsets from data streams is one of the important tasks of data mining. Acquiring new customers, delighting and retaining existing customers, and predicting buyer behavior will improve the availability of products and services and hence the profits. Thus the end goal of any Data Mining exercise in e-commerce is to improve processes that contribute to delivering value to the end customer [84].
Data mining uses algorithms to examine through huge volumes of information for the purpose of detecting frequent patterns hidden in the data. Understanding these patterns quickly leads to improved business intelligence.

Data mining is supported by technologies such as [53]:

- Massive data collection and storage
- Powerful multiprocessor computers
- Data mining algorithms.

Frequent itemset mining is a core data mining operation and has been extensively studied over the last decade. Algorithms for frequent itemset mining form the basis for algorithms for a number of other mining problems, including association mining, correlations mining, and mining sequential and emerging patterns. Algorithms for frequent itemset mining have typically been developed for datasets stored in persistent storage and involve two or more passes over the dataset.

1.2.1 Existing Techniques

Finding frequent itemsets is one of the most important data mining tasks. Data mining algorithms embody techniques that have existed for at least ten years, but have only recently been implemented as mature, reliable, understandable tools that consistently outperform older statistical methods. In business application, finding frequent itemsets is very essential. Especially in a business application, which is distributed over different locations, the process of finding frequent itemsets is very important.
The most well known and influential algorithms are Apriori [1] and FP-growth [35]. Apriori uses an a-priori knowledge of frequent \( k \)-itemsets to generate candidate itemsets of length \((k+1)\) and employs an innovative technique for pruning non-promising candidates. In Apriori Algorithm, multiple scanning of the transactional database is needed to find frequent itemsets, which is costly. The Apriori algorithm performs a breadth-first search in the search space by generating candidate \( k+1 \)-itemsets from frequent \( k \)-itemsets. The frequency of an itemset is computed by counting its occurrence in each transaction. To break the two bottlenecks of Apriori series algorithms, some works of association rule mining using tree structure have been designed. FP-Tree [Han et al 2000], frequent pattern mining, is another milestone in the development of association rule mining, which breaks the two bottlenecks of the Apriori [74]. FP-growth is a well-known algorithm that uses the FP-tree data structure to achieve a condensed representation of the database transactions and employs a divide-and-conquer approach to decompose the mining problem into a set of smaller problems. In essence, it mines all the frequent itemsets by recursively finding all frequent 1-itemsets in the conditional pattern base that is efficiently constructed with the help of a node link structure.

1.2.2 Motivation

A fundamental component in data mining tasks is finding frequent patterns in a given dataset. Frequent patterns are ones that occur at least a user given number of times (minimum support) in the dataset. They allow us to perform essential tasks such as discovering association relationships among items, correlation and sequential pattern mining. The Apriori algorithm accomplishes this by employing a bottom-up search. But almost all previous Apriori-like algorithms use the candidate set generate-and-test approach [103].
In order to overcome the disadvantages of Apriori algorithm and efficient mine association rules without generating candidate itemsets, a frequent pattern-tree (FP-Growth) structure is proposed by Han et al [103]. It has performance improvements over Apriori since it uses a compressed data representation and does not need to generate candidate sets. The construction of FP-Tree requires two data scans. In the first scan, the support of each item is found. In the second scan, items within transactions are sorted in descending order according to the support of items [103].

The FP-Growth was used to compress a database into a tree structure which shows a better performance than Apriori. However, FP-Growth consumes more memory and performs badly with long pattern data sets. In order to further improve FP-Growth algorithm, many authors developed some improved algorithms and obtained some promising results [37, 56, 93, 96].

Chapter 3 describes the proposed algorithm for finding frequent itemsets, which is a vital process for decision-making process. Apriori algorithm needs n+1 number of scanning, which takes more time to find the frequent itemsets. In the proposed algorithm, number of scanning processes had been reduced compared to Apriori algorithm and there is no need for tree construction as in FP-Growth algorithm. The algorithm is simple, easy to implement, easy to understand and involves less computational work. The algorithm is compared with FP-growth algorithm. From the results it has been found that this algorithm is simple and easy to understand. In the proposed algorithm, 2-itemsets are obtained directly from the Support Count Table (SCT). For more than 2-itemsets, checking process of fully connectedness is done along with Logical AND operation. The EJB component has been developed based on the proposed algorithm, which can be used from
anywhere. Enterprise JavaBeans (EJB) is Java’s new component architecture for the development and deployment of reusable Java server components [57].

1.3 INTEGRATION OF COMPONENTS: AN OVERVIEW

Component-based software development encompasses two processes:

1. Developing reusable components and
2. Assembling software systems from software components.

Assembly is the integration of components through some well defined infrastructure, which provides the binding that forms a system from disparate components. Developing large software systems by integrating components has a great potential to reduce costs and time to market [23]. Components are the basic building blocks of enterprise and distributed applications. Component Based Development (CBD) is the technique of using a Component Framework to develop Components and it focuses on developing large software systems by integrating previously existing software components [80]. By using components it is possible to produce more functionality with the same investment of time and money [39].

Component-based software development addresses issues of language independence - seamlessly combining components written in different programming languages and component frameworks define standards for communication among components [79]. One of the challenges of component based development system is how to integrate various components in software systems [102]. Software component integration is so intertwined with component enhancement that it is impossible to cover the former without at least touching on the latter [46]. Integration of software components is an emerging concept in the software development process.
Interoperability is the key concept for integrating the components developed in different languages or on different platforms. Interoperability refers to the ability of software systems to communicate with each other independent of vendors and platforms. Interoperability is a complex concept with many technical and business ramifications [38].

The Institution of Electrical and Electronic Engineers (IEEE) provides the generally accepted definition of interoperability as: "The ability of two or more systems or components to exchange information and to use the information that has been exchanged."

The interoperation between different technology components is in practice much more complex and difficult than in theory [52]. Interoperability refers to the ability of software systems to communicate with each other independent of vendors and platforms. Platform such as J2EE simply does not recognize interoperability between implementations as a key requirement. Companies wishing to integrate their supply chain, customer relationships, resource planning and other essential business processes under a single development platform J2EE or .NET. Even though the idea is nice, reality reveals that no one platform or operating system will be sufficient to meet the needs of the customer. Interoperability between multiple platforms is vital. Commercial Off-The-Shelf Components (COTS) for example, are usually written to some component model defined by e.g., Enterprise JavaBeans, COM, CORBA, or, more recently, .NET [16].

1.3.1 Existing Methods

Component based software engineering was proposed already in 1968[33]. Nevertheless, the focus on systematic adaptation of components in order to bridge interoperability problems are still a field of research [85].
Interest in component-based software development, in software reuse, and in distributed or network-based software systems is growing daily. Software interoperability is fundamental to each of these topics. As a result, a variety of (often partial) approaches to interoperability, including such things as remote procedure calls, client/server architectures, object request brokering (e.g., CORBA), document embedding (e.g., OLE) and applets (e.g., Java) have recently been receiving a great deal of attention. Many commercial and freeware systems have appeared, each offering a potential solution to interoperability problems. Yet what these approaches and systems offer, how they compare, and exactly what problems they are solving are generally unclear [40].

The need to integrate complex business software applications is not new. Until recently, the tools and technologies available for this integration were either crude or too tightly coupled to proprietary technologies. Most firms have used crude methods for integrating software applications, with some success [92]. For application architects, the challenge is to find strategies to integrate and extend existing deployments without ‘rip and replace’ options or costly application rewrites.

Interoperability is not a problem; it’s a software quality. The problem in achieving this quality is heterogeneity of components, which are written in different programming languages and running on different hardware platforms [24]. A major reason for difficulty with interoperability in the past was a lack of agreed upon and adopted standards for such interoperation. There exist two well-defined and mature standards that can help to alleviate the interoperability pains the Internet Inter-Orb Protocol (IIOP) and the Simple Object Access Protocol (SOAP)) [91]. SOAP is text-based and thus by definition verbose, communications using binary protocols
such as IIOP will in almost all cases outperform those that use SOAP as the underlying protocol. IIOP is part of the CORBA standard.

The IIOP protocol used to access the object is well established and quite efficient compared to the fancier and less powerful Web Services. The development of an application using IIOP includes some pitfalls, often caused by the complex and non-trivial configuration and use of the various application servers hosting the EJB services. In particular, the service naming and location may heavily differ among the various servers.

IIOP .NET is a .NET remoting channel based on the IIOP protocol, and the same is used by Java's RMI/IIOP also. IIOP.NET acts as an ORB (a CORBA object request broker); it makes objects defined in .NET application accessible to other remote ORBs, and vice-versa. Java RMI/IIOP implements a subset of the CORBA type system (due to some limitations in Java's type system) and roughly provides the same features as IIOP.NET for the J2EE platform.

1.3.2 Motivation

Almost all interoperability technologies are based on either IIOP or SOAP. In addition, each solution can be placed into one of two categories: open and proprietary. Open technologies are created by consortiums such as Organization for the Advancement of Structured Information Standards (OASIS) or by standards bodies such as the W3C. Proprietary technologies are created by software companies such as Borland with the intent of selling them as a commercial product.

Chapter 5 gives the existing interoperability technologies and their drawbacks. In this chapter, description of the interoperability technologies
such as Web Services, J-Integra and Janeva are made along with their limitations. WebServices can be used only for basic interoperability. Janeva is based on the Internet Inter-ORB (Object Request Broker) Protocol, or IIOP. J-Integra Espresso is a true CORBA solution for the Microsoft .NET framework. But J-Integra and Janeva are proprietary products and therefore has all the limitations associated with a product that's tied to and dependent on a vendor. And in J-Integra products, the class path setting is very tedious, which is a vital point in providing interoperability.

In order to overcome the above said pitfalls, an automated design tool Platform Interoperable Solutions (PIS) has been developed using IIOP.NET. PIS eases the task of configuration process. PIS provides the interoperability between the platforms J2EE and .NET. PIS tool is machine independent and component independent. Using PIS, the integration of different components belonging to EJB and .NET can be achieved easily and effectively. In PIS, there is no need for worrying about class path settings, location of components and machines.

Through PIS, the interoperability between EJB and .NET can be achieved both on server side and client side. Once the interoperability is achieved between .NET and EJB then the integration of them is simple. PIS has been developed using existing software only. Thus for using PIS, no additional software is needed.

The development cost and maintenance cost of PIS is very less. Manual work involved in PIS for configuration process is very less. It offers the flexibility of configuring components either on client side or server side. Components of different platforms (.NET AND J2EE) and different vendors can be integrated for different applications using the concept of PIS Tool, which provides interoperability. PIS is viewed as an efficient design tool for
component based software development. The performance of the tool PIS has been checked for a Banking Application in chapter 8.

1.4 CONCLUSION

This chapter briefly outlines the development of components and advantages of Component Based System Development. Various issues for finding frequent itemsets are discussed. A brief summary of available technologies for integrating heterogeneous components and their limitations are presented. Finally, the motivation for developing a new algorithm for finding frequent itemsets and developing a new tool namely Platform Interoperable Solution tool which offers interoperability among various platforms are discussed.