UNIVERSITY of BOTSWANA

Using Ado.net Entity Framework Component model (EF component) in aiding legacy system integration in Service Oriented Architecture environment.

STUDENT NAMES: ASHERL BWATIRAMBA (201208085)
PROGRAMME : MSC COMPUTER INFORMATION SYSTEMS
SUPERVISOR : DR. G MALEMA

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science in computer information systems at the University of Botswana.

August, 2016

ABSTRACT

The push towards interoperability among different legacy systems in the industry has generated the need for having an integrated platform for legacy systems. However, semantic discrepancy between legacy systems and other applications has made integration a challenge. Current integration approaches addressing legacy system integration have shown gaps that are hindering broader integration. In this study, we aim to mitigate the gap by incorporating and evaluating Ado.net Entity Framework Component model (EF), a component of Microsoft.Net to achieve broader integration. Normalized System theory concepts are applied to provide more systematical way of performing architectural evaluation. This dissertation employed design science research methodology. It explored existing knowledge on EF Component, investigate and analyze possibilities for improving integration. This research involves research activities such as (1) design the artifact (prototype), (2) examine and evaluate the artifact using Normalized System theory (NS Theory) and (3) come up with guide guidelines on how to use EF component model within SOA to aid in integration. The results from the findings show that EF component can be used to aid in integration but has some combinatorial effects especially in its performance and scalability. However, after iterating and analyzing the prototype, this study proposes our five key findings to be followed when using EF in legacy integration.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude towards Dr.G.A Malema for helping me complete my Masters with valuable research experience. His guidance, encouragement, supports and patience has helped me through every step of my study period.

Whole-heartedly, I would also like to thank Mr. Cross Gombiro for his uninterrupted cooperation and help in implementing a significant part of my prototype. I wish him good luck in his career.

Last but not least; I take this opportunity to sincerely thank the entire Department of Computer Science, Faculty of Graduate Studies, all my peers and friends for their cooperation and help during this important phase of my life; I would like to extend a special thanks to my family, for their immense contribution in helping me meet my dissertation deadlines successfully.

Contents

TABLE OF FIGURES

CHAPTER ONE: INTRODUCTION

1.0 INTRODUCTION

Large enterprises are typically supported by many applications. Many of these applications are written in different programming languages. They are standalone systems designed with a sole purpose of fulfilling a need in the organization and there are referred as legacy systems. Enterprises depend on legacy systems for their operations. However, constant technological changes have weakened the value of legacy systems as they now appear absolute; high cost to maintain, poorly documented, complex structure [1]. Scrapping legacy systems and replacing them with modern software applications poses a great business risk such as failure to create an identical system like that was in use because of no proper documentation [1].

Integrating legacy systems can provide benefits such as standardization, effective support of processes, using resources optimally, reducing sub-optimization and minimizing problems with communication between different areas [2]. Integration in this context is referred to as the process of linking together different computing systems and software applications physically or functionally, to act as a coordinated whole.

It is very difficult to integrate existing systems components that are not originally designed to work together. The business functionality remains locked within these applications and is often not accessible for innovation. Most companies do not have the knowledge of accessing the internal workings of the legacy systems due to lack of support from experienced colleagues [3]. Many of the legacy systems were developed using proprietary technologies and most of them lack integration standards such as XML [4].

Consequently, the integration of applications is refered as the Enterprise Application Integration (EAI). EAI solves the problem of integrating modular systems by treating integration as a task for a system, like any other task, rather than a snarled mess of brittle connections. EAI systems bundle together adapters for connectivity, data transformation engines to convert data to an appropriate format for use by the consumer, modular integration engines to handle many different complex routing scenarios simultaneously, and other components to present a unified integration solution [3].

EAI is a broad field, where different types of integration approaches and technologies have been developed [3]. Service oriented Architecture (SOA) approach is a promising approach in EAI [4]. This type of integration revolves around logic level integration. SOA is an architectural framework for distributed software system development in which software components are packaged as Services. Under this paradigm, middleware such as Common Object Request Architecture (CORBA), Java Enterprise Edition (JEE), DCOM, .NET Remoting, Web services and Windows Communication Foundation (WCF) are used. These middleware acts as glue between two applications, connecting two sides of applications and passing data between them. However, these middleware do have some challenges and such challenges include the ones listed below;

- Proliferation of middleware technologies; a number of the middleware platforms are designed for closed systems (systems which are isolated that have no interaction with their external environments, their outputs are knowable only thorough their outputs which are not dependent on the system being a closed or open system). For instance a middleware for java systems only or a middleware for Microsoft systems only. This means these middleware platforms are not able to work with different platforms and languages. As such interfacing with legacy code written in different languages is difficult.

- Complex to configure- In the event that one wants to configure and deploy the legacy systems with middleware, it becomes a tiresome task to assemble the components and sometimes it may require redeveloping the legacy systems or making use of scripts which is time consuming, risky and costly.

SOA as promising as it is, posses some challenges when it comes to the integration of legacy systems. SOA approach fails to provide seamless access to legacy code and middleware which makes it a challenge to integrate legacy systems [4]. This dissertation has identified this failure as a gap that is hindering integrating of legacy system within SOA and this gap has formed the basis of our dissertation.

As such this study uses data level integration approach of EAI. In data level integration, existing data sources of legacy application are made accessible to new applications. The data can be replicated in a new database and made accessible for new applications, or a database gateway could be used [6]. Ado.net Entity Framework Component model (EF) is one of the prominent ORM (Object Relational Mapping) component used in the field of data abstraction. It is a .Net Component which moves the data in its most natural format to and from a permanent data sources. It manages the databases and the mapping between the database and the object in the programming side.

An Inclusion of EF component model within SOA requires an exhaustive study and investigation into its underlying model. Since every component model is fairly comprehensive, EF component model contains an associated methodology and architecture that needs to be used while interoperating with other component models. Thus, the incorporation of any component model into the SOA integration requires a thorough understanding and investigation of the component model.

Despite the identification of success factors and design principles for EAI, and regardless of the type of enterprise application integration [5], integration projects are often considered to be complex. During integration, various nonfunctional requirements need to be taken into account, such as performance, maintainability, evolvability and scalability. As a result, the life cycle of information systems is altered by the integration project, and the complexity of adapting existing applications needs to be considered in addition to the inherent complexity of the integration project itself. Normalized Systems Theory (NS Theory) [6] proposed a structured approach to handle this complexity. It provides a suitable basis for evaluating the quality of information systems' business architectures. A good example of an integrated platform should have high quality and be able to evolve. Recently, NS theory has been proposed as a way to deal with ever-changing requirements for software by building evolvable information systems, based on the systems theoretic concept of stability [7].

This study used an EAI approach for integration legacy systems, incorporating EF in its architecture and introduces (NS) Theory to perform an evaluation of the enterprise architecture. In this dissertation we try to contribute to the solution of the problems of integration by designing an evolving prototype system that supports multiple legacy systems thereby promoting broader integration in software developments.

1.2 BACKGROUND OF THE STUDY

Microsoft.Net Framework is a technology from the Microsoft Corporation which can promote broader integration of systems [9]. Microsoft.Net Framework also deals with the integrations issues/challenges such as data mismatch, pattern mismatch, protocol mismatch and data format which are normally associated with systems that are of different platforms. Microsoft.Net framework support building and running coherent applications and services capable to address the current needs of individual and large organization. It is composed of many components and one of the components called Common Language Runtime (CLR) provides the execution environment to manage and execute .Net application [10]. It enforces code robustness by using a component called Common Type System (CTS) which provides a common set of data types known as base types. The base types allow objects written in different .Net Language to interact with each other.

Microsoft.Net framework includes components such EF and WCF in its architecture [10]. These components are from the same vendor which is Microsoft which means that the data definitions, protocol used, data patterns and data format are the same thereby encouraging integration.

EF enables developers to work with data in the form of domain- specific objects and properties for instance student name, student address, Date of birth etc, without having to be concerned with the underlying databases, table and the columns where the data is stored. Traditionally, a developer had to have the knowledge of the data engine used to store and retrieve data, the relationships of the database table, the data types and formats used and then map them using programming code to the objects in the programming side. It is an object relation mapper (ORM) [11], which means it moves the data in its most natural format to and from a permanent data sources. It manages the databases and the mapping between the database and the object in the programming side.

On the other hand, WCF is another component of Microsoft.net framework used to develop distributed service oriented applications. It is the middleware used to enable communication across platform in distributed environment. It is a unified programming model for developing service oriented architecture (SOA). It combines the features of all .Net middleware technologies such as COM+ services, .Net Remoting and Web services.

SOA has emerged as an effective paradigm in the enterprises computing space for addressing software integration. The service oriented approach has led to the emergence of middleware for integrating purposes. These middleware are softwares that connect to otherwise separate applications. It acts as glue between two applications, connecting two sides of applications and passing data between them. Middleware encapsulates specific services or a set of services to provide reusable building blocks that can compose to develop distributed systems.

Examples of mostly used middleware include Common Object Request Architecture (CORBA), Java Enterprise Edition (JEE), DCOM, .NET Remoting, Web services and Windows Communication Foundation (WCF). However these middleware do have some challenges which have been mentioned above.

Therefore, since middleware from SOA are still not completely solving the problems in integration, the question that follows is *how then do we get access to the legacy code of legacy system so as to achieve integration using WCF as our middleware?*

This question has formed the basis of our dissertation research. This research uses EF component, an ORM tool to be used in aiding legacy system within SOA. EF with the capability of systematically abstracting and persistence data from the relational databases of the legacy systems is used to access data from the legacy applications. However not much has been done on how the EF component model should be used for integration in the context of legacy system integration.

1.3 STATEMENT OF THE PROBLEM

The main problems addressed by this dissertation are mostly related and dependent on tackling issues that arise while bridging different component of the middleware and legacy components within SOA. There are disparities that exist between legacy components and middleware components, hence the need for EF component model. The core common problems/issues include;

- No interoperability among the legacy systems and middleware- middleware platforms are not able to work with different platforms and languages. As such interfacing with legacy code written in different languages is difficult.
- Conversion problems- data from the legacy systems cannot be convert to the language that can be understood by the middleware and vice-versa, because of different data formats, protocols for communications.
- Complex to configure- In the event that one wants to configure and deploy the legacy systems with middleware, it becomes a tiresome task to assemble the components and sometimes it may require redeveloping the legacy systems or making use of scripts which is time consuming, risky and costly.
- Conflicting data stored in different locations results in higher operations costs, reduced customer satisfaction, and other negative impacts to an organization's bottom line.

1.4 OBJECTIVES

The main objective of this research is to evaluate EF component model based on the criteria such as performances, mapping, evolving, abstraction and scalability. The objective of this dissertation of encompassing EF into SOA is subdivided into three specific objectives;

- Investigate and understand the basic idea of service oriented based methodology that enables interoperability among software system.
- Develop EF Component Model and WCF service. EF Component Model and WCF represent basic components to prove the concept of the proposed system.

- Evaluate the prototype based on NS Theory and come up with recommended guidelines for a broader and evolving integration method.

1.5 RESEARCH QUESTIONS

The main research question is *how should software development projects that focus on service oriented approach such as WCF incorporate the EF Component model for legacy system integration?*

The research question can be broken down into sub question as follows;

- *What are the motives and goals for incorporating EF Component model in legacy system integration?*
- *What are the most important factors/criteria to consider when adopting the EF Component model in WCF service and legacy system?*
- *What are the most important guidelines for adopting EF Component model in WCF services and legacy system?*

1.6 JUSTIFICATION

This dissertation evaluates EF Component model to promote broader integration. From the literature review, previous employed methodologies such as point-to point integration, re-engineering and web service integration did not adequately address the above mentioned challenges in integration. EF Component model is a data access technology for Microsoft world and most of the new frameworks, technologies and tools work with Microsoft technologies. However to be able to address the problems mentioned, EF component model was investigated and evaluated against criteria such mapping; abstraction; performance and scalability. The result of the dissertation can be used as guidelines to the software maintainers, who are forced to work independently with each system. Hence, software integration project managers, systems maintainers, system administrators and system developers are the target audience of the dissertation.

1.7 CONTRIBUTION

Our contribution in this dissertation is to promote broader integration in legacy systems using EF component model. The model was to be applied with modifications to many object-oriented applications that take advantages of the capability and versatility of SOA. We evaluated the use of EF component in SOA environment using NS theory to designing an evolving integration system that support multiple legacy system thereby promoting broader integration in software developments.

1.8 SCOPE

The scope of this dissertation outlines the design and implementation of a service oriented system that enables interoperability among legacy components using EF Component model. Though they are other integration mechanism such as Extract Transform Load (ETL), Enterprises Service Bus (ESB) and Message Oriented Middleware, this research target the service oriented integration method in creating two types of research component namely EF Component model and WCF to be used to promote integration in object oriented legacy software systems.

1.9 ORGANIZATION OF THE DISSERTATION

This dissertation comprises of 4 chapters. The chapters include introduction, literature review, research methodology, and conclusion and implication. The research starts by discussing the problem domains, objectives, expected outcome, scope, and significances of the research. A brief description regarding the organization of the research is provided. This material is presented in

Chapter 1 Introduction. **Chapter 2** is the literature review and focus on the generic information on EF and WCF. Then, the chapter will also focus on the integration approach EAI, and chapter 3 is the related works. **Chapter 4** explains the methods involved in the research and finally **Chapter 5** is Conclusion and recommendations.

CHAPTER 2: LITERATURE REVIEW

2.0 INTRODUCTION

This chapter discusses the literature of our research. The discussion aims to identify the broad basis for our research. This chapter also discusses the understanding and the critical points of the current knowledge regarding the selected approach- Enterprise Application Integration (EAI), Normalized System Theory (NS Theory) and Goal-Question-Metric Approach. EAI acts as the preference and basic architecture of the proposed service integration systems. In concluding this chapter, the last issue to discuss is the related works.

2.1 LEGACY SYSTEMS

Legacy systems herein can be referred as the standalone systems designed with sole purpose of fulfilling a particular need in an organization. Legacy systems are very important in the day to day running of the business; therefore their existence serves a powerful purpose in the business as they are flexible, scalable and more often they are customized to the needs of a particular department [1]. As your large enterprise grows, your IT challenges grow with it. Virtually all enterprises have legacy applications and databases [2], and they want to continue to use them while adding or migrating to a new set of applications that utilize the Internet, e-commerce, the extranet, and other new technologies.

In addition, as organizations expand and merge their IT platform deployments often become fragmented. For example, different departments within an organization may use different applications and persistence mechanisms to access the same data. As a result, some data, such as customer information, may exist in multiple locations. These may cause problems such as lack of efficiency (having to enter the same data multiple times) and inconsistency in data that is stored in different locations. Conflicting data stored in different locations results in higher operations costs, reduced customer satisfaction, and other negative impacts to an organization's bottom line.

Having a unified view of all mission-critical data is beyond the capabilities of such a fragmented system.

Today, legacy systems must be designed to be capable of integrating with other applications within the enterprise. However, scrapping legacy systems and replacing them with more modern software involves significant business risk. Replacing a legacy system is a risky business strategy for a number of reasons [1]: a) there is rarely a complete specification of the legacy system. Therefore, there is no straightforward way of specifying a new system, which is functionally identical to the system that is in use, b) Business processes and the ways in which legacy systems operate are often inextricably inter-twined. If the system is replaced, these processes will also have to change, with potentially unpredictable costs and consequences, c) important business rules may be embedded in the software and may not be documented elsewhere, d) new software development is itself risky because there may be unexpected problems with a new system [1].

Connecting to your legacy applications saves the time and expense of having to migrate the legacy applications, and it provides a mechanism for tying together fragmented IT platforms. For years, IT departments have created point-to-point mechanisms, and middleware developers have written millions of lines of code for achieving interoperability of data sources and applications.
Benefits of interfacing with legacy applications include [2];

- **Reduced IT costs due.** Historically, most organizations have addressed interface challenges by writing large amounts of code. Employing well-designed solutions can reduce the initial financial and time outlays as well as the ongoing maintenance costs of this effort.

- **Reduced operational costs through more efficient value-chain processes.** Automating key value-chain processes that reduce business process cycle times can reduce costs in many ways. For example, a more efficient supply chain can reduce the cost of carrying inventory.
- **Higher customer satisfaction and loyalty through new services and programs.** Interface projects are essential for offering new information and business services more quickly than your competitors. For example, key online customer "self-service" operations function more easily when connecting the appropriate systems.
- **Better and faster business decisions.** Aggregating business information and making it available in near-real time can fundamentally improve your ability to make better business decisions more quickly than your competitors can.

One of the promising approaches in legacy system integration is based on the service oriented approach [3]. Service Oriented Architecture (SOA) is a recent architectural framework for distributed software system development in which software components are packaged as Services.

2.2 SERVICE ORIENTED ARCHITECTURE

In the SOA paradigm, functionality is exposed as services thereby enabling service requesters and service providers to interact through messages. Service orientation provides guidelines and principles that govern the creation, implementation and management of services.

Software development is facing a major transformation from an application based to service oriented architecture both within and across enterprise boundaries. The question is why SOA? Organization have been building software from scratch or buying from the vendors to suit the needs of the organization. This has lead to islands of systems which cannot correlate and involve duplication, errors and unstable data. These are the systems we are calling legacy systems in this research. Some of the identified problems from literature include; [1]

- A lack of clean interface making integration with other systems difficult.
- No interoperability among the systems.
- Overlapping functions meaning that a lot of data replication is needed to keep the parts in synchronization.
- Conversion problems- data conversion because of different data formats, protocols for communications.
- Personnel problem- the out of date applications systems are usually run by people who have never worked with newer technologies.

How SOA Works

To understand SOA, consider the architecture commonly found in most organization; data-oriented systems. Departments share information through emails and media. For instance one department may request information about the number of customers who are still outstanding in their payments. To do so, the department which is being requested will respond by putting the information on an excel sheet and send to the other department. That department will take it from an excel sheet and store it in their databases. Such communication between these two systems can be time-consuming and sometimes result inconsistence data, in the sense that any change that happens after the data has been sent to other department will not be captured, thereby causing conflict data and inconsistent data.

The following figure displays the building blocks of SOA.

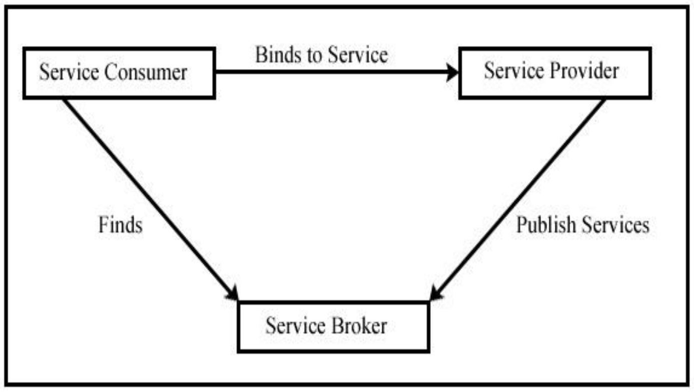


Figure 1: Blocks of SOA

From figure 1, SOA consists of the following building blocks

- Service provider: Publishes a service and registers it in the public registry.
- Service broker: Enables the service consumers to find the service providers that meet the required data.
- Service Consumer: Use a service provider to complement process by binding to a service that is provided by the service provider.

The SOA approach does away with these problems mentioned above in legacy applications. Instead an application is structured as a set of services orchestrated by business process. The legacy systems communicate with each other through services. For instance consider the example above; if legacy system (A) in one department receives information about the customers, that information will be stored in service broker which is the registry in the service and the Legacy system A will be the service provider as shown in figure 1 above. When now the legacy systems B from department request information about the customers from A, it goes through service broker and consumes it, and this legacy system is the Services consumer. However these two systems must be bound to the service. SOA is viewed as the way to bind loosely coupled service or software components from different legacy or open standard applications to empower business agility and facilitate the reusability of the software assets [12].

In SOA approach, services are primarily implemented using WCF. WCF provides a SOA technology that offers the ability to link two non-compatible applications to communicate. SOA approach coupled with WCF simplifies interconnection among existing IT systems. WCF is a Microsoft .Net framework product providing a platform for building services oriented applications.

2.3 HOW WCF WORKS

WCF enables two applications to communicate across platforms in a distributed environment. All communication with a WCF service occurs through the endpoints of the service. An endpoint of a WCF service acts as a gateway for communicating with other applications. It is composed of an address, a binding, and a contract known as the ABC of endpoint, as shown in the following figure.

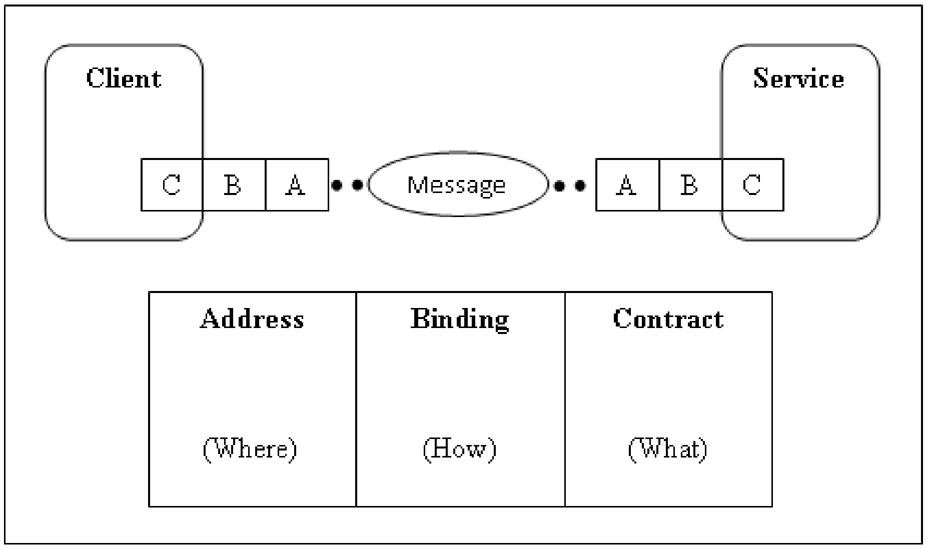


Figure 2: ABC endpoints of WCF

The **address** of a WCF service, from figure 2 specifies the location where the service resides. It is represented in the form of a URL that defines:

- The protocol to be used for sending and receiving messages.
- The name of the system where the service runs.
- The port number at which the service listens to the client requests.
- The path where the service resides and the name of the service.

Binding from figure 2 describes how a WCF service communicates with a client application. It specifies the communication details required to connect to the endpoint of a WCF service. It consists of the message encoder and protocol binding element. WCF provides the following types of bindings to enable a client application to communicate with a WCF service:

Contract from figure 2 exposes the interfaces, classes, methods, and variables of a WCF service to enable client applications to access and use them. A WCF service may contain the following types of contracts:

- **Service contract**
  - Acts as an entry point to access a WCF service.
  - Is implemented as an interface.
- **Operation contract**
  - Exposes the operations that a service can perform.
  - Defines the methods of a WCF service and the parameters and return types of the methods.
  - Is defined by declaring the [OperationContract] attribute in the WCF service.

- **Data contract**
    - Is used to expose user-defined data types in a WCF service.
    - Serializes the user-defined data types in a standard format (XML).
    - Is defined by using the [DataContract] attribute in the WCF service, as shown in the following code snippet:

```
[DataContract]

public class Employees

{    [DataMember]

 public int emp_id;

 [DataMember]

 public int emp_name;

}
```

- **Message contract**
    - Describes the structure of a message exchanged between a WCF service and a client application.
    - Enables you to inspect and control the information contained in a message.
    - Is defined by using the [MessageContract], [MessageBodyMember], and [MessageHeader]attributes, as shown in the following code snippet:

```
[MessageContract]

public class StoreMessage
```

```
{
        [MessageHeader]
          public DateTime CurrentTime;
        [MessageBodyMember]
        public Employees emp_id; }
```

Hosting and Consuming Services in WCF

To host a WCF service, you need to create a .NET application, which can be either a console application or a Windows application.

- To create the application, you need to perform the following steps:

    1. Add the reference of the System.ServiceModel namespace in the application and include the System.ServiceModel namespace.
    2. Embed the code for the WCF service in the application.

Any client application that has the required permissions can consume a WCF service. A client application can be a console application, a Web application, a desktop application, or another service. The process of consuming a WCF service largely depends on the way in which you host the service. To consume a self-hosted WCF service, you need to create either a console or a Windows application (client application). To enable a client application access a self-hosted WCF service, you need to perform the following tasks:

1. Use the System.ServiceModel namespace.
2. Create an object of the binding that you want to use.

3. Create an object of the ChannelFactory class.
4. Create an object of the interface of the WCF service and use the CreateChannel() method to initialize it with the reference to the proxy.
5. Call the methods of the WCF service by using the object of the interface of the WCF service.
6. Compile and execute the application.

2.4 INTEGRATING LEGACY SYSTEMS WITHIN THE SOA

SOA can be used to modernize the legacy systems. SOA focuses on exchange of information among major software components. The primary purpose of adoption of SOA is to improve business communication so that goals of the enterprises can be more readily realized. SOA through its middleware provide a data bridge between incompatible technologies.

However although SOA presents many promising advantage for integrating legacy systems, many issues need to be resolved. First of all, legacy systems usually have proprietary data definitions (systems design exclusively for a specific vendor). This often creates the semantic discrepancy between them and other applications. The gap between the proprietary data definition and SOA semantic foundation needs to be bridged. The service oriented approach has led to the emergence of middleware for integrating purposes. Examples of mostly used Middleware includes Common Object Request Architecture (CORBA), Java Enterprise Edition (JEE), DCOM, .NET Remoting, Web services and Windows Communication Foundation (WCF). However, with these middleware in place, challenges still exist. The middleware platforms are designed for closed systems. This means that these middleware platform are not able to work with different platforms and languages, interface with legacy code written in different languages and interoperate across multiple middleware used by different components developers.

This gap between the legacy systems and SOA middleware has formed the basis of our research. This gap hinders integration and results in the following problems;

- No interoperability among the legacy systems and middleware- the middleware platforms are not able to work with different platforms and languages. As such interfacing with legacy code written in different languages is difficult.
- Conversion problems- data from the legacy systems cannot be convert to the language that can be understood by the middleware and vice-versa, because of different data formats, protocols for communications.
- Complex to configure- In the event that one wants to configure and deploy the legacy systems with middleware, it becomes a tiresome task to assemble the components and sometimes it may require redeveloping the legacy systems or making use of scripts which is time consuming, risky and costly.
- Conflicting data stored in different locations results in higher operations costs, reduced customer satisfaction, and other negative impacts to an organization's bottom line.

2.5 EF COMPONENT MODEL

In today's scenario most of the applications are developed using object- oriented programming (OOP) paradigm. At the same time, the relational databases are used to store and retrieve the data processed in the application [16]. However, there is a mismatch in how data is processed in OOP and stored in relational database tables. ORM have been developed to map the classes of the application to the databases tables so that the data of an object can be directly stored in the database tables. ORM is a layer that is responsible to bridge a gap between object-oriented program and relational database [17]. It provides the object with a persistent service which means that the ability to read from and write data to and delete data from source.

It is from this idea that our research uses EF Component model to enable it to work with relational data as domain specific object, eliminating the need for programming codes to access data from the legacy systems to WCF. This dissertation uses EF Component model to access the legacy system data through its database and pass it to the SOA middleware (WCF) which use an object oriented programming for its operation.

2.5.1 Why EF Component model

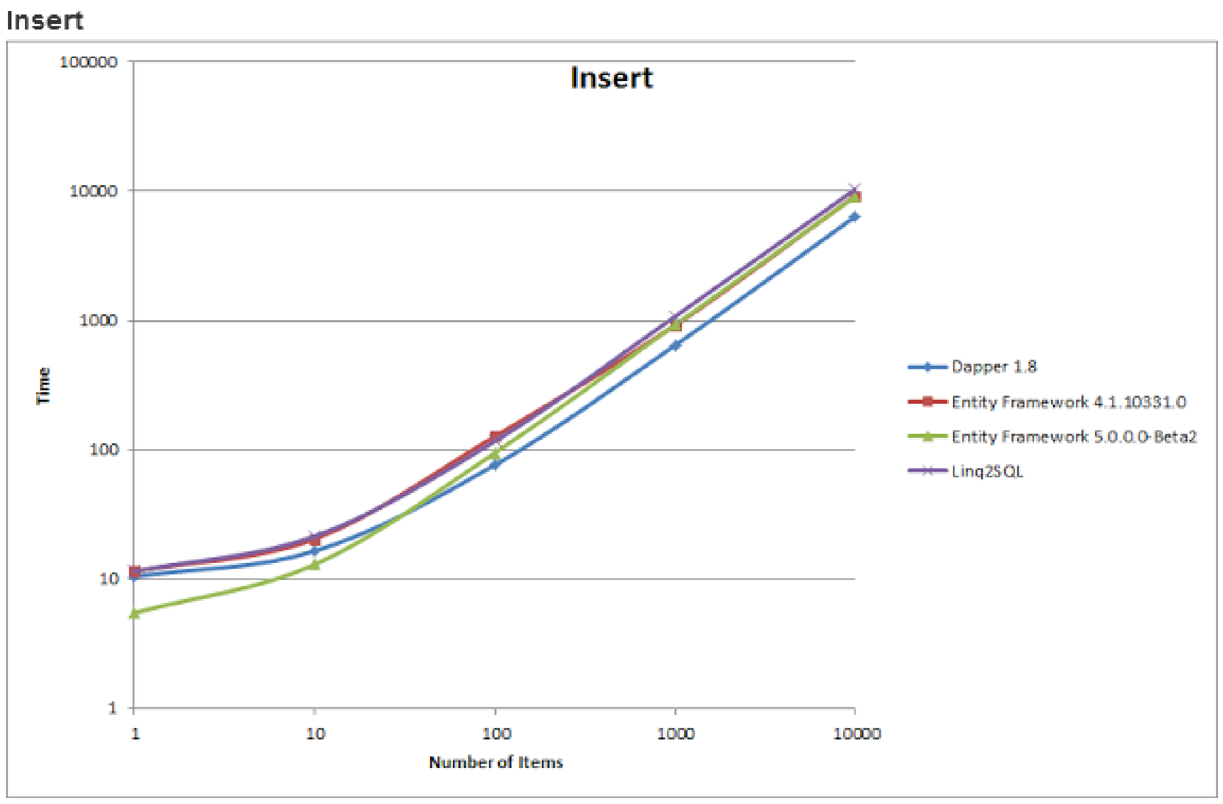
EF is a new technology which continues to exist and improves with time. Researchers in the literature review have researched about this technology and have even compared it with other ORM tools such as NHibernate, LINQ to SQL, Dappers. Below are some experimental research conducted to compare the performance of the ORM. EF component model was called Entity Framework in these experimental studies.

The first experimental study from our literature was to compare four ORM, to test performance. The ORM include the following [18]

- Entity Framework 4.1.
- Entity Framework 5.0 beta.
- Dapper 1.8.
- LINQ to SQL.

The test scenarios were as follows

- Insert of N rows to the database, each row contains random data.
- Bulk select of N random rows from the database.



[14]

Figure 3: Insert of N rows to the database, each row contains random data

From the figure 3 they are four ORM technologies under experiment. The experiment starts with records between 1- 10 records then increase to 100 up until 10000. The graph shows that the entity frameworks are faster ORM tools as far as speed is concerned. This is probably because Entity framework performs some optimization before sending the data to the database. However, as we have noticed in the above experiment, the results showed that over small number of items Entity framework is the fastest in all tests, and tend to drop in speed as the number of items increases. This is of critical importance when picking an ORM. What this means is that if you have an application such as a website which is performing a bunch of small requests to your database to render your page Entity Framework 5 is going to be really quick (so long as you follow a few pointers on how to set up your context).

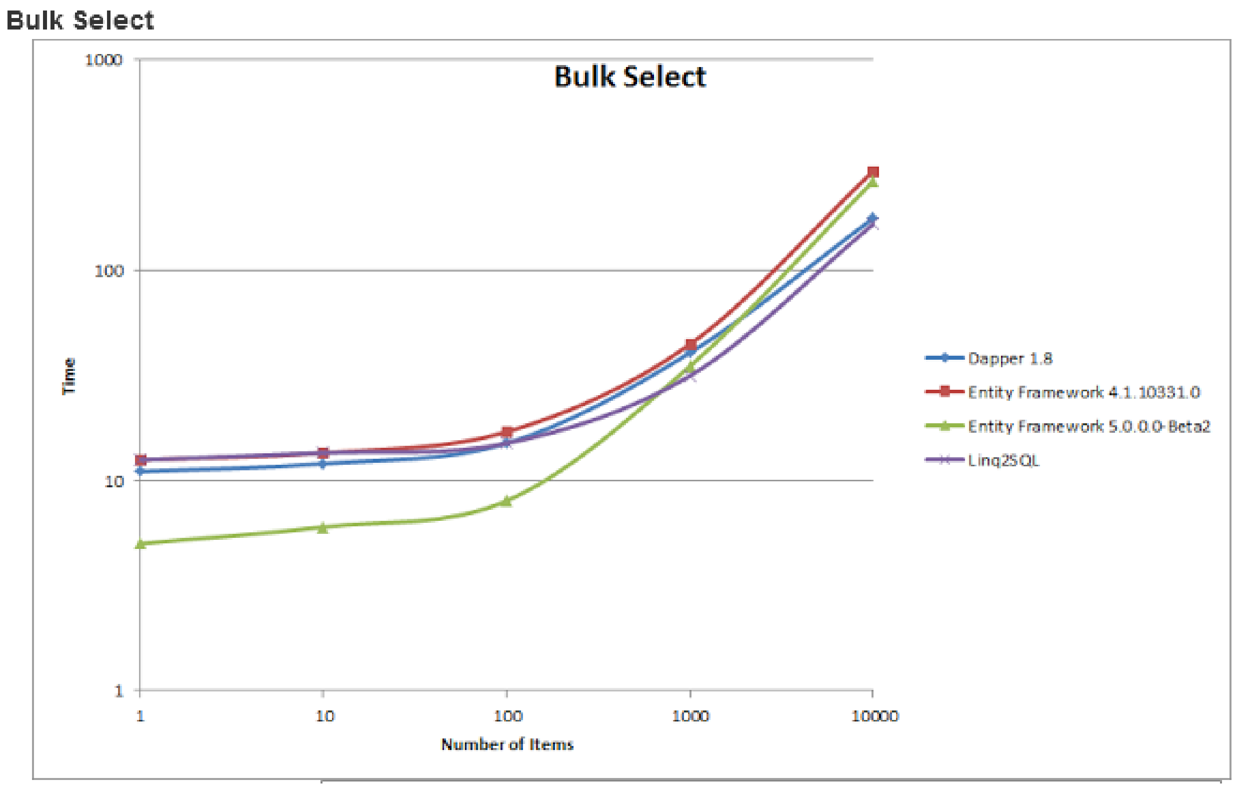


Figure 4: Bulk select of N random rows from the database [*14]*

From figure 4 it can be explained that when doing a *select where* statement with a small number of matches both Entity framework performs faster for small items. The result shows that bulk select for few items is much faster with Entity framework as compared to other ORM tools. Thus the phrase “smaller is better”, meaning for small item Entity framework is faster.

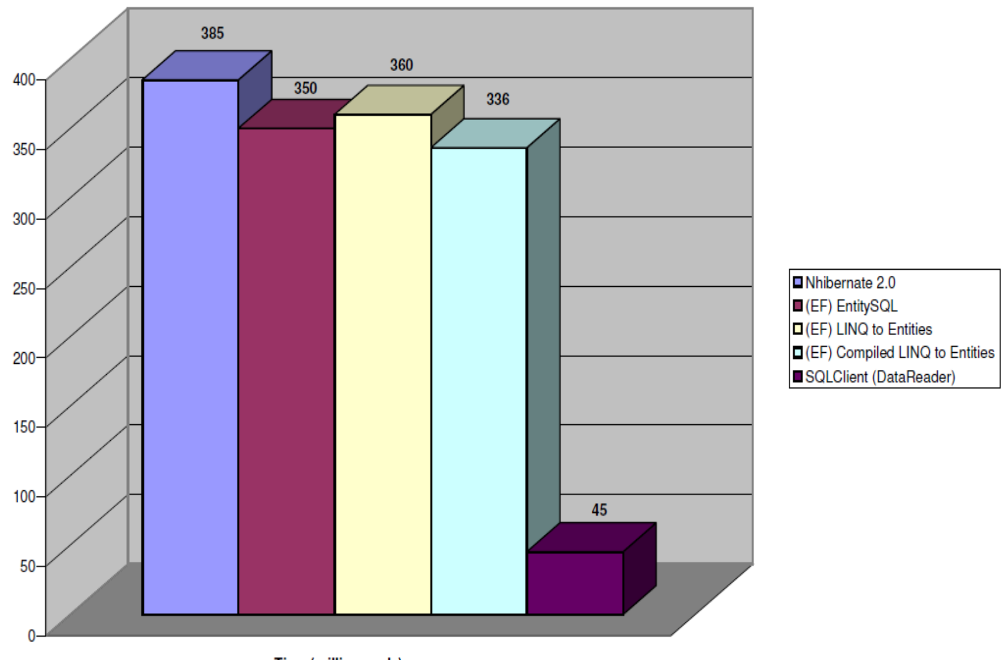
EF was compared with other ORM and in all cases it proved to be a better option for data abstractions. We interpreted the outcome of the experiment and we come with the following explanations;

The first big difference between the Entity Framework and LINQ to SQL is that the EF has a full provider model which means that as providers come online , you will be able to use the EF against not only SQL Server and SQL CE but also Oracle, DB2, Informix, MySQL, Postgres, etc [16]. EF is constantly updated and has very good support from Microsoft Corporation and can be used for different technologies. The second reason we investigated was that hat LINQ to SQL provides very limited mapping capabilities.

Second Experiment

The second experimental research from the literature review, a comparison was made between NHibernate and EF Component model testing the performance [16].

The results



[16]

Figure 5: A comparative evaluation of query performance with DataReader, NHibernate and Entity Framework

From the analysis of this experimental research the researchers concluded that the Entity Framework appears to perform well compared to NHibernate. In fact, the performance of its three querying mechanisms is slightly better than that of NHibernate. This could be explained by different caching strategies employed by the EF [16].

The EF was specifically structured to separate the process of mapping queries/shaping results from building objects and tracking changes. This makes it easier to create a conceptual model which is how you want to think about your data and then reuse that conceptual model for a number of other services besides just building objects. Meanwhile, Microsoft managed to build EDM awareness into a variety of other Microsoft products so that if you have an Entity Data Model, you should be able to automatically create REST-oriented web services over that model

(ADO.Net Data Services aka Astoria), write reports against that model (Reporting Services), synchronize data between a server and an offline client store where the data is moved atomically as entities even if those entities draw from multiple database tables on the server, create workflows[10].

EF is not longer just ORM tool, but also a tool that can be used in EAI for integration purposes.

2.5.2 Other Reasons Why EF Component Model was chosen

Coherent with .Net framework- Entity framework is very closely integrated with other parts of other parts of the .NET framework, including areas such as ADO.NET Data Services. NHibernate does not benefit from this degree of integration and given that it has its roots in the Java world, it does not fit snugly with the .NET framework. Until pretty recently nHibernate did not provide decent support for commonly-used paradigms such as LINQ and IQueryable which made it more of a struggle for .NET developers to start working with.

Testability - EF have always provided good support for testability as they allow you total control over your code design. The most recent version of the Entity Framework has addressed this issue by exposing the code generation mechanism which is driven by T4 templates. Templates are now available for a variety of class design strategies, including fully testable POCO classes.

Support and longevity -The Entity Framework looks like a fairly safe bet at the moment, particularly given the fact that Microsoft have addressed many of its earlier shortcomings in the most recent release. NHibernate has a large development community behind it and is continuing to mature as a technology, but much of this support may have arisen out of the lack of any serious ORM technology being provided by Microsoft [10].

2.6 HOW DOES EF COMPONENT MODEL WORK

EF is useful in three scenarios;

- If you already have the existing database or you want to design your database first then other parts of the application.

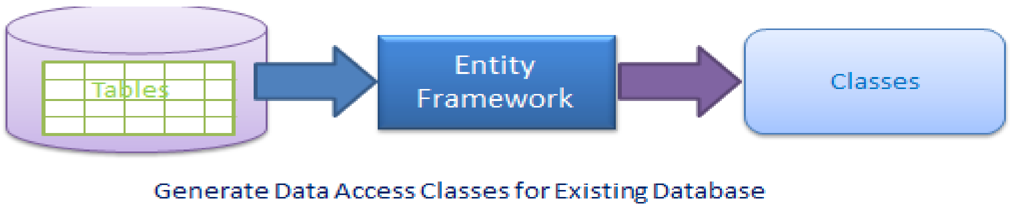


Figure 6: Generate Data Access Classes for Existing database

In above figure 6, EF Component model creates data access classes for your existing database, so that you can use these classes to interact with the database instead of ADO.Net directly.

- If you have domain classes and then create the database from your domain classes

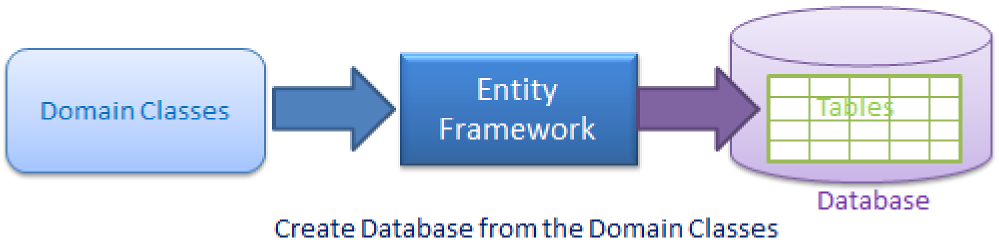


Figure 7: Create Database from Domain Classes

In the above figure 7 EF Component model can also creates the database from your domain classes, thus you can focus on your domain driven design.

- If you want to design your database schema on the visual designer and then create the database and classes.

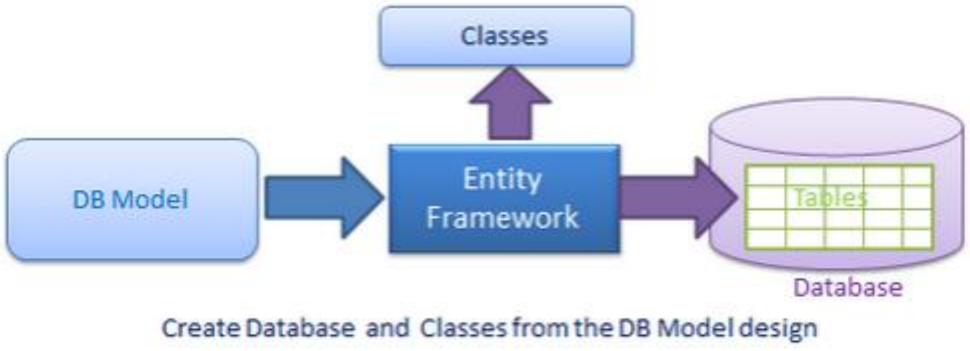


Figure 8: Create Database and Classes from the DB Model design

In the above figure 8 EF Component model provides you a model designer where you can design you DB model and then EF creates database and classes based on db model.

EF Component model includes three main parts: Domain class objects, Relational database objects and Mapping information on how domain objects map to relational database objects (tables, views & stored procedures). EF Component model allows us to keep our database design separate from our domain class design. This makes the application maintainable and extendable.

2.6.1 EF Component Model Architecture

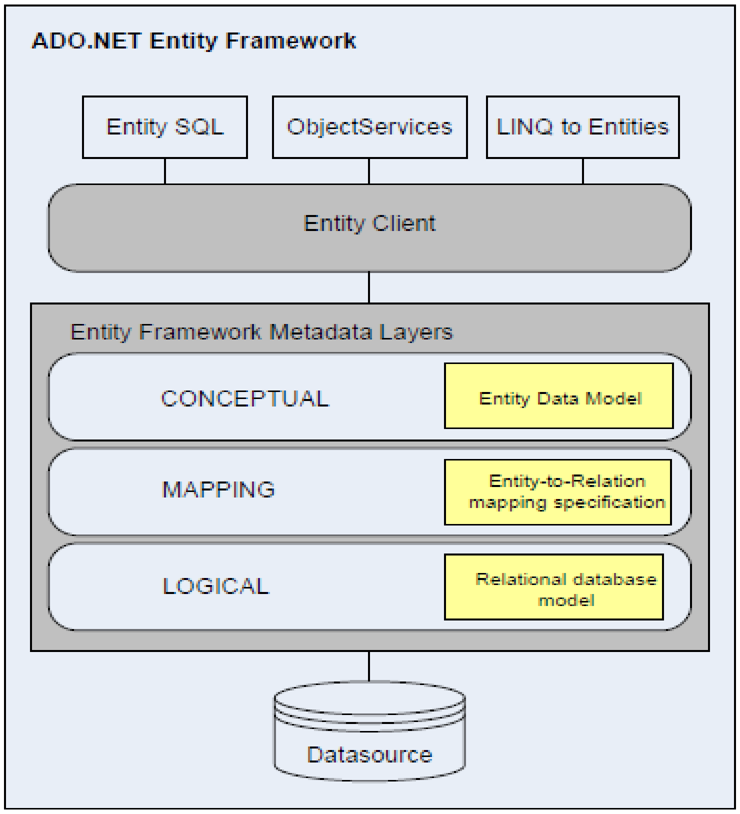


Figure 9: EF Component model architecture [16]

EF Component model use Entity Data Model to describe all the entities and relationship in the domain. The **EDM** consist three main parts- Conceptual model, Mapping and Storage model. Conceptual model is your model classes and their relationships. This will be independent from your database table design. Storage model is your database design model which includes tables, views, stored procedures and their relationships and keys. Mapping model contains information about how your conceptual model is mapped to storage model. The framework has **entity client provider** which work on the EDM and it is the data provider, convert LINQ to Entities or Entity SQL queries into SQL query which is understood by underlying database. It communicates with ADO.Net data provider which in turn sends or retrieves data from database. **Object Services** is another component important and is responsible for materialization which is process of converting data returned from entity client data provider (next layer) to an entity object structure. The framework has two types of methods for querying data, Entity SQL and LINQ to Entities. **LINQ to Entities** is query language used to write queries against the object model. It returns entities which are defined in the conceptual model. **Entity SQL** is again a query language same as LINQ to Entities. However it is little more difficult than L2E and also developer need to learn it separately [12]. All the components are presented in the figure above.

Advantages of the EF are as follows;

- Works with a variety of database servers (including Microsoft SQL Server, Oracle, and DB2).
- Integrates well into all the .NET application programming models including ASP.NET, Windows Presentation Foundation (WPF), Windows Communication Foundation (WCF), and WCF Data Services.
- Provides integrated Visual Studio tools to visually create entity models and to auto-generate models from an existing database. New databases can be deployed from a model, which can also be hand-edited for full control.
- Includes a rich mapping engine that can handle real-world database schemas and works well with stored procedures.

Benefits of EF are as follows; [15]

- Applications are freed from hard-coded dependencies on a particular data engine or storage schema by supporting a conceptual model that is independent of the physical/storage mode.

- Mappings between the object model and the storage-specific schema can change without changing the application code.

2.7 THE APPROACHES USED

The literature review follows a deductive approach using evaluation methods to inform the research phenomena. We used two evaluation methods the NS Theory and (GQM) approach; NS theory provides a suitable basis for evaluating the quality of information systems' business architectures. It is based on the observation that the presence of combinatorial effects influences the flexibility and evolvability of a system in a negative manner [7].The GQM provides a systematic method to find and define tailored metrics for a particular environment. It helps to interpret the values resulting from the collection of these metrics [35]. We also choose the EAI approach to design the prototype for evaluation.

Below is the literature review of the approaches we used

2.7.1 Enterprise Application Integration

There are numerous technologies used in the integration platform and some we have already discussed them, but our research focuses on Enterprise Application Integration. Enterprise Application Integration (EAI) approach was first introduced in the mid- 1990s [20]. EAI is the unrestricted sharing of data and business process among any connected applications and data sources in the enterprise [21]. EAI is a business computing term for plans, method, and tools that are aimed at modernizing, consolidating, and coordinating the overall computer functionality in an enterprise. From these definitions, it can be determined that EAI is the key approach to incorporate existing systems of a single organization into a broader application context.

Legacy system usually cannot communicate to one another in terms of their data sharing and the business logic. Among the reasons include:

- Heterogeneity of the programming language
- Diversity of application platform
- Complexity of legacy system in terms of their business logic and requirements
- Incompatible applications interface

For these reasons, EAI plays an important role to link the applications within the organization in order to achieve objectives such as [3]:

- Data and information integration within a company network.
- To provide single consistent integration interfaces for applications interaction purposes.
- To enable business logic sharing among each application to another.
- To centralize business policies and rules to encourage vendor independent among each of the integration participants.

In this research, Service Oriented Architecture (SOA) is chosen for our proposed system to implement concepts of EAI approach because of its suitability to integrate enterprise application.

2.7.1.1 EAI Architecture

EAI is an approach that has been researched over a decade now and many improvements on it has taken place. Many researchers have proposed two basic architectures of EAI which are Hub/Spoke Architecture and Bus Architecture.

Hub/ Spoke Architecture

Hub/Spoke architecture uses a centralized broker (Hub) and adapters (Spoke) which connect applications to Hub.

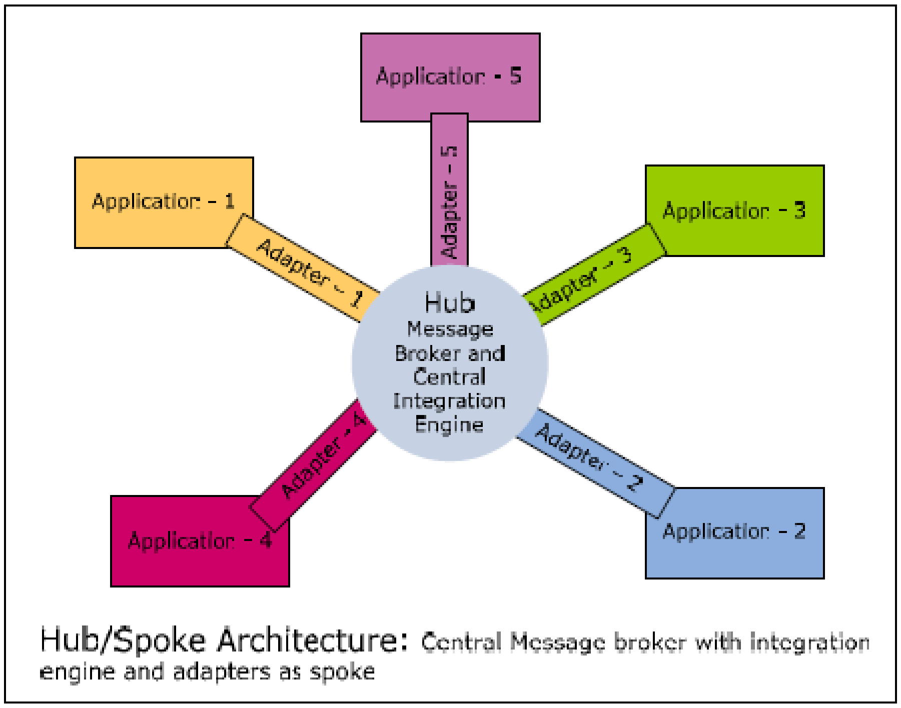

Application - 5
Adapter - 5
Application - 1
Adapter - 1
Application - 3
Adapter - 3
Hub
Message Broker and Central Integration Engine
Adapter - 4
Adapter - 2
Application - 4
Application - 2
Hub/Spoke Architecture: Central Message broker with integration engine and adapters as spoke

Figure 10: Hub/Spoke Architecture [21]

From figure 10, Spoke connects to application and convert application data format to a format which Hub understands and vice versa. Hub on the other hand brokers all messages and takes care of content transformation/translation of the incoming message into a format the destination system understands and routing the message. Adapters take data from source application and publish messages to the message broker, which, in turn, does transformation/translation/routing and passes messages to subscribing adapter which sends it to destination application(s) [3].

Bus Architecture

Bus architecture uses a central messaging backbone (bus) for message propagation [21]. Applications would publish messages to bus using adapters. These messages would flow to subscribing applications using message bus. Subscribing applications will have adapters which would take message from bus and transform the message into a format required for the application. Key difference between hub/spoke and bus topology is that for the bus architecture, the integration engine that performs message transformation and routing is distributed in the application adapters and bus architecture requires an application adapter to run on the same platform as the original applications.

2.7.2 Normalized System Theory

The NS theory is theoretically founded on the concept of stability from systems theory [7, 19]. According to systems theory, stability is an essential property of systems. For a system to be stable, a bounded input should result in a bounded output, even if an unlimited time period considered [7] This implies that the software architecture should not only satisfy the current requirements, but should also support future requirements. The Normalized Systems approach uses the systems theoretic concept of stability [7, 19] as the basis for developing information systems.

Information systems exhibiting stability with respect to a defined set of changes are called Normalized Systems [7], [19]. In contrast, when changes do require increasing effort as the system grows, combinatorial effects are said to occur [7], [19]. In order to obtain stable information systems, these combinatorial effects should be eliminated.

It has been formally proven that any violation of any of the following theorems will result in combinatorial effects that negatively impact evolvability [7]:

- ***Separation of Concerns***, which states that each concern (i.e., each change driver) needs to be encapsulated in an element, separated from other concerns.
- ***Action Version Transparency***, which declares an action entity, should be updateable without impacting the action entities it is called by.
- ***Data Version Transparency***, which indicates a data entity, should be updateable without impacting the action entities it is called by.

- ***Separation of States***, which states all actions in a workflow should be separated by state (and called in a stateful way).

Looking at one of the theorem ***Separation of concern*** describes that, any external technology which is used (i.e., any library, framework, or programming language which is not the background technology) should be considered as a change driver, since the evolution of that technology can be different than the background technology. As a result, the use of external technologies should always be separated in a separate construct. When programming manually, the programmer ultimately decides on the separation of concerns in constructs, and needs to take into account all implications of the theorems at all times. This makes it very unlikely to attain software free of combinatorial effects without the use of higher-level primitives or patterns. Therefore, NS Theory proposes a set of five elements (action, data, workflow, connector and trigger) that serve as patterns. These elements consist of modules (i.e., software constructs in a certain background technology such as JEE) which are separated based on the implications of the NS theorems (e.g., every external technology is separated). Based on these elements, NS software is generated in a relatively straightforward way through the use of the NS expansion mechanism. For this purpose, dedicated software (called NS expanders) was built by the Normalized Systems eXpanders factory (NSX).

This study evaluates the inclusion of EF into SOA architecture using Normalized System theory. Our contribution is to be able to promote broader integration which is stable and less combinational effects. As such NST is the base of our evaluation.

2.7.3 The Goal Question Metric Approach

The Goal Question Metric (GQM) was developed in response to the need for a goal-oriented approach that would support the measurement of processes and products in the soft-ware engineering domain. The GQM Paradigm (sometimes called the GQM approach) supports a top-down approach to defining the goals behind measuring software processes and products, and using these goals to decide precisely what to measure (choosing metrics) [22].

The GQM approach is a systematic method to find and define tailored metrics for a particular environment [35]. The GQM approach helps to identify the reasons why particular metrics are chosen. It also helps to interpret the values resulting from the collection of these metrics .It consists of the following steps [35];

- Starting from the definition of *goals* that should be achieved by the conducted measurements. A goal is defined using a template which consists of the following parts:
  - **Purpose** What should be achieved by the measurement?

    Issue Which characteristics should be measured?

    Object Which artefact will be assessed (this may be a product, a process or a resource)?

    Viewpoint From which perspective is the goal defined (e.g. the end user or the development team)?
- The next step is to define *questions* that will, when answered, provide information that will help to find a solution to the goal.
- To answer these questions quantitatively every question is associated with a set of metrics. It has to be considered that not only objective metrics can be collected here. Also metrics that are subjective to the viewpoint of the goal can be listed here.

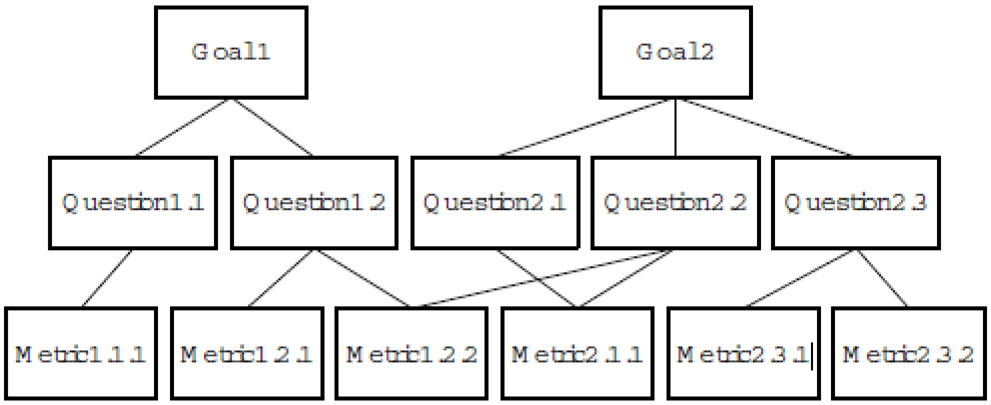


Figure 11: GQM structure as defined in [35]

The final objective must be improvement of products and processes. Measurement should be viewed as an infrastructure technology that is necessary to achieve systematic improvement [35] Measurement is necessary to characterize the current state of affairs quantitatively; i.e., to derive a quantitative baseline. A "quantitative baseline" is nothing other than a model that captures some concrete information about the status quo. For example, the statement "90% of all faults in a design document are detected by project XYZ's design inspections" is a quantitative baseline [36].

2.8 SUMMARY

This chapter provided a description of the integration technologies. It also gave the description of the Microsoft technologies for integration such as EF Component model and Windows Communication Language (WCF). The chapter also briefly introduced the EAI which is the approach we used to integrate the legacy systems and NS Theory which is used to evaluate the prototype. The GQM is the standard metric which is used to evaluate the criteria mentioned above. The SOA is implemented using WCF in the integration processes and have problems in extracting data from the legacy systems codes. As such there is a need for an approach that bridges the gap between legacy systems and the SOA.

CHAPTER 3: RELATED WORKS

3.0 INTRODUCTION

This chapter discusses the approaches that are being used or that have been used to solve problems in integration. This chapter identified four approaches used in solving integration problems; these are Migration approach, point to point, the EAI approaches (Enterprises service Bus (ESB) and Message Oriented Middleware (MOM)). Below are the discussions of these approaches

3.1 MIGRATION APPROACH

A web service is middleware used to integrated legacy systems [13]. Web service may be integrated with the legacy systems by presenting a stack of interrelated protocols such as SOAP, WSDL and UDDL to support integration. **SOAP** (Simple Object Access Protocol), is the communication protocol utilized by the web services. It is an XML data encoding standard and platform independent. **WSDL** (Web Service Description Language) is used to define the abstract and concrete interface of web services. Like SOAP it understands XML grammar and platform independent. **UDDI -**The UDDI is a Business Registry standard for indexing Web Services, and enables businesses to find preferred services and transact with one another [14].

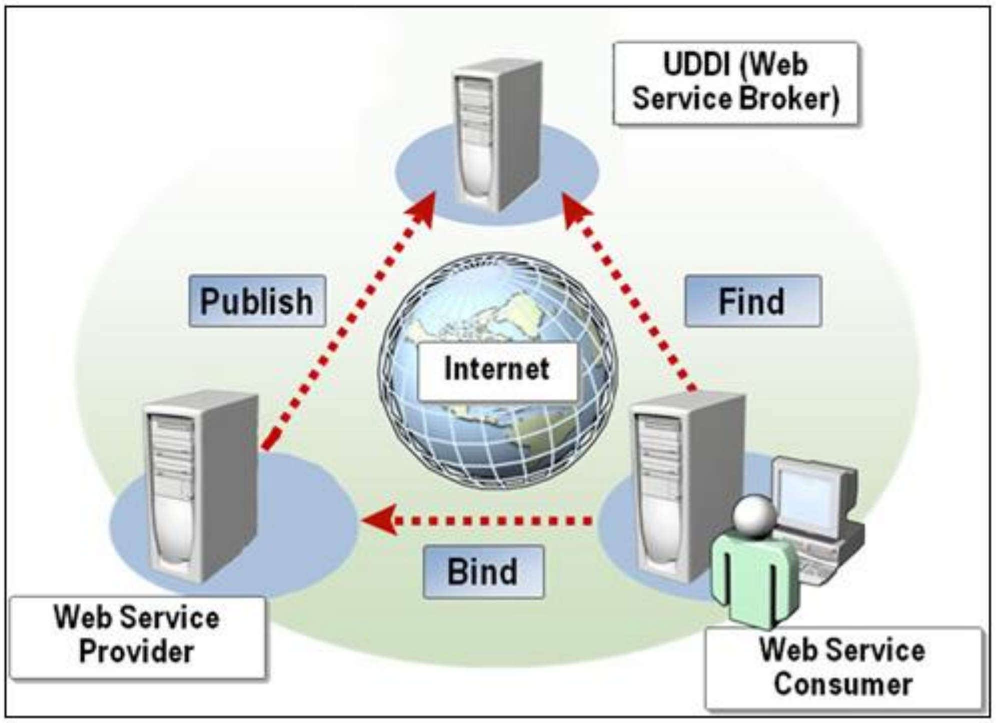


Figure 12: Life Cycle of a Web service

Figure 12 shows life cycle of a Web service consists of the following fundamental operations:

- **Publishing services**: In this operation, the service providers publish their services to a service broker (UDDI). Published services contain information about the location of the service providers, supporting information, and service information definition.
- **Finding services**: In this operation, a service consumer accesses the service broker to find the required services.
- **Binding services**: In this operation, the binding process authenticates the consumers and binds them to the specific services provided by the service provider.

A web service is the middleware but the gap between the legacy systems and web services is dealt by re-engineering the legacy systems. Re-engineering the legacy system first, then writing the legacy system from scratch and then migrate to a new web environment. The legacy systems had to be re- engineered and their behavior needs to be specified in terms of well defined object oriented interface. After re-engineering legacy systems needs to be migrated into service web so that they can be exposed and consume the web service. The legacy system will now have to be wrapped. Wrapping means surrounding existing data, individual programs, application and interface with new interface .In other words, this gives old components a new operation look. The wrapped components acts as server, performing some functions required by an external client that does not need to know how the service is implemented.

This approach involves practically moving an existing, operational systems to a new platform; retain the legacy systems and causing a little disruption to the existing operational and business environment as possible. This is a great challenge and can cause data inconsistent with the database. Some of the weaknesses of migration approach are listed below [15];

- Rewriting legacy systems can amount to reinventing these systems from scratch. This can be extremely costly and sometimes result in solutions that are less reliable and cost-effective than the original.
- Data mismatch and Inconsistence- To populate the target database, there is a lot of work to do. Data must be mapped at instance level.
- Time consuming-A lot of time might be spent on testing which is ongoing process during migration.

3.2 POINT TO POINT INTEGRATION MODEL

The use of unique connectors in the form of programming scripts is another integration approach that has been advocated by other researchers. The legacy systems and the middleware had to be bridged using programming script. The approach uses connectors for integration and automation [1]. This connector handles all data transformation, integration and other related services that must take place between only the specific pair of integration [24]. When used with small infrastructure, where two or three systems must be integrated, this model can work quite well, providing a lightweight integration solution tailor made to the needs of the infrastructure.

This approaches of point to point integration present several limitations such as follows

- Requires number of different programming scripts for particular systems. For instance each application has its own script to exchange data with middleware.
- When there is a new application introduced, an programming script needs to be modified to account for the differences in protocols and formats supported by the new applications.
- Remember that each of these connectors must be separately developed and maintained across system version changes, scalability changes, and more (or, in some cases, even purchased at high cost from a vendor), and the unsuitability of point-to-point integration for complex enterprise scenarios becomes painfully clear.

3.3 ENTERPRISE SERVICE BUS (ESB)

Enterprise Service Bus (ESB) is a standard integration platform that exploits web services, messaging middleware, intelligent routing, protocol transformation, and service mapping in a distributed environment to connect and coordinate the interaction of diverse system applications [17]. In an integration process, ESB supports service request and response communication between the service provider and the service consumer via a service bus (messaging backbone). Hence, it also support more-complex message exchange patterns (MEPs) via the event-driven and standard based bus.

ESB Architecture

The architecture of the ESB takes the form of Bus Architecture of EAI but with some additional advance enhancement. ESB is implemented in five tier distributed architecture. The tiers include enterprise service bus; web services providers, database servers, application interface and application servers. The figure below depicts the ESB architecture;

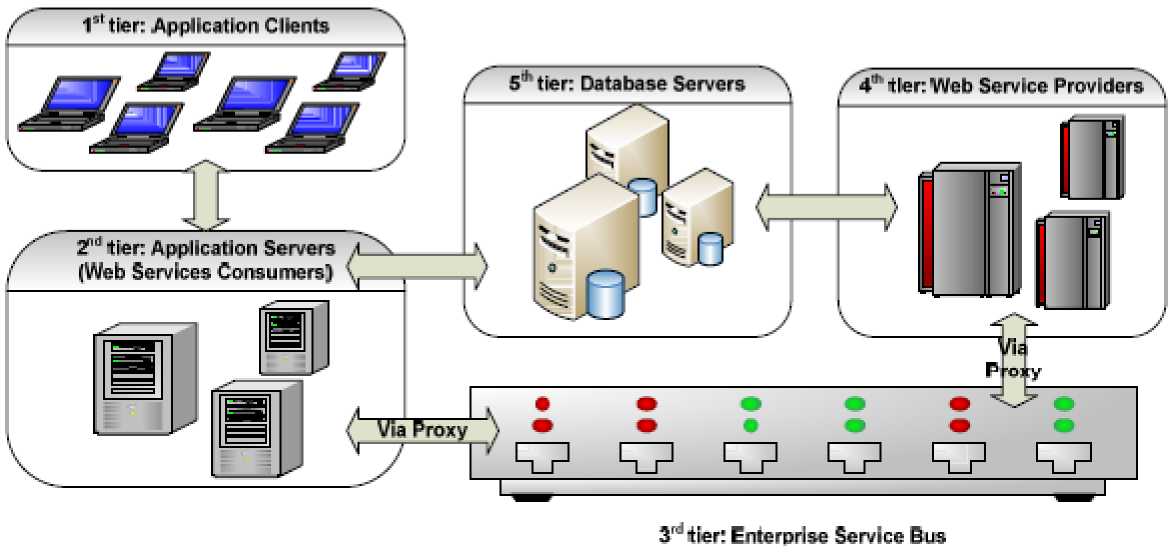

1st tier: Application Clients
5th tier: Database Servers
4th tier: Web Service Providers
2nd tier: Application Servers (Web Services Consumers)
Via Proxy
Via Proxy
3rd tier: Enterprise Service Bus

Figure 13: ESB Architecture [21]

The key benefits to implement integration using ESB are: [3]

- Avoiding failure caused by single centralized broker since ESB using distributed messaging services and this encourage zero downtime of service availability.
- Less development time and cost since more configuration than integration construction in ESB implementation.
- One time application integration configuration to provide ready for reuse service type, this enable same services used in different purposes.

There are some other drawbacks for the bus, which are:

- Since ESB implementation requires more configuration than coding, it usually involves more hardware to handles processing.
- Learning curves of developers on ESB take more time than simple point-to-point messaging because new skill and knowledge required configuring ESB.

- Service Management component is able to control service version, but required ongoing management to avoid tight coupling among the services.

3.4 MESSAGE ORIENTED MIDDLEWARE

Message Oriented Middleware (MOM) is one of the cornerstone foundations that distributed enterprise systems are built upon. MOM can be defined as any middleware infrastructure that provides messaging capabilities. A client of a MOM system can send messages to, and receive messages from, other clients of the messaging system. Each client connects to one or more servers that act as an intermediary in the sending and receiving of messages. MOM uses a model with a peer-to-peer relationship between individual clients; in this model, each peer can send and receive messages to and from other client peers. MOM platforms allow flexible cohesive systems to be created; a cohesive system is one that allows changes in one part of a system to occur without the need for changes in other parts of the system.

Architecture of MOM

MOM is a client/ server infrastructure that operates on the concept of passing and queuing messages, where passing messages is non-blocking [21]. It is software to be installed in both portion of client and server workstation, which provide transparent communication services through asynchronous calls or publish-subscribe between client and the server screen.

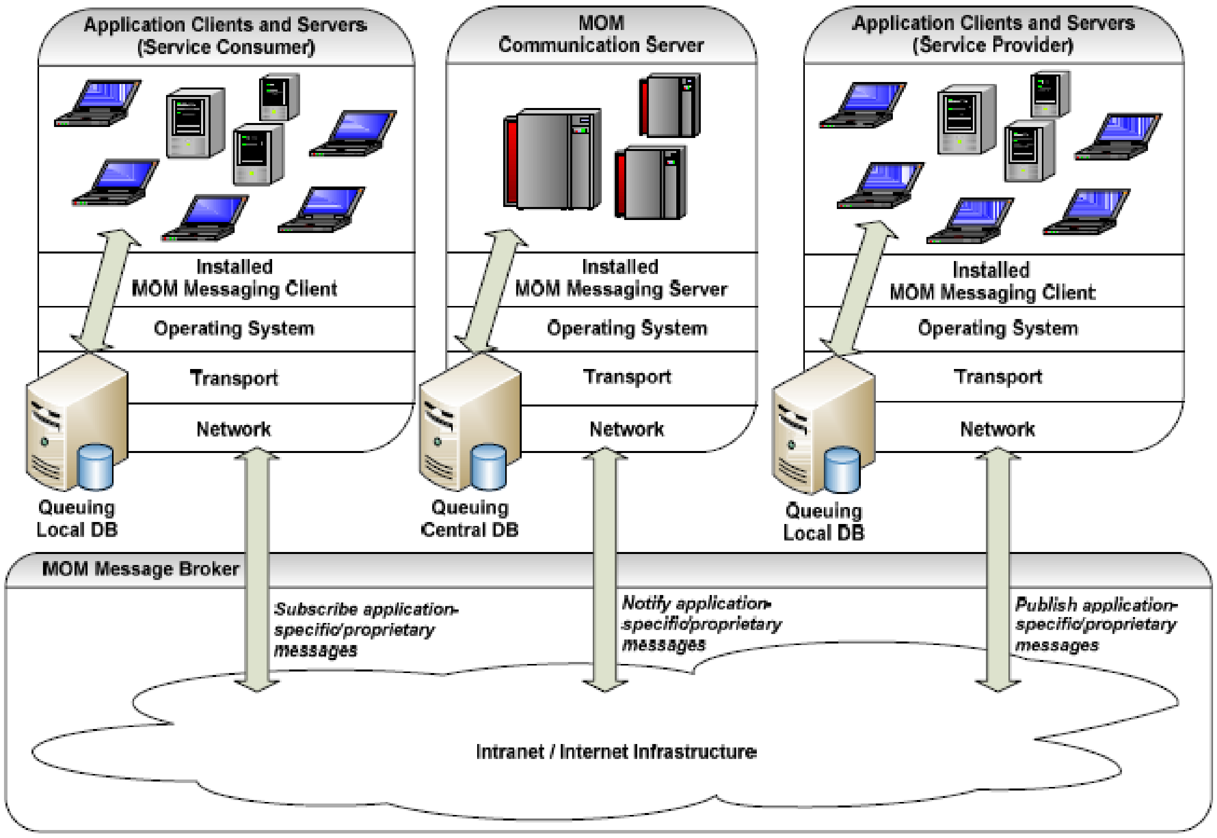

Application Clients and Servers (Service Consumer)
MOM Communication Server
Application Clients and Servers (Service Provider)
Installed MOM Messaging Client
Installed MOM Messaging Server
Installed MOM Messaging Client
Operating System
Transport
Network
Queuing Local DB
Queuing Central DB
Queuing Local DB
MOM Message Broker
Subscribe application-specific/proprietary messages
Notify application-specific/proprietary messages
Publish application-specific/proprietary messages
Intranet / Internet Infrastructure

Figure 14: MOM architecture [24]

The primary advantages of MOM are such as: [3]

- The service consumer and provider are not required to connect to the network at the same time, and this increase the flexibility and reliability of the applications being integrated.
- Typically MOM provides built-in transformation mechanism to transform the received messages to be the application server’s native formats. That means the message sender need not convert the message format before the transmission.

However, MOM has some critical limitations, which are:

- MOM may incompatible with other MOM implementation due to the use of application specific or proprietary messaging structure in a single MOM implementation. This could affect the flexibility, interoperability, and portability when a new application joins into the integration domain.

- The tendency asynchronous architecture of MOM may not fit some synchronous based inter-application communications. This is because some application requires immediate response from the opponents before the next process to be started.

Related works of integration approaches has highlighted some approaches that have been used for integration. This research however does not discard these integration approaches but rather our contribution is to improve integration by incorporating legacy systems in the process of integration. The migration approach and the point-point integration approach attempted to use reengineering and programming scripts to cater legacy systems but however these approaches have shortcomings that were discussed above. The ESB and MOM are types of EAI but however they do not cater for legacy systems. It is from this background that our research exploits the EAI to include legacy systems by incorporating the EF component model to improve broader integration.

3.5 SUMMARY

Related works were discussed in this chapter and referred and consulted in order to understand and research the problem. The Migration was the approach used whereby the legacy systems had to be re-engineered so that they become integrated through use of standard protocols such as the SOAP, UDDI, and WSDL. The point to point was another approach used to integrate legacy systems which involves writing programming scripts such java script and vb scripts to integrate the legacy systems. ESB and MOM were also integration approaches discussed. However all these approach shows some gap and that gap can be covered using EF Component model. EF Component model is used to extract data and query the database of the legacy systems and pass to the middleware (WCF) for integration purposes. However it is not clear on how to use the EF Component model to integrate legacy systems in SOA environment.

CHAPTER FOUR: RESEARCH METHODOLOGY

4.0 INTRODUCTION

Our research has been conducted using the design science methodology. Our research goal is to evaluate EF to automatically identify violations to the Normalized Systems design theorems to promote broader integration. The design science methodology is appropriate in this case, since design science is primarily aimed at solving problems by developing and testing artifacts, rather than explaining them by developing and testing theoretical hypotheses. In this research we used the Normalized System theory to evaluate EF to identify potential issues with respect to stability and evolvability.

The research is concerned with build and evaluates phases of an instantiation artifact. The importance of building artifacts has been emphasized by Newell and Simon, by writing: “Each new program that is built is an experiment; it poses a question to nature, and its behavior offers clues to the answer” [27].

4.1 DESIGN SCIENCE RESEARCH METHODOLOGY

This model is an adaptation of a computable design process model developed by Takeda, et al. [26] .The design-science paradigm has its roots in engineering and the sciences of the artificial [29]. It is fundamentally a problem solving paradigm. It seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, management, and use of information systems can be effectively and efficiently accomplished [28].

To achieve a true understanding of and appreciation for design science as an IS research paradigm, an important dichotomy must be faced. Design is both a process (set of activities) and a product (artifact).a verb and a noun [30]. It describes the world as acted upon (*processes*) and the world as sensed (*artifacts*). This Platonic view of design supports a problem solving paradigm that continuously shifts perspective between design processes and designed artifacts for the same complex problem. The design process is a sequence of expert activities that produces an innovative product (i.e., the design artifact). The evaluation of the artifact then provides feedback information and a better understanding of the problem in order to improve both the quality of the product and the design process. This build-and-evaluate loop is typically iterated a number of times before the final design artifact are generated [31]. During this creative process, the design-science researcher must be cognizant of evolving both the design process and the design artifact as part of the research.

Guidelines for Design Science in Information Systems Research

We used the seven guidelines of design science research proposed by Lan R. Hevner [28]. Table below summaries the seven guidelines. Each guideline is discussed in detail below

Design- Research Guidelines	
Guidelines	**Description**
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via

	well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Figure 15: Design-Science Research Guidelines *[28]*

How the Guidelines were used in Our Study

Guideline 1: Design as an Artifact - An innovative research instrument (guidelines from the literature and claims from the product vendors) was designed for evaluation of the prototype. EAI approach was used to design the prototype.

***Guideline 2: Problem Relevance* -**The use of research instrument in IS evaluation provided a better understanding of the need for different evaluation methods at different stages of the IS development.

Guideline 3: Design Evaluation - Comparison of the outputs of several evaluation methods shows coherence in research findings. Experimental results from evaluating a prototype are compared with results from the prototype system is used in practice.

Guideline 4: Research Contributions

A novel research instrument was designed by focusing on the practical utility of that IS evaluation instrument for both researchers and practitioners. The instrument also accommodates the differences between practice-driven and research-driven goals.

Guideline 5: Research Rigor

Design is guided by literature on evaluation methods and integration methods from several research fields particularly IS design science research.

Guideline 6: Design as a Search Process

Various evaluation methods were studied and piloted in order to form an effective multi-method research instrument. Outcomes of the range of evaluation methods that were used, were compared with each other to assess face validity of the outputs.

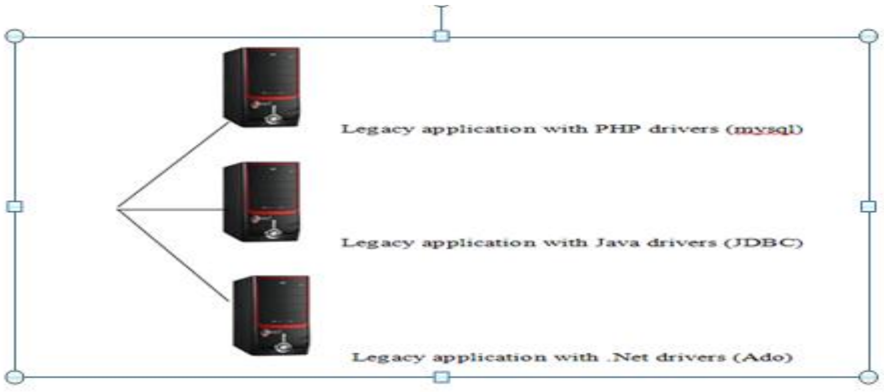
Guideline 7: Communication of Research

Researchers involved in conducting the evaluation perceived the research instrument as useful during the iterative design cycles of the intelligent products system.

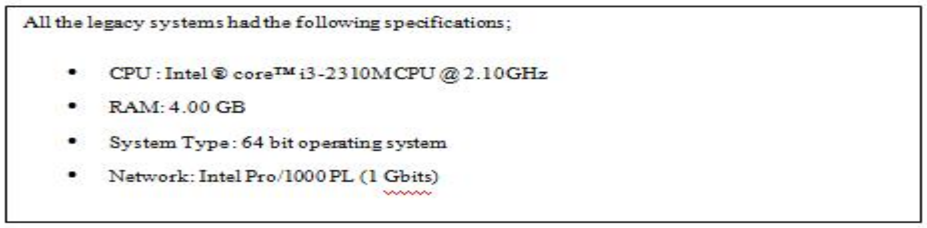
4.2 TEST ENVIRONMENT

Our testing environment used a 3 separate legacy applications and a web server setup with the database on a separate machine from the client application. Machines are in the same rack, so network latency is relatively low, but more realistic than a single-machine environment. We used three legacy systems, the first legacy system was created using PHP, the second legacy system was created using Java and the third legacy system was designed using C-sharp. Figure 16 below shows in diagram the structure of the test environment which involves;

LEGACY APPLICATIONS



Legacy application with PHP drivers (mysql)
Legacy application with Java drivers (JDBC)
Legacy application with .Net drivers (Ado)



All the legacy systems had the following specifications;

- CPU : Intel ® core™ i3-2310M CPU @ 2.10GHz
- RAM: 4.00 GB
- System Type : 64 bit operating system
- Network: Intel Pro/1000 PL (1 Gbits)

Figure 16: Test Environment

APPLICATION SERVER

<u>Software Environment</u>

Entity Framework and WCF environments

- OS Name: Windows Server 2008 R2 Enterprise SP1.
- Visual Studio 2010 – Ultimate.
- For more on the Software platforms used see ***appendix A***

Hardware Environment

- Dual Processor: Intel(R) Xeon(R) CPU L5520 W3530 @ 2.27GHz, 2261 Mhz8 GHz, 4 Core(s), 84 Logical Processor(s).
- 2412 GB RAM.
- 136 GB SCSI250GB SATA 7200 rpm 3GB/s drive split into 4 partitions.

The experiments have been conducted in aid of our statement that an EF component promotes broader integration in SOA environment. However, due to the high abstractions being done between two incompatible systems (legacy application and WCF), the supplementary logic has a significant influence on the response time of the software framework. Evaluating these experiments can help with guidelines on how to use EF model in broadening integration.

It should be noted that because of the differences between programming languages, software frameworks and EF implementations these tests should not be considered as an exact benchmark for performance, but more as a validation for the negative trend that is common regardless of technology used.

4.2.1 Prototype Architecture

From the two architecture of EAI discussed in the literature review; the bus architecture and hub/spoke architecture, the architecture of the prototype is based on Hub/Spoke architecture. The figure below explains the architecture;

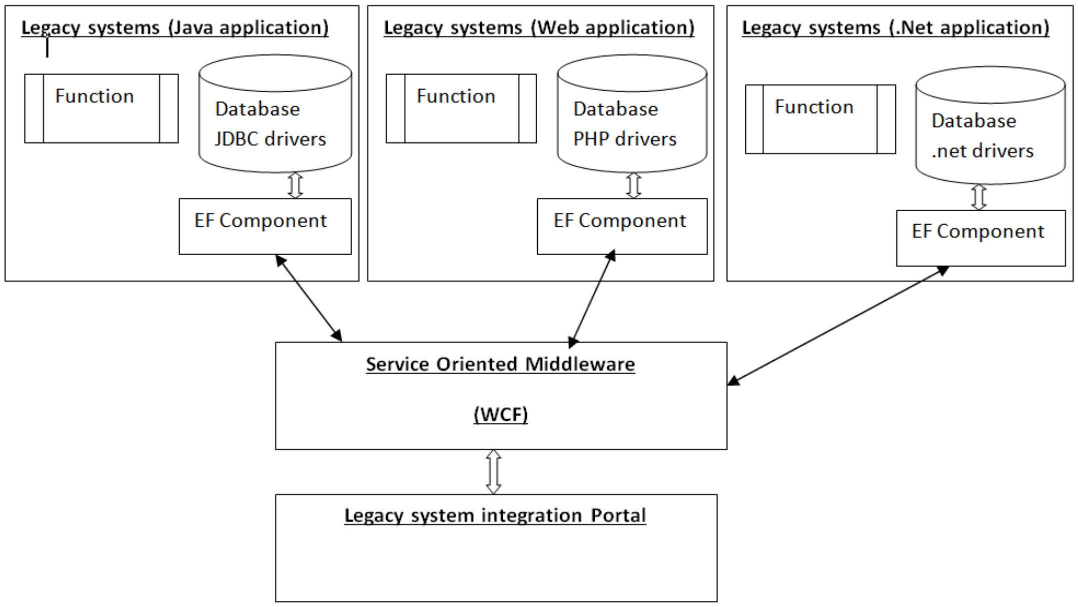

Legacy systems (Java application)
Function
Database
JDBC drivers
EF Component
Legacy systems (Web application)
Function
Database
PHP drivers
EF Component
Legacy systems (.Net application)
Function
Database
.net drivers
EF Component
Service Oriented Middleware
(WCF)
Legacy system integration Portal

Figure 17: Proposed architecture

The figure 17 shows how the legacy systems are exposed using EF Component model. The legacy systems were modified using EF component model to bridge a gap between the legacy applications and SOA middleware (WCF). All the legacy applications were hosted on a portal. The portal is the central place containing links to the three legacy systems in the figure above. We made this portal to be accessed through the ftp platform. EF component model as an ORM tool, enabled us to work with relational data as domain-specific objects, eliminating the need to come up with different programming codes to access data. We issued queries using LINQ, then retrieve and manipulate data. ***See Appendix B for snapshots***.

Having accessed the data from the legacy systems with EF component we used WCF to enable interoperability among our systems. WCF enables two or more applications to communicate across platforms in a distributed environment. ***See figure 2, Chapter literature review***. The data can be from any legacy systems regardless of different programming platform. For instance legacy system (Java) can be consumed by any legacy systems through the portal. ***See Appendix B for snapshots***.

In the design phase the prototype is developed, which serves as a proof-of-concept for the claims of EF component in the literature review. We evaluate the prototype by using NST. The purpose of building this prototype is threefold. We use the guidelines of the literature to design the prototype. In this way we can first evaluate whether the guidelines provide enough guidance or whether they need more detailed specification. This also shows the guidelines' limitations regarding scope. Then we have the possibility to provide additional guidelines to extend the scope.

Secondly, we want to evaluate whether following the guidelines results in a broader integration that is more evolvable. To do this we propose a set of (anticipated) changes that we apply to the prototype. This list is based on the theoretical changes that we use to derive the combinatorial effects. After implementing these changes, we evaluate for each change whether it causes (a) combinatorial effect(s) or whether following the guidelines prevented the occurrence of combinatorial effects. In this way we add a proof-of-concept on top of the theoretical proof for our guidelines. Moreover, this way of working might reveal additional issues and/or combinatorial effects.

Thirdly the prototype is developed iteratively. If we find possible ways to improve the guidelines or additional combinatorial effects, we adjust or extend our guidelines and integrate the solution in the prototype. In this way, the final prototype will be evolvable (anticipated changes can be applied without causing combinatorial effects). However, in this study a second evaluation of the prototype was not conducted due to the limited scope of the study.

4.3 THE EXPERIMENTS

We now discuss the evaluation phase of the design research process. In order to evaluate our prototype, we considered non function requirements which hinders integration such performance, abstraction, evolvability and scalability.

The Normalized Systems Theory

We evaluated the designs with regard to their evolvability by using the NS Theory. NS Theory is originally applied in software design [7, 19], but has shown its relevance in business process design and enterprise architectures [34]. We evaluated EF component against criteria such performance, scalability stability and evolvability. We used the NST theorems (***separation of concerns, separation of states, version transparency and instance traceability***) [7, 19, 34] in this dissertation to search for any via lotion of these theorems.

During evaluation we searched for combinatorial effects in the prototyped designs. These combinatorial effects represent violations of the NST theorems. We use the NST theorems to evaluate EF Component with regard to evolvability and stability.

We searched the combinational effects by proposing changes to the designs of the prototype and evaluating their impact on the different designs by illustrating different contexts. If the impact is not only proportional to the change, but also proportional to the size of the system, we report a combinatorial effect.

The Goal- Question-Metric Approach

The GQM approach is a systematic method to find and define tailored metrics for a particular environment [35]. The GQM approach helps to identify the reasons why particular metrics are chosen. It also helps to interpret the values resulting from the collection of these metrics .It consists of the following steps [36]. We used this approach as our standard measure on all the experiments on issues such as mapping, performance and scalability.

A solution that facilitates this comparison is to define project specific metrics. These metrics can be founded by using the GQM method and creating questions that involve specific scenarios and tasks that can be found in the particular project and domain. The disadvantage of the metric is however that they are consequently only valid in the narrow domain for which they were defined. Instead of being universally valid they only provide results for the particular goals of the evaluation plan.

Below are a summary of the experimented we conducted and the evaluation using NST to find if there are any violations to the architecture of the design. If they are no combinational effects, EF would have answered the challenge of broader integration.

4.3.1 Experiment 1: Mapping

To measure the mapping of the prototype we defined our first measurement goal according to GQM method as follows:

Goal 1: Purpose: Comparison.

Object: Prototype.

Issue: Mapping.

Viewpoints: The software development and maintenance team.

The following question was elaborated to cover the first goal; the question is of rather general nature as they ask for common development efforts.

Question 1.1: How big is the initial effort to understand the technology and to create a design how the technology can be implemented into the system?

Measuring Criteria

In order to answer this question we will gather the following metrics:

M1.1.1: *Person-days* to design and implement the mappings that are needed for a special application (prototype). This includes the effort that is needed do initial work.

M1.1.2: *Amount of aspects* that have to be implemented manually (such as transaction handling, caching, queries and referential integrity).

M1.1.3: *Amount of workarounds* that were needed to implement the technology in the system.

M1.1.4: Amount of time, measured in *milliseconds*, spent to implement the workarounds that are needed.

4.3.2 Experiment 2: Abstraction

EF Component model performs faster in abstracting and will not have impact on the run time behavior of the systems

To measure the performance we defined our goal according to the GQM method as follows

Goal 2: Purpose: Comparison.

Object: Different types of legacy systems database drivers (JDBC, Ado.Net and My SQL).

Issue: Performance.

Viewpoints: The software development and Maintenance team.

Question 2.1: which impact does the choice of the legacy system database drivers have on time that is needed to start the application?

To measure this question we derived the following metrics

Metric 1.1.1: Time to initialize the application measured in *milliseconds*. This metric is measured from the entry of an special *init-method* until its end. Within this method initializations components such as database connections and caches are executed.

The interpretation of the results of this metric will be based on the assumption that a short initialization time is better than a long one. However it is hard to say whether a vast or a narrow distribution where the values are rather small respectively large is more convenient. Each section provides a response time value upon running the test which indicates the time from start to finish performing the operations as measured by a regular clock. In order to avoid inaccuracies all tests were performed several times and an average response time value was calculated.

The test was divided into three sections which were benchmarked:

- Create entity instances and the relationships between them.
- Browse through records and their relationships and update them.
- Retrieve records and their relationships and delete them.

To prove the results' independence from a specific software platform the experiments were done for several software platforms (JDBC; ADO.NET and MYSQL). Three legacy application systems were used written in different programming languages; PHP using mysql drivers; Java using JDBC drivers and C-sharp using ado.net.

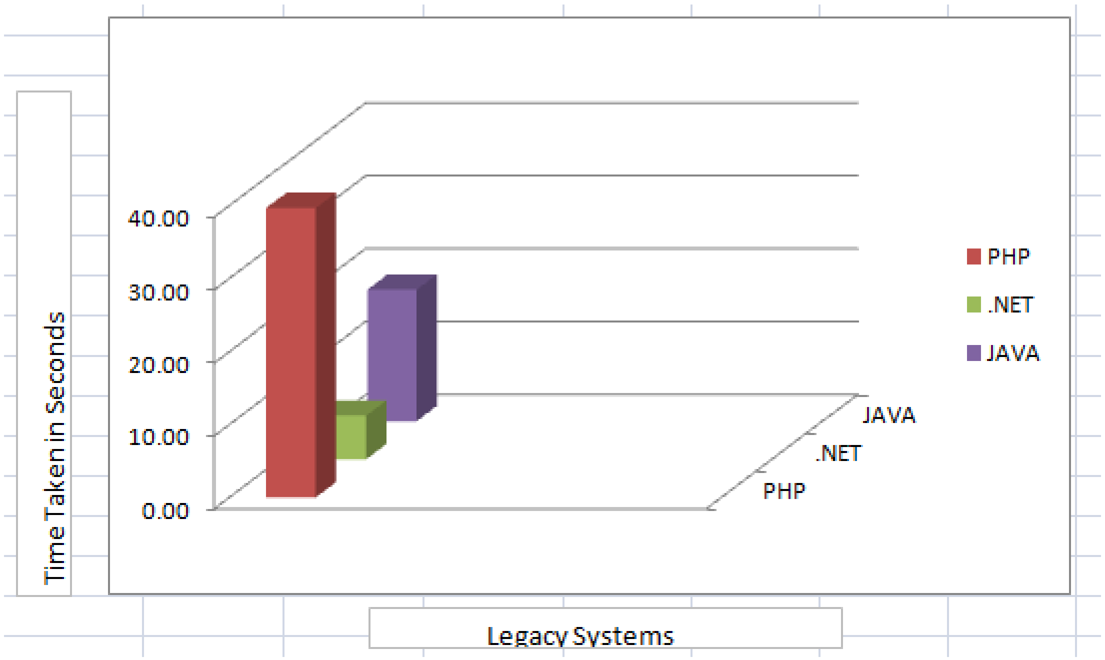
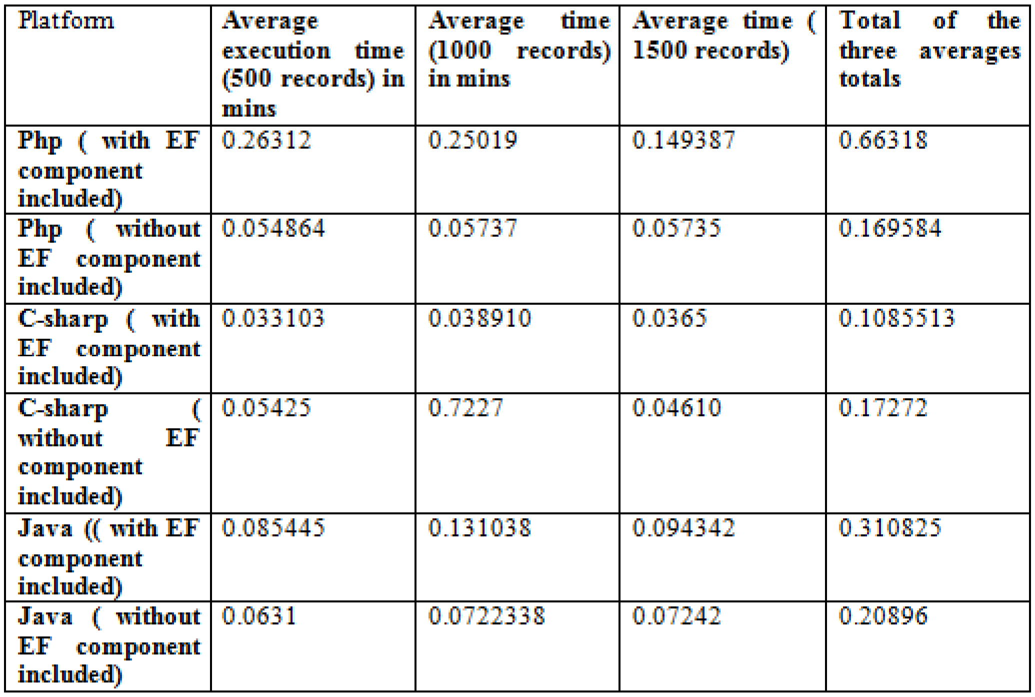


Figure 18: Performances charts with experiment results

The performance chart shows that the difference between response times when the EF component is used on the legacy application with PHP database drivers and Legacy application with .Net drivers (c-sharp). For example the PHP test results imply that the average response time when EF component was used (39.7908 seconds) which is very poor than the response time when C-Sharp which has a .Net database driver is used. For more results see the tables below.

Table 1: Legacy systems with EF vs Legacy systems without EF



Platform	Average execution time (500 records) in mins	Average time (1000 records) in mins	Average time (1500 records)	Total of the three averages totals
Php (with EF component included)	0.26312	0.25019	0.149387	0.66318
Php (without EF component included)	0.054864	0.05737	0.05735	0.169584
C-sharp (with EF component included)	0.033103	0.038910	0.0365	0.1085513
C-sharp (without EF component included)	0.05425	0.7227	0.04610	0.17272
Java ((with EF component included)	0.085445	0.131038	0.094342	0.310825
Java (without EF component included)	0.0631	0.0722338	0.07242	0.20896

Table 2: Performance response Comparison

Platform	**EF model**	**No EF model**	**Time Slower**
PHP	0.66318	0.169584	3.08
C-Sharp	0.108513	0.17272	-1.06
Java	0.310825	0.20896	207

The experiment results are aligned with our statement that EF works with a variety of databases servers (including Microsoft Server, Oracle and DB2); different software for databases may be used for different legacy systems, Database Management Systems (DBMS) such as oracle, MySqL, sql server, Java DB.

Discussions of the Experiment

From the experiment, we noted that EF component can work with variety of databases servers, but it is very faster with, .Net applications such as c-sharp in terms of execution speed. The reason being that EF component is from Microsoft.Net Applications, as data conversion is not a problem. Unlike other drivers like PHP and Java we noted that for them to be faster, they need to be Entity framework enabled, if not for faster data abstraction we needed an EF component from Entity framework 4.0 and beyond.

This experiment was evaluated with NS theory and found that it does not violets the theorems of NST. The first theorem ***separation of concern*** requires that every change driver or concern is separated from other concern. In this experiment were able to change different database drivers from the legacy systems without actual changing anything from other applications like WCF. Second theorem ***(data version transparency)*** states that data is communicated in a transparency way without affecting other modules in the system; this was with our prototype we switched different database drivers (PHP, Ado.net Java) and still be able to abstract data from and to the WCF and Legacy systems. In this experiment there were not combinational effects.

4.3.3 Experiment 3: Query performance comparison tests

To measure the performance of the prototype we defined our second measurement goal according to the GQM method as follows;

Goal 2: Purpose: Comparison.

Object: Different types of persistency techniques (EF and Nhibernate).

Issue: Performance.

Viewpoints: The software development and Maintenance team.

Academic records model, from the prototype was used to execute these tests. It was generated from the legacy applications databases using the Entity framework.

Measuring criteria

Question 1.2: How good is the responsiveness of the system from the client's view?

M1.2.1: Response time in *milliseconds* of the methods of the business logic facades.

M1.2.2: *Frequency distribution* and *standard deviation* of the response time of each method.

The interpretation of the results of this metric will be based on the assumption that a short initialization time is better than a long one. However it is hard to say whether a vast or a narrow distribution where the values are rather small respectively large is more convenient. Therefore the responsiveness of each of these methods will reflect the performance of the user interaction.

Experiment Conducted

The experiment is designed to measure the performance of EF using query methods. The results show that with proper optimization EF can provide better performance. The experiments involve querying small data and large data. The query time will be recorded, in order to compare the effectiveness of performance on EF component model. The data was generating randomly. From the three legacy systems, we randomly picked the legacy application with C-sharp and ado.net drivers. Table 3 shows the records of small data and Table 4 shows the records of large data. The comparison performance is between EF component model and Hibernate. Hibernate is an Object/Relational Mapping technique which manages database tables with persistence objects automatically. The experiment is designed to evaluate the query time for each mechanism. Both statements will query from the same data with the same source. In table 3, the experiment queried data from 100 records to 1000 records. The experiments were iterated to give average results. The table also shows the average queried record for both EF and Hibernate and also the average time difference between the two. A negative difference shows that EF took less time in execution as compared to Hibernate, which we interpreted the negative difference as being faster in execution. For instance for 100 records table 3 shows an average of 4.975 seconds for EF compared 4.96 of Hibernate. The difference in time is positive and we interpreted the results as EF being slower in execution under 100 records.

Table 3: The execution time value of EF model and Hibernate with small query data records

Queried records set	Average queried time of EF (sec)	Average queried time of Hibernate (sec)	The average time difference between EF and Hibernate
100	4.975	4.96	0.015
200	8.02	7.12	0.9
300	12	11	1.5
400	16.17	16.21	-0.04
500	20.35	20.75	-0.40
600	24.02	25.33	-1.31
700	30.1	30.98	-0.97
800	32.96	34.06	-2
900	37.67	38.15	-0.48
1000	40.68	40.81	-0.13

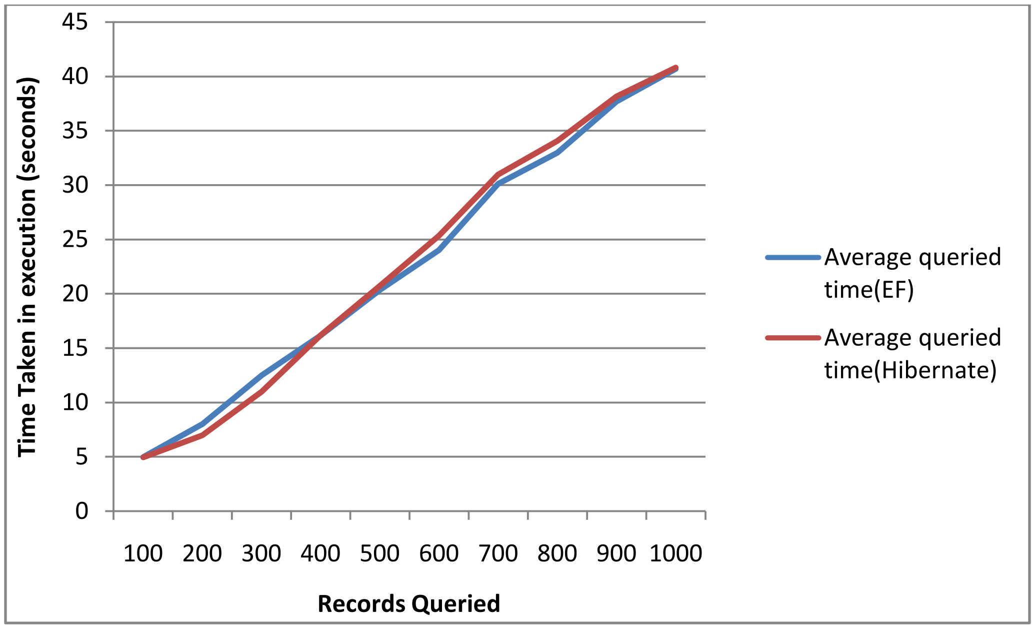


Figure 19: The percentage of time difference between EF component and NHibernate with small queried data of records in the database

Fig 19 illustrates the number of records against time. We started loading data in the prototype using EF component and load the prototype using NHibernate. The experiment starts with 100 records up until 1000 from dummy data from legacy database. There is a decreasing trend of the average of time difference. When the data is queried for 100 records the EF start slowly as compared to Nhibernate. As the records increases for instance between 500 -1000 record EF works faster in terms execution speed.

The second set of experiment was on querying of large data records. In table 4 below, the experiment queried data from 1000 records to 5000 records. The experiments were iterated to give average results. The table also shows the average queried record for both EF and Hibernate and also the difference between the two. A negative difference shows that EF took less time in execution as compared to Hibernate, which we interpreted as being faster in execution.

Table 4: The execution time value of EF model and Hibernate with large query data of records in the database

Queried records set	Average queried time of EF (mins)	Average queried time of Hibernate (mins)	The average time difference between EF and Hibernate
1000	40.68	40.81	-0.13
1500	79.72	87.09	-7.37
2000	112.81	128.09	-15.28
2500	162.27	180.89	-18.72
3000	218.11	219.478	-1.368
3500	260	278.12	-18.12
4000	300	290	10
4500	340.12	330	10.12
5000	360.2	381.2	-21.0

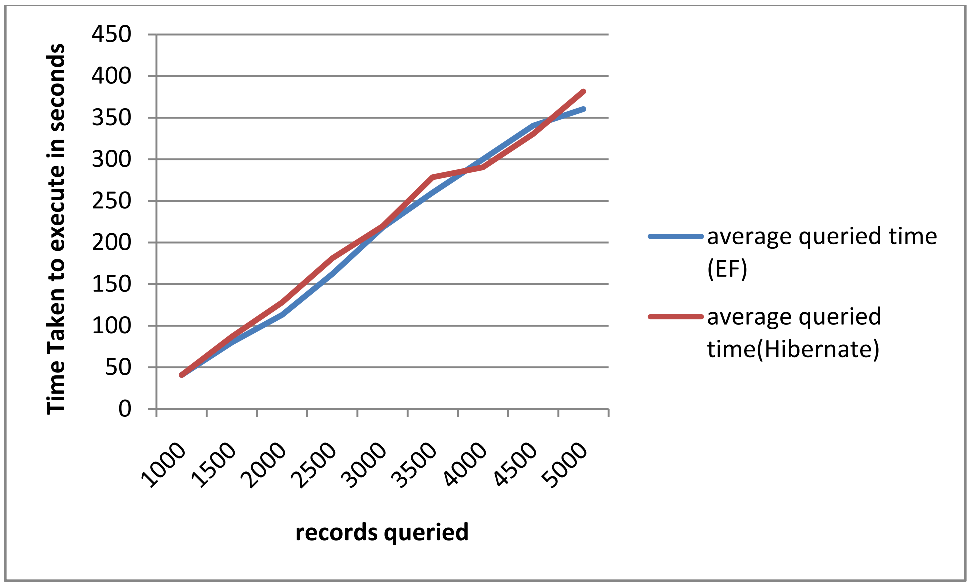


Figure 20: The percentage of time difference between EF and hibernate with large queried of records in the database

Figure 20 show that both approaches are optimal in the environment of 1000-5000 queried record sets in the database, which was a good sign for EF component model. The values of average time difference are quite stable when querying large data. This shows the advantage of the support EF has from the Microsoft. Previous literature in our literature review [14] experiments showed that EF is only good in small records and now has improved on large data.

DISCUSSION OF THE QUERRY RESULTS

According to our results from figure 19 to figure 20, EF gives slow performance in time execution as at the beginning of querying records. This is so because EF loads the metadata (data about data) into the memory first, takes a time. It builds in memory representation of the model edmx file or source code and on the other hand, NHibernate allows greater specification of loading strategies. You can eager load using a join in an initial query or a subsequent select, and even the specify an optimal batch size on a select. None of these features are available with Entity Framework. However, Entity Framework, produced and supported by Microsoft has the advantage over NHibernate in terms of developers being able to find answers to questions. Microsoft has announced the new version of Entity framework to have those areas rectified [10].

This experiment was evaluated with NS theory and found that it violents the fourth theorem, ***separation of states*** which requires that actions or step in a workflow are separated from each other in time keeping state after every action or state. EF component fails to keep up with the speed when querying for the first time. The number of data to be recorded depends on the size of the system and thus implies a combinational effect.

4.3.4 Experiment 4 Scalability

Scalability refers to the system ability to react to significant increases in work load without the degradation of performance. [21]

To measure the scalability we defined our goal according to the GQM method as follows

Goal 3: Purpose: Comparison.

Object: Different types of persistency techniques (EF and Nhibernate).

Issue: Scalability.

Viewpoints: The software development and Maintenance team.

Measuring criteria

Question 2.4: How big is the impact of an increasing load on the prototype?

M2.4.1: To find out the scalability behavior of the prototype the measurements of questions 1.2 will be performed using an increasing amount of request doing different operations.

Experiment Conducted

For each platform a test was ran involving 1 to 20 concurrent requests and measuring the total time it took for EF component model to send the request.

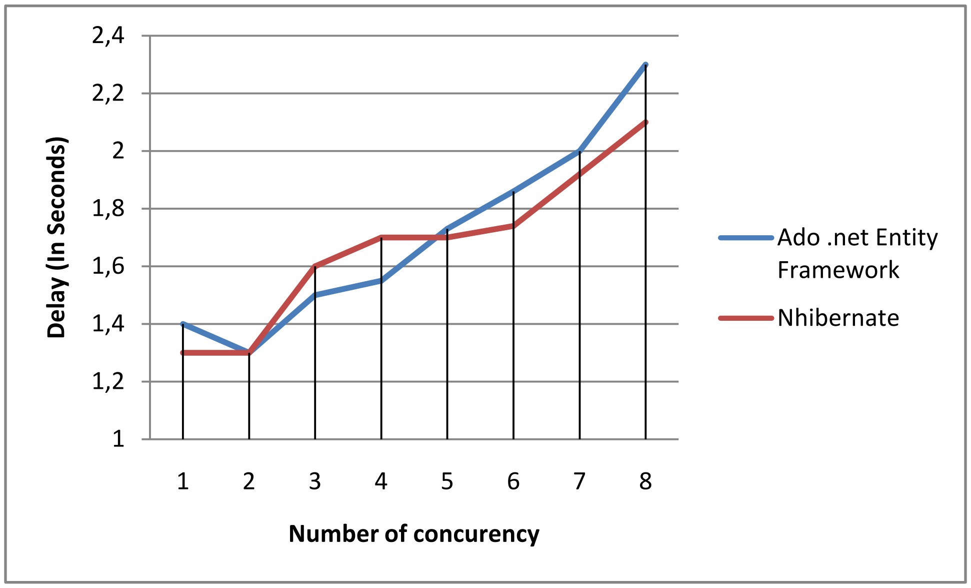


Figure 21: Scalability Graph

Discussion of the results

The figure 21 shows how the rate of change in execution time when the number of concurrency request increases. From figure 21, using the application when requests are below 3, the executing time is almost similar. However, when the number of request exceeds 5 the response time rate the for EF Component model delays while the NHibernate drops normal. We realized that EF Component is more viable solution for mostly smaller number of request, but wouldn't scale easily for more requests. We deduced that this might be probable due to the abstraction layers (between storage models etc) and materialization.

This experiment violent the fourth theorem, ***separation of states*** which requires that actions or step in a workflow are separated from each other in time keeping state after every action or state. EF component fails to keep up with the steady performance when the number of request increases and thus implies a combinational effect.

4.4 DISSUSSIONS AND SUMMARY OF THE EXPERIMENTS

It must be noted that each of these theorem is not completely new, rather using these theorem that identify these combinatorial effects aids to build information systems that contain a minimal number of combinational effects. A remarkable aspect of these theorems is that a violation of each of one of these theorems, by any developer at any moment during development or maintenance, results in a combinatorial effect.

We conducted the evaluations of the experiment and came across combinatorial effects. This allowed us to compare designs of the experiments to see which design cause combinatorial effects and how they manifest themselves. After this analysis we formulated general applicable design guidelines that prevent combinatorial effects. This iterative method of design allows us to adjust and/or refine guidelines as we gain more insight into the problem. Subsequently, we relate the guidelines to our findings from the experiments.

We derive the following key findings:

- Legacy application's data access drivers/ connectors.
- Mapping.
- Small to medium enterprise.
- WCF Data Service.
- Reverse engineering.

We shall discuss our findings below;

Legacy Application's Data Access Drivers/ Connectors

For Legacy applications wanting a generalized abstraction layer with support for multiple data sources, use **ODBC database drivers and ADO database drivers**. This is the most efficient, full-featured API in which Microsoft will continue to invest. For legacy applications built on **PHP driver**, **JDBC driver** and **OLEDB,** should start looking at EF Component model 4.5 going upwards as Microsoft has incorporated additional API for abstraction.

Having evaluated our prototype using the criteria of different connectors, we deduced that, the legacy systems that connect the database with database connectors such as **ODBC database drivers and ADO database drivers, have no problem connecting with the EF Component model of any version. However, some drivers such as** PHP **driver**, **JDBC driver** and **OLEDB** gave problems with connecting with EF Component model until we had to use a much more latest version of EF Component model 4.5.

Mapping

New applications considering one of Microsoft's Object/Relational Mapping technologies should start by looking at the **EF Component model in .NET 4** (including LINQ to Entities) for data access. The Entity Framework was introduced with .NET 3.5 SP1; many improvements in .NET 4 and most new investments in object/relational mapping has happen in the Entity Framework going forward. This guideline we found that it can reduce the violation of theorem ***separation of states*** as now the Microsoft Corporation has claimed that the latest version of EF Component addressed the issue of speed.

WCF Data Service

Use **WCF Data Services** for services that primarily expose data with few (if any) service operations; that is, you're primarily exposing a data model. The data model can come from several sources: an Entity Data Model from the Entity Framework (easiest), through a Reflection Provider over CLR objects, through Data Service Provider interfaces, or through a customer implementation of the OData specification

EF Component is good for small to medium enterprises

Our evaluation pointed that EF Component model is not scalable but however it is fastest when dealing with small number of users at a given time. This means if an application has less number of users at a given time that the entity framework is going to be the best option to use. This guideline answers the violation of the four theorems of NST. Instead of completely ignoring EF component model we can make use of it in small to medium enterprise

Reverse Engineering

In a legacy application where sometimes documentation no longer exists, it becomes necessary to effectively reverse-engineer and extract important concept from its relational model. From the literature we deduced that EF component model should be used for reverse engineering the legacy database and migrate the data to the WCF service for consuming and providing data and we have managed to prove it in our experiments. The EF Component model will deal with issues of extracting inheritance structures from relational model, extracting relationships and extracting association of different cardinalities.

4.5. SUMMARY

This chapter provided an insight on the methodology of this research which is the design science research and also the evaluation method used which was based on the NS Theory. The bases of this chapter was centered mainly on different research activities (i.e., build, evaluate, theorize and justify). The chapter concluded with a discussion on the general guidelines of EF that may be considered for further research on EF.

CHAPTER FIVE: CONCLUSIONS AND RECOMMENDATIONS

5.0 INTRODUCTION

In this research we have provided a first start to solve the problem of integration by the use of design science and NS theory. We have used the EAI approach to incorporate the EF to solve the gap that exists between legacy systems and the middleware when integrating applications. We first design the prototype guided by the claims in the literature review and vendors of the products (Microsofts). The prototype was evaluated using the NS theory with respect to evolvability. We contributed to NS Theory by showing its relevance to the design of an integrated system. After the evaluation and iterating the prototype with many changes, we then proposed our five findings to prevent combinatorial effects so that we can have an integrated platform that promotes heterogeneity applications.

The adoption of these key findings depends on the availability of appropriate tools and expertise and would not only enable designers to integrate legacy systems but also address practical issues in implementing legacy integration.

5.1 CONCLUSION OF THE STUDY

The research presented the EF Component model as technological tool to be used to aid the legacy integration in a SOA environment. The EF Component model allows applications and data centric service to operate at a higher level of abstraction than relational tables via Entity Data Model and rich support between the conceptual schemas and data schemas.

We need to conclude that although we have provided our findings to the design of the prototype, they are not sufficient to prevent all combinatorial effects. The findings we have provided are limited to the non functional requirement we have identified because of the limited scope.

5.2 KEY FINDINGS

This research has given us an initial knowledge on how to use EF Component model in aiding legacy integration in SOA environment. Most importantly we have managed to come up with five guidelines which provide guidance in using EF Component model. These key findings emphasize the importance of abstracting data from legacy system through a method of reverse-engineering. These include the following;

- Legacy application's data access drivers/ connectors- we deduced that database drivers from PHP, JDBC and OLEDB needs to use EF component model of latest version 4.5 and beyond.
- Mapping-EF component model 4.5 and beyond is the best ORM tool for mapping as Microsoft has made numerous changes to accommodate more legacy systems of different programming language and relational models.
- Small to medium enterprise- in terms of scalability the EF component model best works when dealing with less number of request at a time. However with EF component being support we hope in the future the issue of scalability will be rectified.
- WCF Data Service – WCF is the best Service Oriented Architecture that share data easily using service.
- Reverse engineering- EF component is best for reverse engineering. Reverse engineering the legacy database and migrate the data to the WCF service for consuming and providing data and we have managed to prove it in our experiments. The EF Component model will deal with issues of extracting inheritance structures from relational model, extracting relationships and extracting association of different cardinalities.

5.3 FUTURE RESEARCH

Despite the above results and contribution by this dissertation project, the research findings have inherent some limitations, which suggest some sort of care and attention in interpreting and applying or using the research findings.

The prototype couldn’t be developed more and elaborated in the discussion because the study is limited to the available information and the time limitation of this dissertation project.

However, having concluded this research a few recommendations are made for further research.

- More prototypes should be designed with many legacy systems as our research only used only three software platforms. This is to confirm or disapprove our guidelines we have proposed.
- We can engage Microsoft vendors to contribute in making EF Component model a best technology to use especially where the study has showed some short comings

REFERENCES

1. Bisbal, Jesús, et al. "Legacy information systems: Issues and directions." *IEEE software* 16.5 (1999): 103-111.
2. Yajaman, Naveen, and Josh Brown. "Web service facade for legacy applications." *Microsoft Corporation, Jun* (2003).
3. N. Erasala, D. C. Yen, and T. Rajkumar, "Enterprise application integration in the electronic commerce world," Computer Standards & Interfaces, vol. 25, no. 2, pp. 69–82, 2003.
4. Chong, Kwong Chen. *A middleware integrating ERP, CRM and supply chain management system using service oriented architecture*. Diss. University of Malaya, 2011.
5. P. Johannesson and E. Perjons, "Design principles for process modeling in enterprise application integration," Information Systems, vol. 26, no. 3, pp. 165–184, 2001
6. Huysmans, Philip, et al. "Integrating Information Systems Using Normalized Systems Theory: Four Case Studies." *Business Informatics (CBI), 2015 IEEE 17th Conference on*. Vol. 1. IEEE, 2015.
7. Mannaert, Herwig, Jan Verelst, and Kris Ven. "Towards evolvable software architectures based on systems theoretic stability." *Software: Practice and Experience* 42.1 (2012): 89-116.
8. Seth, Ashish, Himanshu Agrawal, and Ashim Raj Singla. "Unified Modeling Language for Describing Business Value Chain Activities." *International Journal of Computer Applications®(IJCA)), USA* (2012).

9. Wigley, Andy, et al. *Microsoft. net compact framework: Core reference*. Microsoft Press, 2002.

10. Mössenböck, Hanspeter, et al. . *NET Application Development: With C#, ASP. NET, ADO. NET, and Web Services*. Pearson Addison Wesley, 2004.

11. Yemelyanov, Andrey. "Using ADO. NET Entity Framework in Domain-Driven Design: A Pattern Approach." (2008).

12. Selamat, Mohd Hasan, and Abdulmonem Al Kharusi. "Service Oriented Architecture in Education Sector." *IJCSNS International Journal of Computer Science and Network Security* 9.5 (2009): 301-305.

13. Zou, Ying, and Kostas Kontogiannis. "Towards a Webcentric Legacy System Migration Framework." *Proceedings of the ICSE 2001 Workshops of 3rd International Workshop on Net-Centric Computing: Migrating to the Web*. 2001.

14. Erl, Thomas. *Soa: principles of service design*. Vol. 1. Upper Saddle River: Prentice Hall, 2008.

15. Zhang, Zhuopeng, and Hongji Yang. "Incubating services in legacy systems for architectural migration." *Software Engineering Conference, 2004. 11th Asia-Pacific*. IEEE, 2004.

16. J. A. Blakeley, S. Muralidhar, A. Nori. The EF Component model: Making the Conceptual Level Real. In ER, 2006.

17. Castro, Pablo, Sergey Melnik, and Atul Adya. "ADO. NET entity framework: raising the level of abstraction in data programming." *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007.

18. Köpcke, Hanna, and Erhard Rahm. "Frameworks for entity matching: A comparison." *Data & Knowledge Engineering* 69.2 (2010): 197-210.

19. Mannaert, Herwig, Jan Verelst, and Kris Ven. "Exploring the concept of systems theoretic stability as a starting point for a unified theory on software engineering." *Software Engineering Advances, 2008. ICSEA'08. The Third International Conference on*. IEEE, 2008.

20. Khoumbati, Khalil, Marinos Themistocleous, and Zahir Irani. "Evaluating the adoption of enterprise application integration in health-care organizations." *Journal of Management Information Systems* 22.4 (2006): 69-108.

21. Umapathy, Karthikeyan, Sandeep Purao, and Russell R. Barton. "Designing enterprise integration solutions: effectively." *European Journal of Information Systems* 17.5 (2008): 518-527.

22. Chen, H., Yin, J., Jin, L., Li, Y., & Dong, J. (2007). *JTang Synergy: A Service Oriented Architecture for Enterprise Application Integration.* Paper presented at the Computer Supported Cooperative Work in Design, 2007. CSCWD 2007. 11th International Conference, Melbourne, Victoria.

23. Hart, J. (2003). Web Sphere: The leading software platform for on demand business.

24. Jung, D.-G., Paek, K.-J., & Kim, T.-Y. (1999). *Design of MOBILE MOM: Message oriented middleware service for mobile computing.* Paper presented at the Parallel Processing, 1999. Proceedings. 1999 International Workshops, Aizu- Wakamatsu.

25. Apshankar, K., Clark, M., Hanson, J. J., Mittal, K., & Myerson, J. M. (2002). *WebServices Business Strategies and Architectures*. Birmingham, UK: Expert Press.

26. Peffers, Ken, et al. "A design science research methodology for information systems research." *Journal of management information systems* 24.3 (2007): 45-77.
27. Newell, Allen, and Herbert A. Simon. "Computer science as empirical inquiry: Symbols and search." *Communications of the ACM* 19.3 (1976): 113-126.
28. Hevner, Alan, and Samir Chatterjee. *Design science research in information systems*. Springer US, 2010.
29. Simon, Herbert A. *The sciences of the artificial*. MIT press, 1996.
30. Walls, Joseph G., George R. Widmeyer, and Omar A. El Sawy. "Building an information system design theory for vigilant EIS." *Information systems research* 3.1 (1992): 36-59.
31. Markus, M. Lynne, Ann Majchrzak, and Les Gasser. "A design theory for systems that support emergent knowledge processes." *MIS quarterly* (2002): 179-212.
32. Raje, R. R., M. Auguston, B. R. Bryant, A. Olson and C. Burt, "A Unified Approach for the Integration of Distributed Heterogeneous Software Components," Proceedings of the Monterey Workshop on Engineering Automation for Software Intensive System Integration, pp. 109-119, 2001
33. Gupta, Natasha S., Stanley Y. Chien, and Rajeev R. Raje. *An Exploratory Analysis of the. Net Component Model and Uniframe Paradigm Using a Collaborative Approach*. INDIANA UNIV-PURDUE UNIV AT INDIANAPOLIS DEPT OF COMPUTER AND INFORMATION SCIENCES, 2004.
34. Goldschmidt, Thomas, Jochen Winzen, and Ralf Reussner. "Evaluation of maintainability of model-driven persistency techniques." *IEEE CSMR 07-Workshop on Model-Driven Software Evolution (MoDSE2007)*. 2007.

35. Goldschmidt, Thomas, Jochen Winzen, and Ralf Reussner. "Evaluation of maintainability of model-driven persistency techniques." *IEEE CSMR 07-Workshop on Model-Driven Software Evolution (MoDSE2007)*. 2007.

36. Caldiera, V. R. B. G., and H. Dieter Rombach. "The goal question metric approach." *Encyclopedia of software engineering* 2.1994 (1994): 528-532.

37. Huysmans, Philip, and Jan Verelst. "Improving Enterprise Architecture Evaluation Based on Concepts from the Normalized Systems Theory." *International Journal of IT/Business Alignment and Governance (IJITBAG)* 3.2 (2012): 38-50

38. Tim McLaren, M. B. A., and Paul Buijs. "A design science approach for developing information systems research instruments." (2011).

39. Benatallah, Boualem, et al. "Developing adapters for web services integration." *Advanced Information Systems Engineering*. Springer Berlin Heidelberg, 2005.

40. Mitleton-Kelly, Eve, and Frank Land. *Complexity & information systems*. Blackwell Publishing Limited, 2012.

41. Ran, Chong-shan, and Ning Li. "Design and implementation of communication model based on Silverlight and WCF in EAM system." *Information and Financial Engineering (ICIFE), 2010 2nd IEEE International Conference on*. IEEE, 2010.

42. Ismaeel, Ayad Ghany, and Sanaa Enwaya Rizqo. "Optimal Productivity of Succoring Patients System using Mobile GIS Based on WCF Technology." *arXiv preprint arXiv:1305.0673* (2013).

43. Zhang, Wei, and Guixue Cheng. "A service-oriented distributed framework-WCF." *2009 International Conference on Web Information Systems and Mining*. 2009.

44. Bandgar, Shrimant B., T. M. Bansod, and B. B. Meshram. "Design Distributed Application System on SOA by Using WCF and WPF." *International Journal of Managment, IT and Engineering* 2.8 (2012): 103-114.

45. Srinivasan, Latha, and Jem Treadwell. "An overview of service-oriented architecture, web services and grid computing." *HP Software Global Business Unit* 2 (2005).

46. W. R. Cook, A. H. Ibrahim. Integrating Programming Languages and Databases: What is the Problem? ODBMS.ORG, Expert Article, Sept. 2006

47. Sneed, Harry M. "Encapsulating legacy software for use in client/server systems." *Reverse Engineering, 1996., Proceedings of the Third Working Conference on*. IEEE, 1996.

48. Yellin, Daniel M., and Robert E. Strom. "Protocol specifications and component adaptors." *ACM Transactions on Programming Languages and Systems (TOPLAS)* 19.2 (1997): 292-333.

49. Aiken, David, et al. "An Adapter Framework for a Web Service Execution Engine." *European Semantic Web Conference ESWC*. 2004.

50. Jevtović, Branislav, et al. "Cloud Computing and Virtualization in Embadded Devices Space Using WCF." *Acta facultatis medicae Naissensis* 27.4.

51. Early Look, “Introducing Windows Communi-cation Foundation”, David Chappell, Chappell & Associates, September 2005,

52. Kester, Quist-Aphetsi, and Ajibade Ibrahim Kayode. "Using SOA with Web Services for effective data integration of Enterprise Pharmaceutical Information Systems." *arXiv preprint arXiv:1307.8179* (2013).

APPENDIX A

SOFTWARE PLATFORMS DOCUMENT ANALYSIS

This section provides a summary of each software framework documentation analyzed when conducting this study. For each framework a short description is included along with a more comprehensive discussion

JDBC

Java Database Connectivity (**JDBC**) is an application programming interface (API) for the programming language Java, that defines how a client may access a database. It is part of the Java Standard Edition platform, from Oracle Corporation.

JDBC technology allows you to use the Java programming language to exploit "Write Once, Run Anywhere" capabilities for applications that require access to enterprise data. With a JDBC technology-enabled driver, you can connect all corporate data even in a heterogeneous environment.

ADO.NET

ADO.**NET** is a set of computer software components that programmers can use to access data and data services from the database. It is a part of the base class library that is included with the Microsoft .**NET** Framework.

It is an integral part of the .NET Framework, providing access to relational, XML, and application data. ADO.NET supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects used by applications, tools, languages, or Internet browsers.

MSQL

MySQL is an open-source relational database management system (RDBMS); in July 2013, it was the world's second most [a] widely used RDBMS, and the most widely used open-source client–server model RDBMS.

NHIBERNATE

Hibernate ORM (Hibernate in short) is an object-relational mapping framework for the Java language. It provides a framework for mapping an object-oriented domain model to a relational database. Hibernate solves object-relational impedance mismatch problems by replacing direct, persistent database accesses with high-level object handling functions

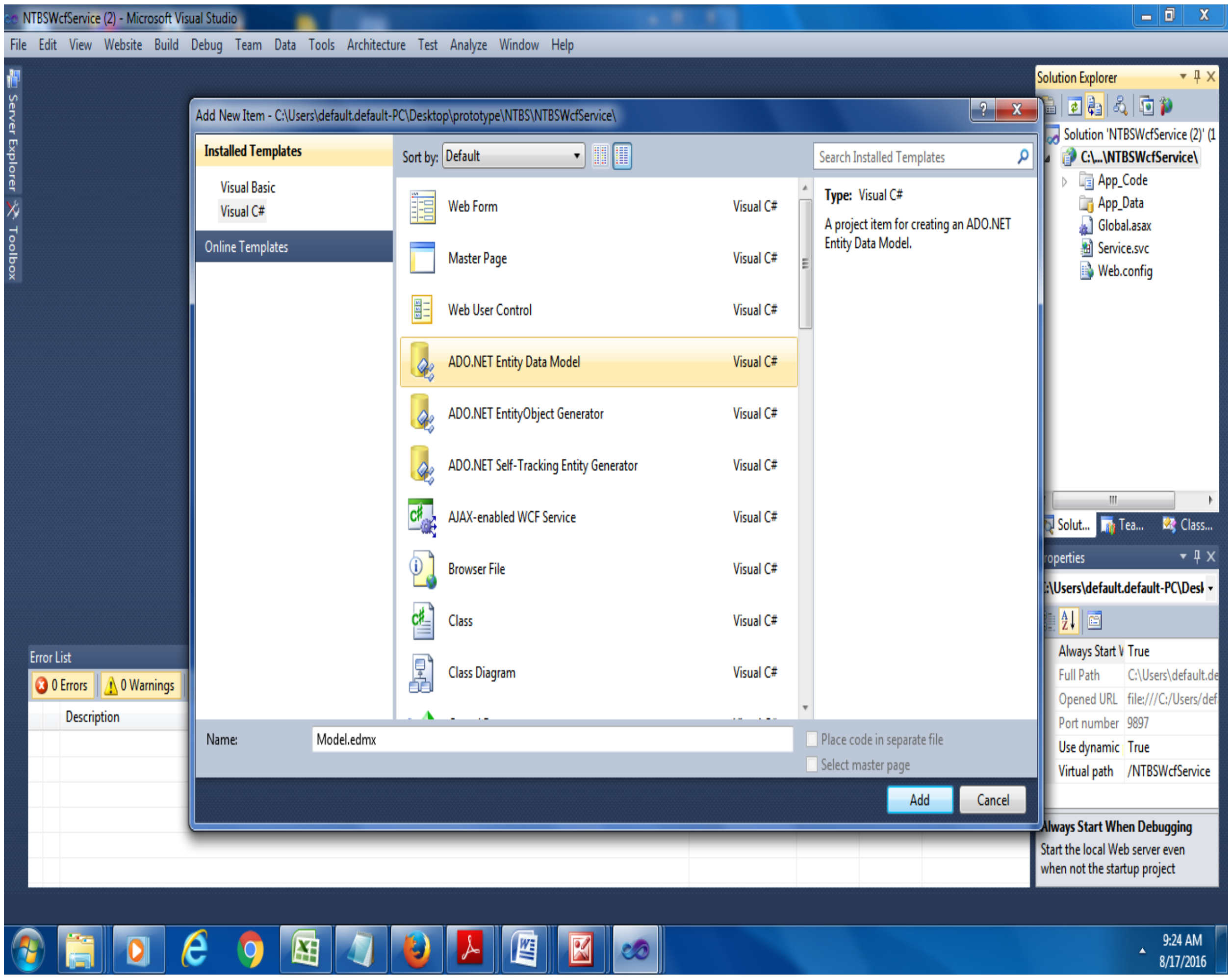
Hibernate's primary feature is mapping from Java classes to database tables; and mapping from Java data types to SQL data types. Hibernate also provides data query and retrieval facilities. It generates SQL calls and relieves the developer from manual handling and object conversion of the result set.

APPENDIX B (SNAPSHOTS OF PROTOTYPE)

These are the screenshots of one of the legacy systems we used for the experiments in this research to demonstrate how EF component model can be used in aiding legacy integration in a service oriented environment.

<u>Modeling the legacy Database</u>

Using the legacy system database named AdventureWorks, we modelled this database using EF component model.


NTBSWcfService (2) - Microsoft Visual Studio
File Edit View Website Build Debug Team Data Tools Architecture Test Analyze Window Help
Server Explorer
Toolbox
Add New Item - C:\Users\default.default-PC\Desktop\prototype\NTBS\NTBSWcfService\
Installed Templates
Visual Basic
Visual C#
Online Templates
Sort by: Default
Search Installed Templates
Web Form Visual C#
Master Page Visual C#
Web User Control Visual C#
ADO.NET Entity Data Model Visual C#
ADO.NET EntityObject Generator Visual C#
ADO.NET Self-Tracking Entity Generator Visual C#
AJAX-enabled WCF Service Visual C#
Browser File Visual C#
Class Visual C#
Class Diagram Visual C#
Place code in separate file
Select master page
Add
Cancel
Error List
0 Errors
0 Warnings
Description
Solution Explorer
Solution 'NTBSWcfService (2)' (1
C:\...\NTBSWcfService\
App_Code
App_Data
Global.asax
Service.svc
Web.config
Solut...
Tea...
Class...
roperties
:\Users\default.default-PC\Desl
9:24 AM
8/17/2016

Type: Visual C#

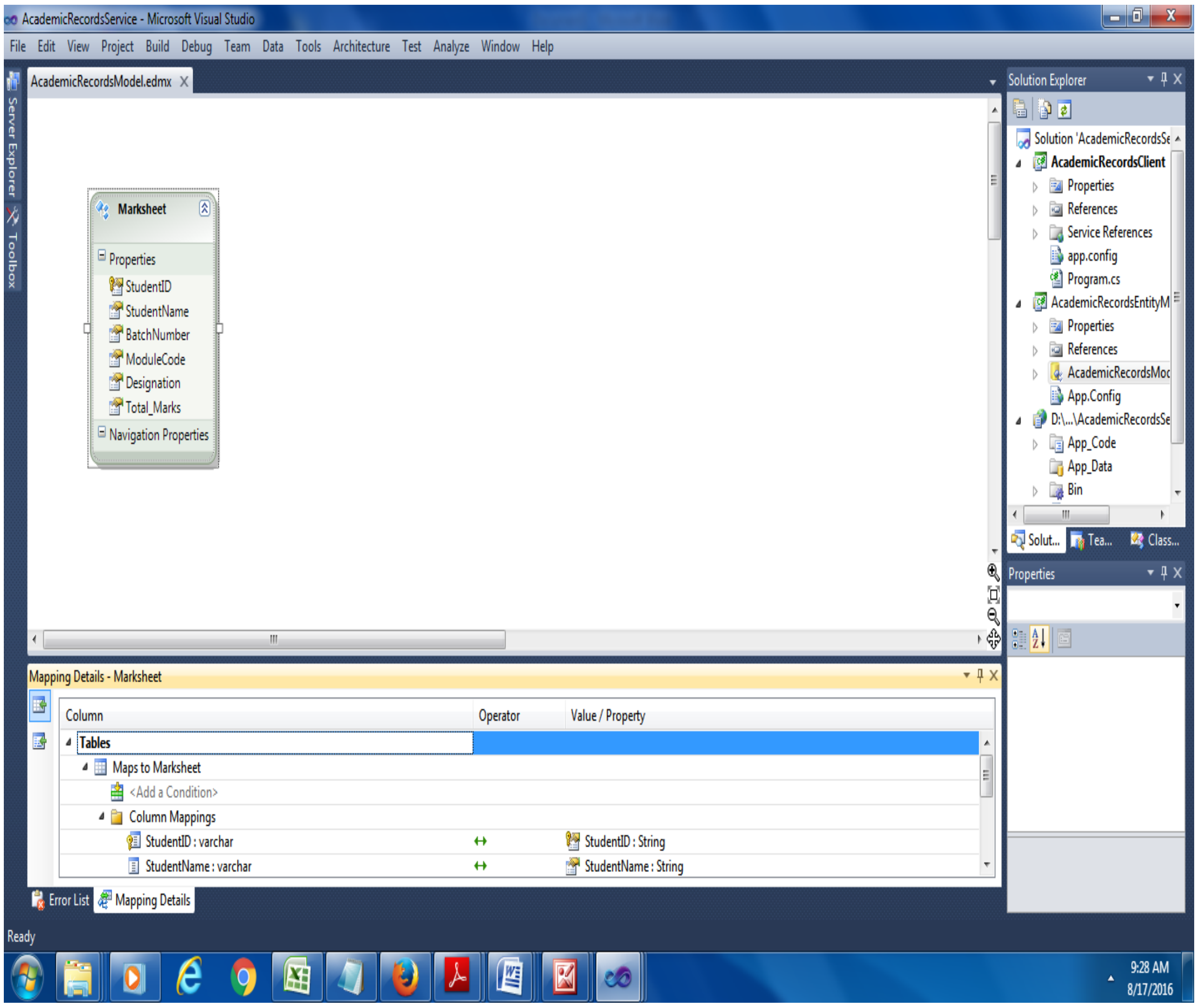
A project item for creating an ADO.NET Entity Data Model.

Name: Model.edmx

Always Start V	True
Full Path	C:\Users\default.de
Opened URL	file:///C:/Users/def
Port number	9897
Use dynamic	True
Virtual path	/NTBSWcfService

Always Start When Debugging
Start the local Web server even when not the startup project

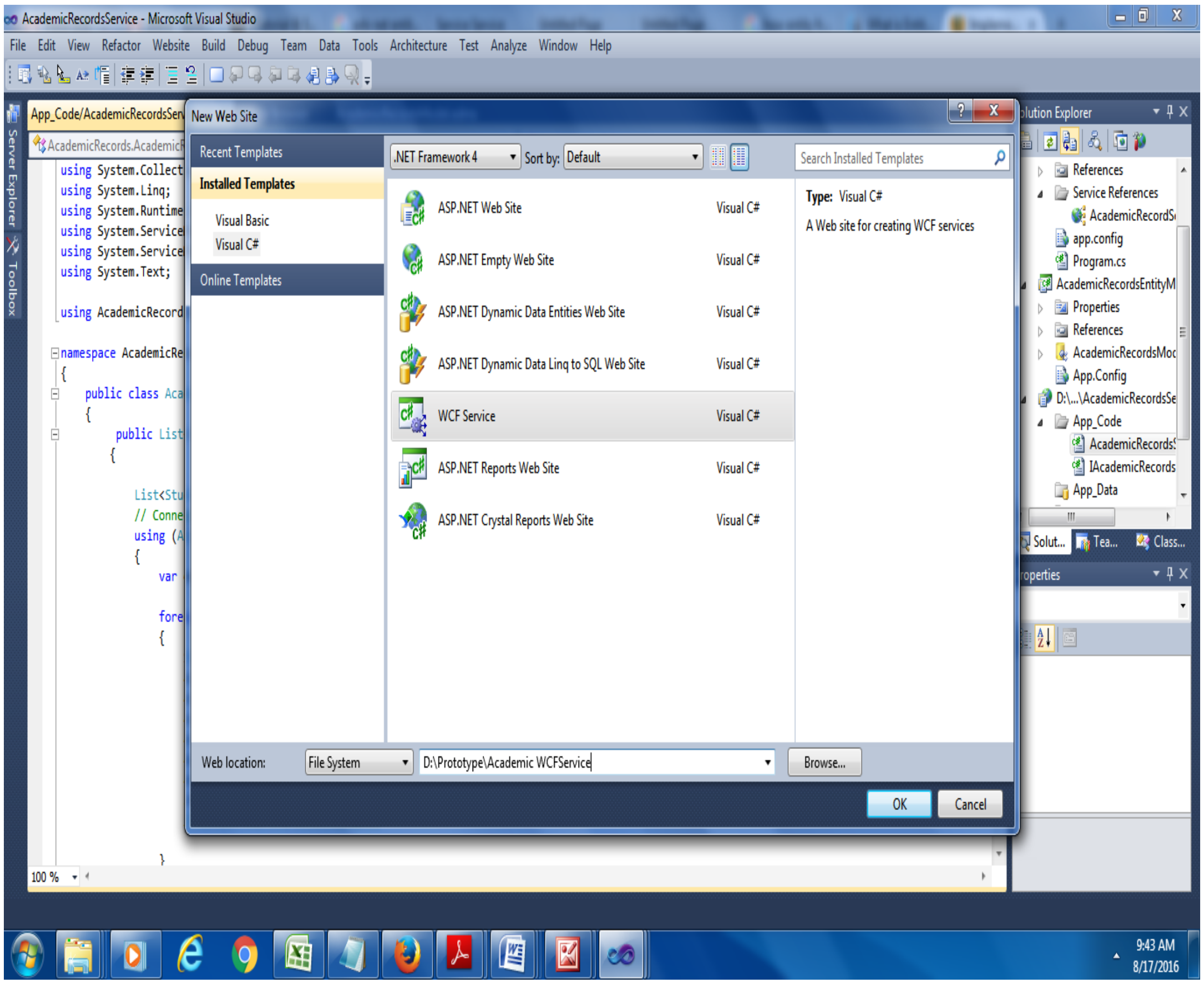
After modeling the academic records database we used **LINQ to entities** to retrieve the real student details from Marksheet information from the database.



Column	Operator	Value / Property
Tables		
Maps to Marksheet		
<Add a Condition>		
Column Mappings		
StudentID : varchar	↔	StudentID : String
StudentName : varchar	↔	StudentName : String

Creating the WCF service

This WCF service contains the ***GetAllStudent*** operation, which returns students details to the client. The student details are hard code in this WCF service. Below is the screen shoot.


AcademicRecordsService - Microsoft Visual Studio

```
using System.Collect
using System.Linq;
using System.Runtime
using System.Service
using System.Service
using System.Text;

using AcademicRecord

namespace AcademicRe
{
    public class Aca
    {
        public List
        {

            List<Stu
            // Conne
            using (A
            {
                var

                fore
                {
```

New Web Site

Recent Templates
Installed Templates
Visual Basic
Visual C#
Online Templates

.NET Framework 4 — Sort by: Default

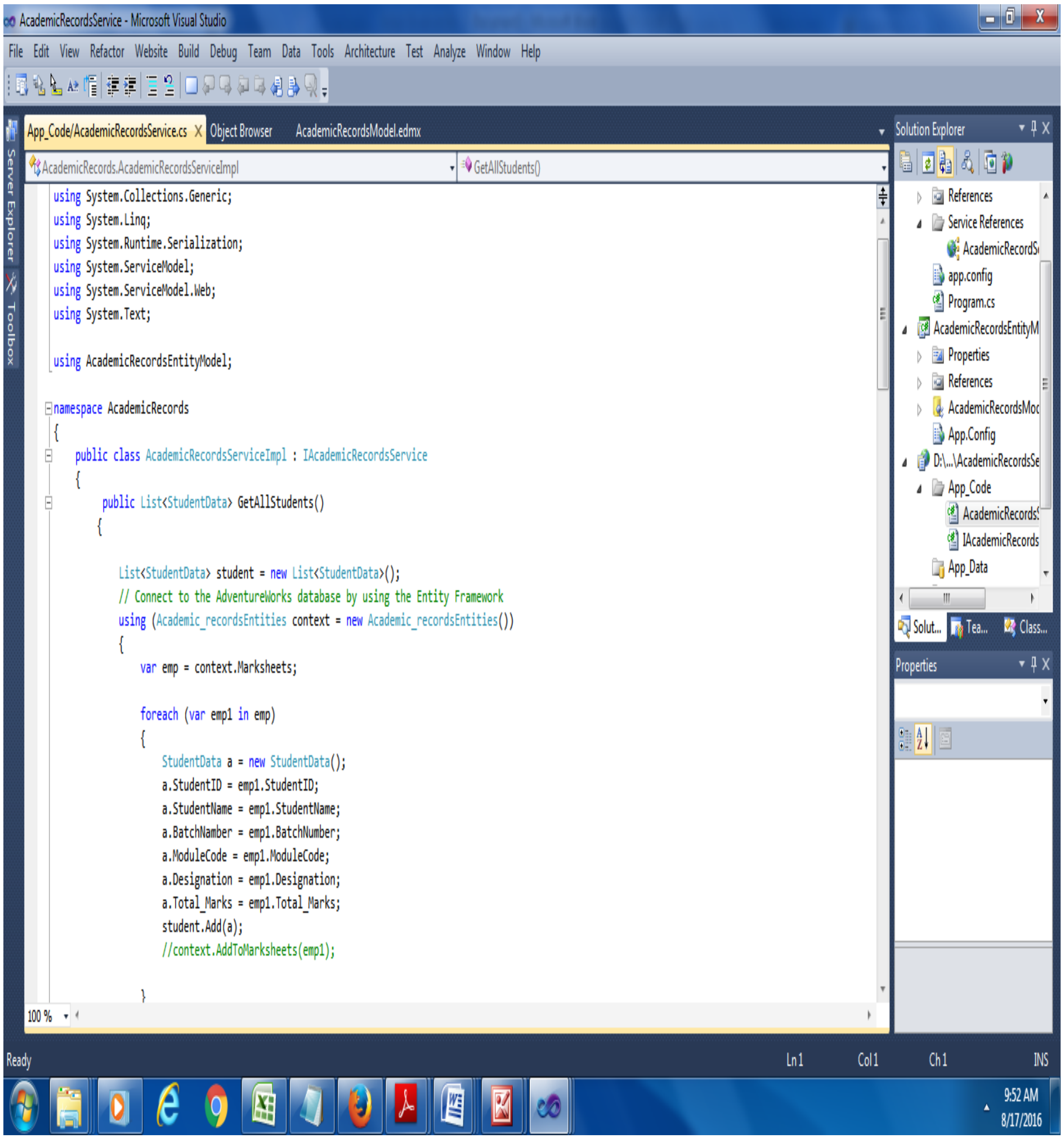
ASP.NET Web Site	Visual C#
ASP.NET Empty Web Site	Visual C#
ASP.NET Dynamic Data Entities Web Site	Visual C#
ASP.NET Dynamic Data Linq to SQL Web Site	Visual C#
WCF Service	Visual C#
ASP.NET Reports Web Site	Visual C#
ASP.NET Crystal Reports Web Site	Visual C#

Type: Visual C#
A Web site for creating WCF services

Web location: File System — D:\Prototype\Academic WCFService

Creating the service interface

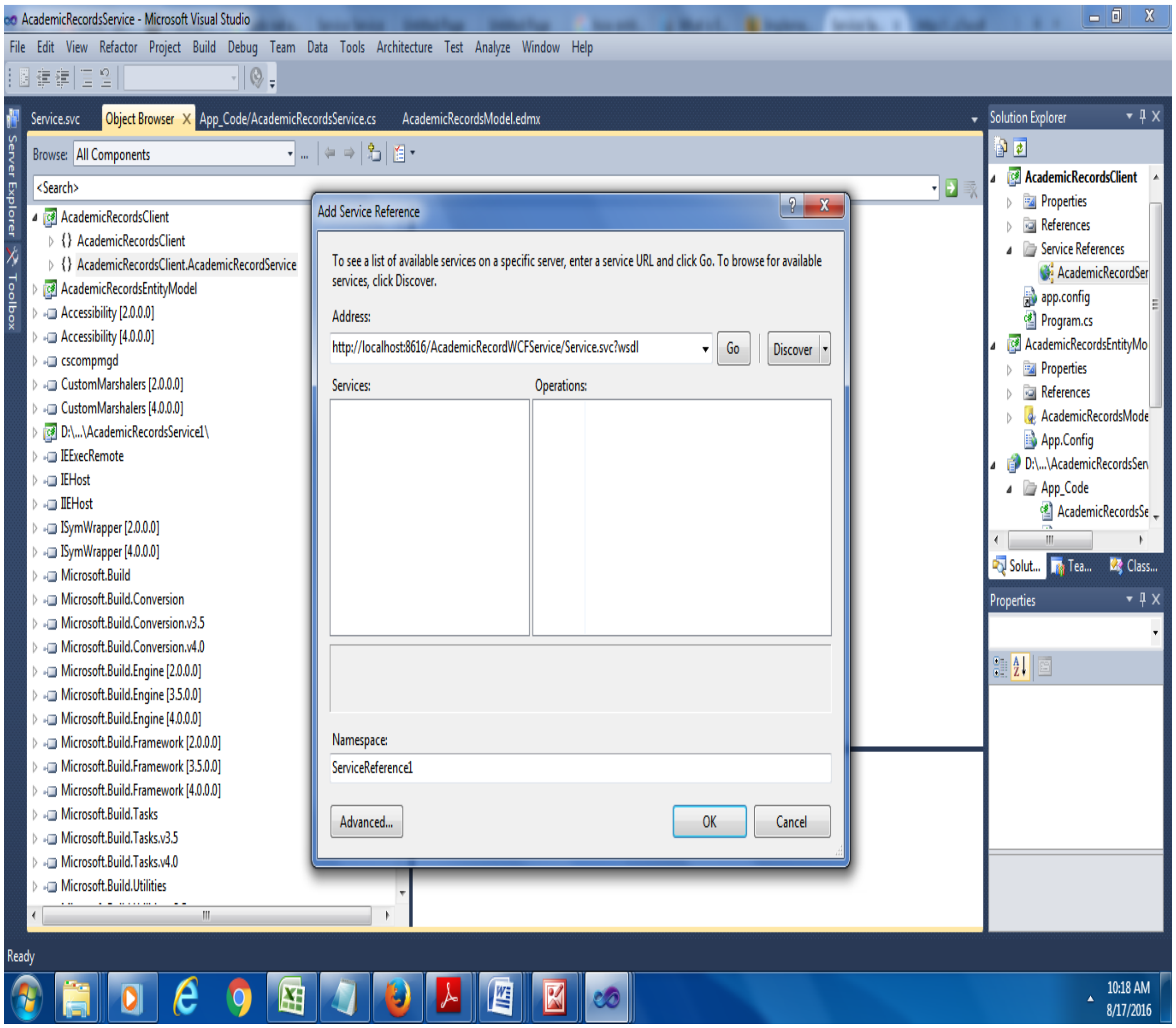
After creating the WCF we created the service interface contracts. ***For more see figure 2, Chapter 2 literature review, and pages 17 to 20.***



```
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

using AcademicRecordsEntityModel;

namespace AcademicRecords
{
    public class AcademicRecordsServiceImpl : IAcademicRecordsService
    {
        public List<StudentData> GetAllStudents()
        {

            List<StudentData> student = new List<StudentData>();
            // Connect to the AdventureWorks database by using the Entity Framework
            using (Academic_recordsEntities context = new Academic_recordsEntities())
            {
                var emp = context.Marksheets;

                foreach (var emp1 in emp)
                {
                    StudentData a = new StudentData();
                    a.StudentID = emp1.StudentID;
                    a.StudentName = emp1.StudentName;
                    a.BatchNamber = emp1.BatchNumber;
                    a.ModuleCode = emp1.ModuleCode;
                    a.Designation = emp1.Designation;
                    a.Total_Marks = emp1.Total_Marks;
                    student.Add(a);
                    //context.AddToMarksheets(emp1);

                }
```

Making the Service available

Making the service available, see the snapshots below


AcademicRecordsService - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Service.svc
Object Browser
App_Code/AcademicRecordsService.cs
AcademicRecordsModel.edmx
Browse: All Components
<Search>
AcademicRecordsClient
AcademicRecordsClient
AcademicRecordsClient.AcademicRecordService
AcademicRecordsEntityModel
Accessibility [2.0.0.0]
Accessibility [4.0.0.0]
cscompmgd
CustomMarshalers [2.0.0.0]
CustomMarshalers [4.0.0.0]
D:\...\AcademicRecordsService1\
IEExecRemote
IEHost
IIEHost
ISymWrapper [2.0.0.0]
ISymWrapper [4.0.0.0]
Microsoft.Build
Microsoft.Build.Conversion
Microsoft.Build.Conversion.v3.5
Microsoft.Build.Conversion.v4.0
Microsoft.Build.Engine [2.0.0.0]
Microsoft.Build.Engine [3.5.0.0]
Microsoft.Build.Engine [4.0.0.0]
Microsoft.Build.Framework [2.0.0.0]
Microsoft.Build.Framework [3.5.0.0]
Microsoft.Build.Framework [4.0.0.0]
Microsoft.Build.Tasks
Microsoft.Build.Tasks.v3.5
Microsoft.Build.Tasks.v4.0
Microsoft.Build.Utilities
Add Service Reference
Services:
Operations:
Advanced...
OK
Cancel
Go
Discover
Solution Explorer
AcademicRecordsClient
Properties
References
Service References
AcademicRecordSer
app.config
Program.cs
AcademicRecordsEntityMo
Properties
References
AcademicRecordsMode
App.Config
D:\...\AcademicRecordsSer
App_Code
AcademicRecordsSe
Solut...
Tea...
Class...
Properties
Server Explorer
Toolbox
Ready
10:18 AM
8/17/2016

To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

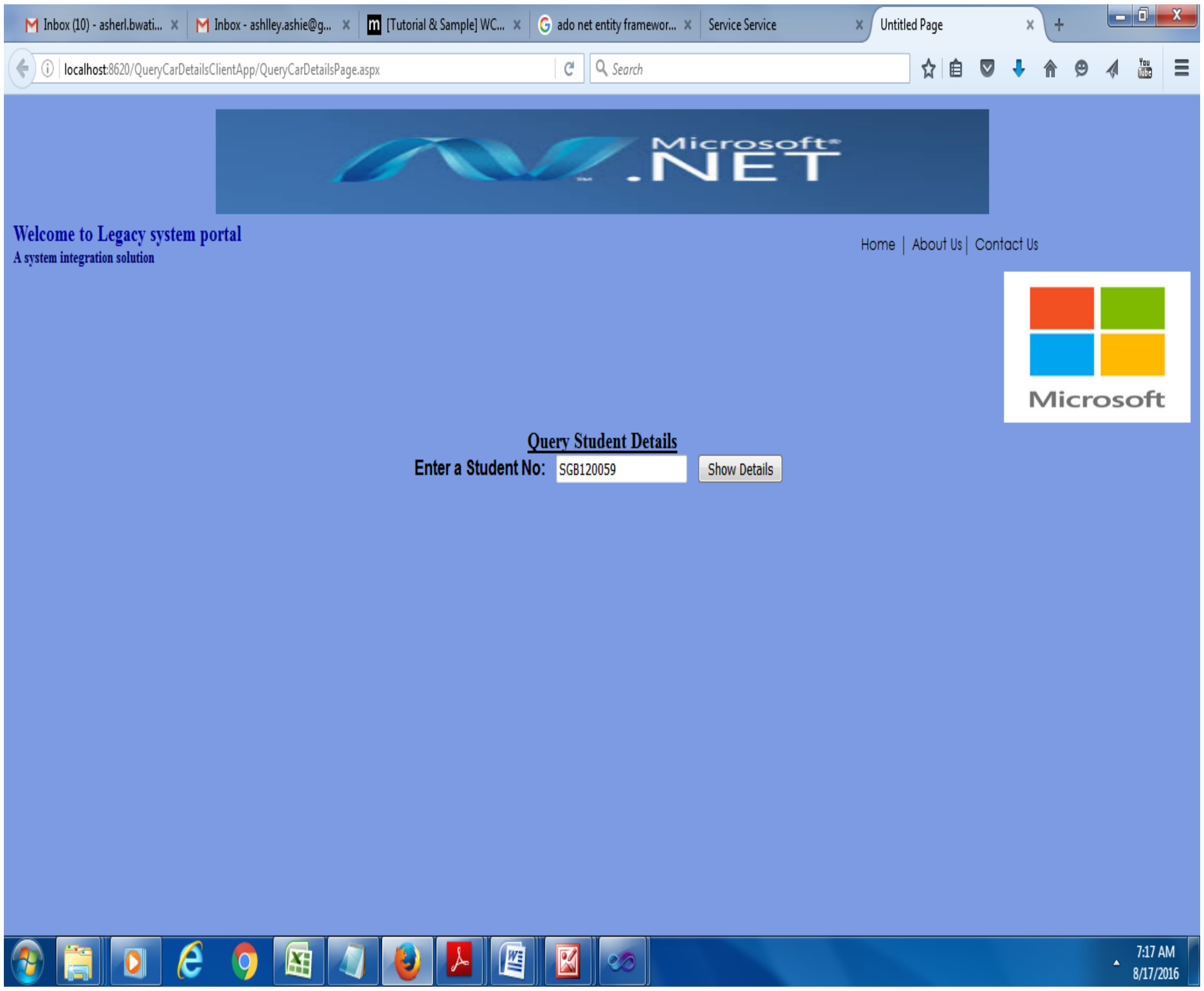
Address:

http://localhost:8616/AcademicRecordWCFService/Service.svc?wsdl

Namespace:

ServiceReference1

Legacy system Portal


Inbox (10) - asherl.bwati...
Inbox - ashlley.ashie@g...
[Tutorial & Sample] WC...
ado net entity framewor...
Service Service
Untitled Page
localhost:8620/QueryCarDetailsClientApp/QueryCarDetailsPage.aspx
Search
Microsoft .NET
Home | About Us | Contact Us
Microsoft
Show Details
7:17 AM
8/17/2016

Welcome to Legacy system portal

A system integration solution

Query Student Details

Enter a Student No: SGB120059