# C Sharp Enterprise Zone

### Introduction

This book covers Enterprise application architecture, design and development using C#. The Habanero framework is referenced throughout the book as a sample of an application framework that implements a solution to the common problems experienced in Enterprise applications. In its broadest sense Enterprise applications create, store and manage information in Enterprises. In simple terms an enterprise application is not an operating system, a compiler an mp3 player, or a game. Enterprise applications are applications used in the running of a business where the information is stored in a structred manner over a long period of time (usually Relational databases). The information is usually shared between multiple users and is frequently used in geographically distributed locations. Examples are Customer Order Management, Procurement, Asset Management, Financial, Planning scheduling, logistical and supply chain management.\

Habanero is a framework for developing Enterprise applications. In essence Habanero solves the problems common to developing and maintaining enterprise applications.

This book is structured so that it can be used by junior developers as well as senior developers. The book assumes you have a reasonable knowledge of object oriented programming and practices, C#, DotNet and database management systems. The book also assumes that you have some knowledge of Enterprise application development. It is impossible for one book to cover in sufficient depth all the topics relating to the development of an enterprise application. We have therefore referenced other books, articles etc for those who are interested or require more detail. Having said this we will cover all the relevant topics in sufficient detail to provide the reader with the knowledge required to develop robust scalable enterprise applications while dodging the common mistakes.

This book covers all aspects of Enterprise application development including User interface development, Domain modelling, business rule implementation, object relational mapping and reporting.

### The Habanero application development framework

Habanero has been developed over eight years and adapted to various technologies and development platforms. The architecture was developed by a company that specialises in Custom application development. Due to the nature of this business the applications had to be robust, easily extendable, easily deployed, secure, scalable from a single user to thousands of users, easily maintainable by any developer in the company or developers in the customer’s company. The framework had to be easily used while retaining extensibility, since developer skills ranged from young graduates to seasoned professionals and project sizes ranged from 2 weeks for a single developer to 3 years with a large team of developers.

As an additional advantage, this framework and access to its accumulated knowledge is available to Dot Net developers as free open source, allowing developers to review, study and contribute to it.

**Firestarter**

Firestarter is an application modelling and code generation tool. To maximise productivity and eliminate repetitive tasks Chillisoft developed Firestarter. This tool allows you to reverse engineer an existing database into an object model as well as to generate your classes, properties, relationships, CRUD user interfaces, business rules and basic unit tests. Firestarter is included for free with Habanero.

### Why this book

As the Habanero development community has continued to grow, it has become clear that there is a need to provide a more extensive guide to the use of Habanero. We have therefore developed a full length book to provide developers with additional information not only on how to use Habanero and Firestarter but also on the common problems and solutions related to the development of Enterprise applications. This book will form the ideal guide book for developers new to Habanero and Enterprise applications or to developers with previous Habanero or Enterprise application experience.

### Layout of the book

This can be used in many ways.

To help you determine how to read this book we have divided readers into two extremes. Most likely you will fit somewhere in between these:

1. A developer who wants to know how to use Habanero framework to develop an application.
2. A developer who wants to understand why and how the framework was developed, how to extend it and or how to use the concepts to develop their enterprise applications.

We understand that the readers of this book and the users of Habanero are busy. We have therefore tried in every way to ensure that the book can be used to provide immediate benefit at each stage. Since software developers learn best by example we have included extensive examples as well as a worked example that will be developed throughout the various sections of the book.

Section - Overview1:

This section provides the generation of a sample application using habanero and introduces all the principles required for developing enterprise applications. We would suggest that all readers read these chapters.

Chapter 1 covers the use of Habanero and Firestarter (an application code generator specialised for Habanero). This is very brief and used merely to give the reader a quick overview of the Framework.

Chapter 2 covers the principles and practices of Enterprise application development. This chapter will touch on all the concepts that will be covered in the remaining chapters of the book.

Section 2 – Business Logic Layer:

This section covers the development of the Business logic layer of an enterprise application, also called the domain model. The domain model involves a group of objects that model the area of the business that your system is implementing. The domain model in the simplest case closely resembles the relational database model but in more complex business scenarios can be significantly different. The domain model provides the layer where you model the domain behavior, data, validation and object relationships. One of the important functions of this layer is to hide the database, so this section will not discuss any database and object persistence issues. Object persistence will be addressed in Section 4.

Each chapter in the section starts with a brief introduction to the topic. This is followed by an example of how to model this using Firestarter and Habanero and how to use this in an application. A developer merely wanting to use Habanero can stop at this point and move onto the next chapter. (TODO ICON for each section)

The chapter will then delve into the more advanced features of Habanero and Enterprise applications including how Habanero implements these features, the design decisions behind the implementation and where relevant a discussion of why these decisions were made when compared to alternates.

Finally the chapter move onto how these features of Habanero can be extended to deal exceptional situations and requirements.

Section 3 - Presentation:

This section covers the development of the Presentation layer. The Presentation layer covers user interfaces for capturing, searching, manipulating, reporting on and viewing objects. The technologies covered here include, ASP, C# and Visual WebGui (an AJAX framework for web development). The reporting tool preferred is Active reports.

Section 4 - Data Access:

This section covers the development of the Data layer i.e. the mapping of business objects to a data store. This section includes persisting objects to XML, memory caches, object data stores and relational databases. Since most enterprise applications persist their data to relational databases, a large part of this section revolves around persisting objects in a relational database (using object relational mapping).

The structure of these chapters is the same as section 2 and should be used in the same manner.

Section 5 - Miscelaneous:

In addition to the presentation of Domain objects in a user interface, our business objects also may need to be available to other applications e.g via webservices. In this section we will show various solutions for these.

The book:

This book has been written by

Eric Savage

Peter Wiles

Brett Powell

Soriya Das

Hagashen Painintheass

Sherwin AnotherPain

Mark Whitfeld

Stuart Wiener

This book is provided free for use to the Habanero community under …….Creative Commons

If you wish to contribute then please post errata suggestions etc to …….

If you find this book useful and wish to make a financial contribution then please contribute to …. A charity for the orphans of AIDS.

# Section 1 – Overview:

## Chapter 1:

In t

Start with Simple Sample Application.

-- Require Firestarter can generate a fully functional UI Layer.

In this chapter we will cover a very simple sample application. This sample application is merely to demonstrate the power and speed of use of Habanero and Firestarter. In Section 2 to 4 a more realistic application will be developed.

Chapter 2:

Enterprise applications:

* Typically involve object persistence (often for very long periods of time).
* Will usually be subject to significant changes over their life cycle as the business adapts current business processes and implements new processes.
* Will usually involve many concurrent users.
* Have many user interfaces for capturing, viewing and organizing data.
* Produce and maintain data that is used extensively for business analysis and reporting.
* Often integrate with other enterprise applications.
* Are affected by all the political and business decisions that are made and need to be supported. Business systems will therefore often refuse to be standardized according to a software developer’s logic.

If you want a more detailed description of enterprise applications, the problems presented and common solutions then refer to Patterns of Enterprise Application Architecture by Martin Fowler which is in our opinion one of the best books on this topic

### Habanero

Principles driving the development of the Habanero Architecture

Automate the tasks common to programming and enterprise application. Make the difficult programming tasks easy and the complex tasks doable.

Dry (Do not repeat).

Objectives of the framework.

Maximise Coherence, Minimise Coupling, Eliminate Duplication.

Work equally well for Web (ASP), Web (AJAX enabled) and Windows application. Providing a responsive interface required for windows applications and providing a postback response interface required for Web (ASP) while suffering no performance loss for due to the additional capabilities provided for Windows application.

Provide a Business object framework that allows the UI/Application developer a natural programming experience where the complexities of … are hidden

A Layered architecture. For the time being we will be discussing the logical architecture. The logical architecture separates the architecture into its layers based on a logical separation of responsibilities. The physical architecture relates to the actual physical deployment of the software i.e. which computers the various components are deployed onto. The logical architecture is thus independent of the physical architecture. It is important that a layered architecture is implemented regardless of the physical architecture. The reasons for this are numerous and most of them come back to the separation of responsibility. Traditionally application architectures have been separated into 3 layers.



Figure 1: The Traditional 3 layers architecture.

1. Presentation Layer (UI Layer)

The presentation layer is solely responsible for the communication with the user, which is typically via forms and reports. Although we think of the presentation layer as primarily relating to Graphical intefaces the presentation layer of an application would include webservices that you publish, command line calls, batch processing interfaces etc. There should be no business logic in the presentation layer due to the fact that it is difficult to test via automated tests and maintain in this layer. The presentation layer layer handles the user interface presentation and navigation. From this it can be seen that the presentation layer can actually be subdivided into two layers. The Presentation layer follows the MVP pattern (TODO: Reference Pattern) with a passive view, which means that the views (the top UI layer) are managed and directed by the presentation layer. The presenters are responsible for giving the views their data, pulling data from the views to be saved in the lower layers, and in general for responding to events raised by the views.

1. Business logic layer (Domain Model layer)

The business logic layer is the layer where all business/domain specific logic resides. The business logic contains

* 1. The static rules and Business Object model, which contain the Entities of concern in the business domain (Business Objects/Entity Objects), the rules for how these objects relate to each other, the properties, the behaviour and the validation rules for the object (e.g. what properties are compulsory).
  2. The dynamic and business process rules used to process logic in the business (e.g. the rules for how an order is processed through order entry).

1. Data Storage Layer (Persistence Layer or Data Access layer).

This layer traditionally consists of the actual Data store (e.g. a relational database) and the retrieving and updating of the Business Objects from the Data Store. This layer can also be divided into two parts: Data Storage and object persistence. Data Storage involves the actual physical storage and management of data using SQL Server, Oracle, XML, WebServices, while object persistence refers to the synchronisation of the Business Objects’ state to the datastore.

For our purposes the traditional 3 layered architecture will be further broken down and will include the datastore thus providing us with 6 layers.



Figure 2: The traditional 3 layered architecture further subdivided.

##### The UI/Presentation Layer

The separation of the Presentation layer into a UI View layer and a UI Process layer may not be immediately obvious to Windows developers but to a web developer it is more obvious,

1. The UI View Layer would be the web browser, style sheets, javascript etc .
2. The UI Process layer is all the logic related to mapping business object data to controls, handling UI Events etc.

##### Business Logic Layer (Domain model + Application logic)

1) The business process layer. This layer is used to manage process logic. For instance, for the process of order entry the user must select the customer, select a product and enter a quantity. At this point the system will do a credit check. If the credit check passes, the order entry capture may continue, otherwise the order will be placed on hold and forwarded to a senior manager for approval. This layer is often called the workflow layer or the application logic layer. This layer is only required where complex process logic (workflow logic) exists.

2) The other type of business logic that is modelled in the business logic layer is the static model of the business. The static model consists of

1. The classes of concern in the problem domain and the relationship between these classes e.g. an order will reference one customer and will consist of one or more order items.
2. The rules for a class e.g. an order with order items cannot be deleted.
3. The properties of an order e.g. an order has an order number, order date and expected delivery date.
4. The property rules for the properties of a class e.g.
   1. An order number and order date are compulsory. When an order is captured the order date must be less than today and within the last 2 months.
   2. The order number and order date cannot be edited once the order has been persisted.
   3. The expected delivery date cannot be captured for a new order.

Many applications and architectures implement these two sub layers in one layer. It is however better to separate them out for the simple reason that the business process layer is likely to be subject to relatively frequent change whereas the static business model is usually subject to significantly less change. For this reason they should be separated so as to isolate the changes and simplify change management.

##### Data Storage Layer (Persistance Layer)

The data storage layer is the architectural layer that is related to the storage and retrieval of business objects.

The business objects may be stored in multiple formats e.g. Relational databases, Files, XML Files, Object databases or Web Services. Regardless of the data storage mechanism, the business objects typically require the ability to Create, Read, Update and Delete (CRUD) themselves from the data store. A single system may in fact have business objects from multiple datastores.

The Data Storage Layer of the architecture is therefore also logically divided into two parts.

1. The actual physical storage and retrieval of the data. In the case of storage in a relational database the data will be stored using a relational database management system (RDBMS, e.g. MySQL, Firebird, MS SQL and Oracle). This object data is retrieved and stored in the data store via the use of either SQL or stored procedures. The physical storage of the data thus forms one sub layer.
2. In Dot Net any data retrieved from a relational data base is typically read into an ADO.Net dataset or data reader. The relevant data from this dataset is then used by the Business Objects e.g. Order and order item.

Differentiating between the presentation layer and the domain layer. It is often difficult to differentiate between these two layers since they interact strongly to provide the user with their total experience. The guideline we use is that if you are developing a Web Application, any logic that would be duplicated when replacing this with a windows application should be in the domain layer.

Object Modelling objects about behaviour i.e. non duplication of code. Relational modeling about data i.e. non duplication of data. This can often result in a mismatch between the data model and the object model. This is commonly known as the object relational impedence or mismatch. By using the various patterns proposed in …. Enterprise architecture book? And allowing the modeling of these this mismatch can be eliminated.

Upgrading and integrating with legacy systems where the existing database(s) may have design flaws and cannot be redesigned. The architecture must allow the developer to implement his/her own ORM strategy for that object.

Generatability- Minimum number and size of templates, minimum code generation and duplication in templates. Principle if you have to duplicate then have a single store and generate the remainder.

Scaleable.

High performance.

Extendable

Testability.

Incremental implementation and total lifecycle management of the software solution.

Security and access control

Transactional support.

Concurrency control.

Duplicate object control

Business object Property rules

Custom and standard property rules – Standard rules. There are a number of standard rules that are applied in many situations e.g. a property is compulsory. In addition each data type will have a particular standard rules e.g. A string length min and max and pattern match. These are modeled and are therefore implemented with no code for the developer. The architecture also allows the capturing of custom rules e.g. A property value must be greater than another property value. These custom rules can easily be programmed and applied by inheriting from ..??

IDataErrorInfo – allow grid or error provider to automatically provide support for showing invalid data.

Undoable: the business object and each of its properties must have the capability to undo i.e. where the user makes edits to the business object and then chooses to cancel or closes the form and the user objects state must be cancelled.

Multiple datasources.

Frameworks:- A frameworks is a set of cooperating classes that make up a reusable design for a specific class of software. A framework provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. A framework captures the design decisions that are common in the application domain and thus emphasis design as well as code reuse. A framework will usually include a set of concrete subclasses that can be used as is. A framework will usually implement many design patterns. In fact a framework can be viewed as an implementation of a system of design patterns.

##### Coupling

Modules (Classes/packages) should be as independent as possible. Coupling is therefore a measure of the level of dependency between two modules.

##### Cohesion

Degree to which modules instructions are functionally related to each other. Highly cohesive modules contain instructions that collectively work together to solve specific tasks. The goal is to ensure that the modules exhibit a high degree of cohesiveness.

Object oriented design axioms

Axiom 1:- independence maintain the independence of the components.

Axiom 2:- Information. Minimise the information content of the design.

Occams razor:- The best theory explains the known fact with the minimum amount of complexity and maximum simplicity and straightforwardness.

Corollaries:-

Corollary 1: Uncoupled design with less information content. Highly cohesive classes and components can reduce coupling because only the minimum amount of essential information need be passed between objects.

Corollary 2: Single Purpose each class/component/method must have a single, clearly defined purpose. If you cannot clearly define the purpose of the class etc in a few sentences then you probably have not achieved this.

Corollary 3: A Large number of simple classes.

Corollary 4: Strong mapping. There should be a strong mapping between the physical system and the logical design.

Corollary 5: Standardisation. Promote standardisation by promoting interchangeable components and promoting reuse of existing classes/components.

Corollary 6: Design with inheritance. Common behaviour should be abstracted to superclasses.

Corollary 7: Design with interfaces.

Tracking an objects state, IsDirty IsNew, IsValid etc.

Object analysis and modeling.

Relationships between classes

1. Inheritance.
2. 1:M referenced.
3. Lookup lists.
4. 1:M composite.
5. M:M referenced.

BindingList why not using this for binding to forms and grids. System.ComponentModel.IEditableObject and System.ComponentModel.IDataErrorInfo.

Sorted views of collections. Rocky chapter 2 pg 87.

Command execution. Frequently the programmer tasks are related to updating and editing business objects. But there are also many times when the developer is required to execute a command or bulk task e.g. calculate depreciation at the end of each month. The execution of this type of command should be carried out in the logic layer. Where the logic layer acts on the business objects. In cases where the performance ?? move to a later section.

Defining Layers

User Inteface generalization and generation.

The generalization of the user interface is significantly more difficult than the generalization of the Business object layer. The reasons for this are that the flexibility and adaptability of the user interface design are in many cases what gives an application its uniqueness. Having said that we have found in our development that between 60 and 90% of a business application can be used from generated controls and forms. In addition to this the ability to rapidly produce an application with standardised generated forms contributes significantly to the ability to produce and useable system quickly as per the agile methodology.

MetaProgramming.

Resource localization for language independence: NNB there was a tool to do this see my brain add ins.

Reporting

DataAdaptor: Many reporting tools e.g. Crystal are still very much focused on the model of reading data directly from a database using SQL or stored procedures.

In some cases for performance reasons this is the correct approach. More frequently the approach of using SQL or stored procedures to retrieve and manipulate data directly results in duplication of logic (i.e. logic that is already tested and working in the business objects is duplicated in the report). Reports that cannot be tested using automated test tools such as NUnit. In these cases what the developer requires is the ability to produce the objects to a report. In these cases Active reports provides a better model for achieving this but if for whatever reasons another reporting tool such as crystal reports is used then a dataadaptor can be used the dataadaptor will provides the ability to transform a business object into a dataset. Which can be used as the datasource of a report. This cannot be used for large datasets but is usually preferred for most reports required. For a more complete discussion see reporting??