Implementation of Model-View-Controller Architecture Pattern for Business Intelligence Architecture

Medha Kalelkar
Vidyalankar Institute of Technology, University of Mumbai,
Mumbai, India

Prathamesh Churi
Lecturer, Department of Information Technology, Vidyavardhini's College of Engineering and Technology, University of Mumbai,
Mumbai, India

Deepa Kalelkar
Sardar Patel Institute of Technology,
University of Mumbai,
Mumbai, India

ABSTRACT
This paper presents a new approach to develop the strategy of Model–View–Controller architecture pattern in Business Intelligence (BI) architecture. The Business Intelligence architecture consists of the ETL tools and processes, the data warehouse, the technical infrastructure, and the user-interface tools. The major issue related to Business Intelligence architecture is to store the data from heterogeneous sources. The heterogeneous sources might lead to inconsistencies in storage and retrieval of data. Another issue related to Business Intelligence architecture is to provide multiple and synchronized views. The Business information must be presentable in various formats with 'look and feel' approach. Model-View-Controller provides independent approach to the components of Business Intelligence architecture. Model provides guideline for storing the heterogeneous data in one format and implements core functionality of the system. View provides multiple and synchronized views so that information can be available in presentable format. Controller handles user's input. All the three independent components are tightly coupled which ensures consistent and flexible Business Intelligence architecture.

General Terms
Software architecture, Business Intelligence

Keywords
Business Intelligence, Model-View-Controller, Metadata, Data warehouse, View, MVC, BI

1. INTRODUCTION
Any application is represented in its structured format using architecture pattern. The data flow, data storage, control flow is properly governed by architectural pattern. These architecture patterns are classified into four major categories, viz., from mud to structure, distributed system, interactive system and adaptable system [1]

Interactive systems focuses on user interaction [1] and thus they have proper user interface to justify the concept. To extend the usability of any software system we have high degree of user interaction. When specifying the architecture of such systems, the aim is to keep the core task independent of the user interface. The core of interactive systems is based on the functional requirements for the system, and usually remains stable. User interfaces, however, are often subject to change and adaptation. Modification of data must reflect the intent of other view which is associated with the other software system. Model-View-Controller architecture pattern divides task into three major components. Model handles core functionality of the system which is independent of user interface. View displays the information to the user in read-only mode. Controller implements event handling function that handles the user input.

Stackowiak et al. (2007) define Business Intelligence as the process of taking large amounts of data, analyzing that data, and presenting a high-level set of reports that condense the essence of that data into the basis of business actions, enabling management to make fundamental daily business decisions [2]. The basic Business Intelligence components are raw data, data warehouse, and metadata and end user tools. The major issue related to typical Business Intelligence architecture is to maintain the independency between core database operation and end user interface. Seperatabilty of these tasks is difficult in Business Intelligence architecture. Another issue of Business Intelligence architecture is to present the information in different formats when there is a requirement of multiple views associated with the same architecture.

However, Model-View-Controller solves the issue up to a great extent. Seperatability of component is the major characteristic of Model-View-Controller. Model component handles collection of raw data from heterogeneous data sources, handles Extract-Transform-Load process, and maintains metadata. View components displays the information stored in data warehouse in the form of graphical and textual view. Controller will handle the user input which is restricted to read-only mode i.e. user can only view the data in data warehouse. Any updation or addition of new data is restricted.

2. RELATED WORK
Architectural patterns [1] provide structural notations for any software system. They define each component with proper responsibility. They also provide some rules, axioms and relationships between them. They also encompass core data and functionality in the system. However, Model-View-Controller architectural pattern divides the software system into three subsystems, viz., model, view and controller.

Business Intelligence architecture has basic components, viz., ETL tools and processes, the data warehouse, the technical infrastructure, and the user-interface tools. It is used by stakeholders such as analysts to view the business data in different formats as per the requirement. Business Intelligence architecture is a type of interactive software system.

2.1 Model-View-Controller Architectural Pattern
The Model-View-Controller Architectural Pattern (MVC) is basically applied for interactive software systems [3]. The Model component contains the core functionality and data, the View component gives information to the user and the

Controller component handles user input using validation. View and Controller together comprise the user interface. A change-propagation mechanism [1] ensures consistency between the user interface and the model.

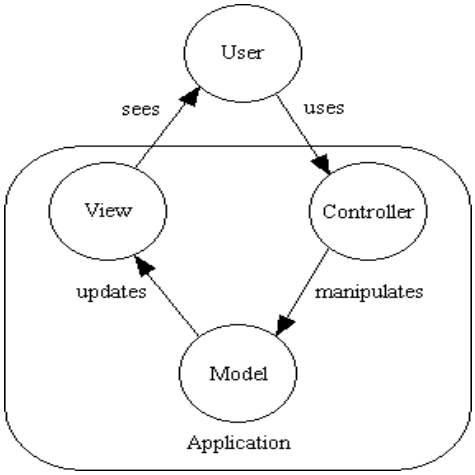

Fig 1: Model-View-Controller Architecture Pattern

The Class-Responsibility-Collaborator Cards diagram [4] for Model-View-Component is given below:

2.1.1 Model:

Model component of MVC encapsulates data and functionality. It is independent of its input and output behavior of the interactive system.

Table 1. Class-Responsibility-Collaborator Card for Model component

Class : Model	**Collaborators :**
Responsibilities : • Provides core data and functionality. • Informs the components about data modification.	• View • Controller

2.1.2 View:

As the name suggests, the View component always displays the information to the user. There can be multiple views associated with single MVC architecture. However, designing such multiple views is the major aim of our paper. Multiple views are highly synchronized with each other. Views must also be dynamic in nature. Each view has an associated controller component.

Table 2. Class-Responsibility-Collaborator Card for View component

Class : View	**Collaborators :**
Responsibilities : • Displays information to the user. • Retrieves the data from the model.	• Model • Controller
• Implementation of data update procedure.	

2.1.3 Controller:

Controller always follows the change-propagation mechanism. In future, if the user changes the controller of one view, all the views that are associated with it should reflect the same change; this property explains its dynamic behavior.

Table 3. Class-Responsibility-Collaborator Card for Controller component

Class : Controller	**Collaborators :** • Model • View
Responsibilities : • Accepts user handling. • Translates user events to request for the model. • Maintain dynamic behavior of synchronized views	

2.2 Business Intelligence Architecture

Business Intelligence (BI) systems are software applications that enable better understanding of organizational data and provide the information required by organizations to make accurate decisions. [5]

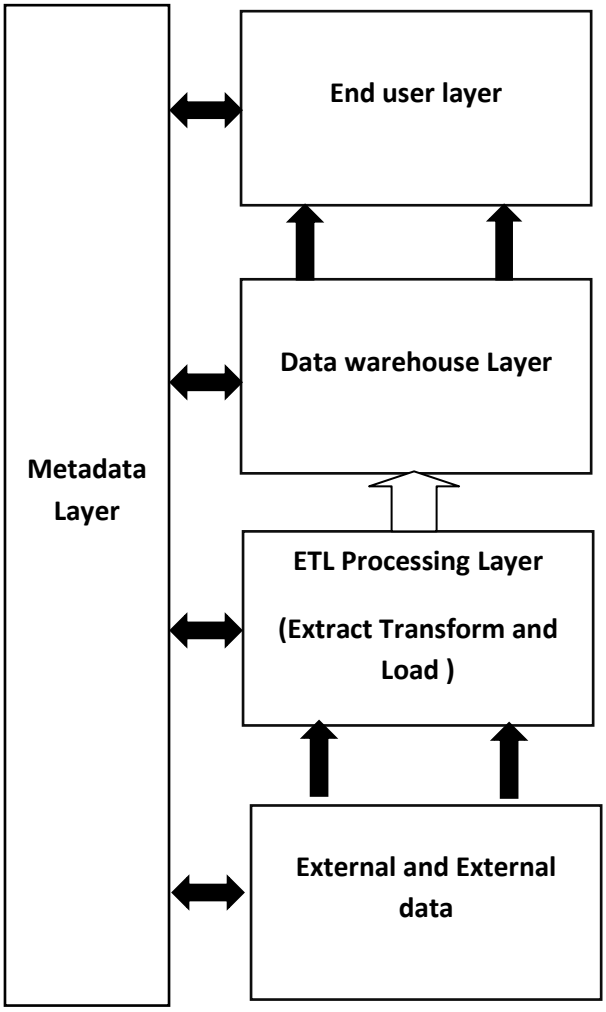

Fig 2: A Five-layered Business Intelligence Architecture

Business Intelligence Architecture is divided into five layers. The five layers are Data source layer, ETL layer, Data warehouse layer, End user layer and Metadata layer. [6] The description of five layers is described below:

2.2.1 Data Source Layer:

Data source layer includes both structured as well as unstructured data [7]. Internal data source [8] refers to data that is captured and maintained by operational systems inside an organization such as Customer Relationship Management [9] and Enterprise Resource Planning systems [10]. External data source [8] refers to those that originate outside an organization. This type of data can be collected from external sources such as business partners, syndicate data suppliers, internet, and government and market research organizations.

2.2.2 ETL (Extract-Transform-Load):

This layer mainly consists of three processes, viz., Extract, Transform and Load. Extraction process collects and identifies the relevant data from raw data sources. This layer also removes some noisy data [11] from multiple data sources. It is necessary to select relevant data for decision making with the help of extraction process.

On the other hand, transformation process is responsible for converting the data according to predefined business rules [12]. Data transformation process also includes defining business logic for data mapping and standardizing data definitions in order to ensure consistency across an organization.

The data staging [13] area is used to temporarily store the data before transformation process. This is done to avoid the need of extracting data again in case any problem occurs.

Loading is the last phase of ETL process in which the transformed data is loaded into target repository.

2.2.3 Data Warehouse Layer:

Data warehouse is subject oriented, integrated, time variant and non-volatile collection of data that supports management's decision processes [14].

Subject oriented data refers to the data from various sources that is organized into groups based on common subject areas that an organization would like to focus on, such as customers, sales, and products.

Integrated data refers to collection of consistent data in data warehouse in terms of naming conventions, formats, and other related characteristics.

Time variant data refers to storage of data along with its time dimension to keep track of the changes or trends on the data.

Non-volatile data refers to store the data in Read-Only mode. Users are not allowed to modify or re-write the data in data warehouse.

2.2.4 Metadata Layer:

Metadata [14] gives details of data. It describes the location where the data is stored, the purpose for which the data is used, information about source data and other useful information about data. Metadata repository is used to store technical and business information about data as well as business rules and data definitions.

2.2.5 End User layer:

End user tools are the programs that display information to end users. End user tools can provide multiple views of same information for different types of users. From implementation point of view it might be difficult to create multiple views of same Business Intelligence model. However, we must create synchronization among these views.

The implementation of predefined architectural pattern Model-View-Controller in basic architecture can avail following benefits which are tabulated below. (Benefits are tabulated using Non-functional properties of software architecture [15]).

Table 4. Benefits of MVC architecture in typical BI architecture

Benefit	Description
Efficiency	Business Intelligence systems using Model-View-Controller are highly efficient in terms of storing the heterogeneous data in data warehouse, synchronization of views.
Complexity	Dividing the task into independent components, viz., Model, View, and Controller reduces complexity of Business Intelligence architecture.
Scalability	The business applications becomes highly scalable so that new components of business applications can be added or removed without affecting the functionality of the Business Intelligence system.
Heterogeneity	Business Intelligence systems are heterogeneous so that they can run on different computing environments [16]
Maintainability	Business Intelligence systems maintain synchronized views, consistency in data when they are implemented using Model-View-Controller architectural pattern.

3. METHODOLOGY

The implementation of Model-View-Controller in Business Intelligence architecture might be tedious task. The approach of above problem is briefly described below:

3.1 Context

Design Interactive Business Intelligence applications with multiple and synchronized views. Software system should also produce flexible Human computer interface so that different types of stake holders [17] can view interactive applications.

3.2 Problem

The problem of typical Business Intelligence architecture is that the amount of data is vast. The data processing capability can also be varied from one business organization to another

business organization. The operational systems [18] provide the basic data that feed the data warehouse either in real-time or on a periodic basis. The underlying foundation of a BI architecture is complex. The implementation can either facilitate Business Intelligence or become so monolithic and inflexible that it becomes a technical data wasteland.

Another problem associated with Business Intelligence architecture is the maintainability of synchronized views. Building a Business Intelligence system with the required flexible views is expensive and error-prone when the user interface is tightly interwoven with the functional core. Different types of stake-holder may demand to represent the data in different forms such as bar-charts, tables, histogram etc.

The problem is formulated with the help of following points:

- The same business information must be presentable in different format.
- Changes applied to user interface must be flexible and easy.
- Business Intelligence architecture must support 'look and feel' approach to their views to show business information.

3.3 Solution

The obvious solution of above problem is to divide the business application into three basic parts viz. input, processing and output. Each part is independent of each other in their respective functionality but must have a tight synchronization.

Model-View-Controller is the solution of above problem. Model contains core data and functionality, i.e., it contains source data, metadata and the data present in data warehouse. The model is completely independent of specific output representation of input data.

View is responsible for displaying the output of data ware house to the user. There can be multiple views of same information. For example, bar-charts, dashboards etc.

Controller handles user input. Each view has individual component associated with it. Event handling technique can be used to handle the user input.

3.4 Structure

The structure of three individual components viz. Model, View, Controller can be shown with the help of Class-Responsibility-Collaborator Cards below:

Table 5. Class-Responsibility-Collaborator Card for Model component with respect to Business Intelligence Architecture

Class : Model	**Collaborators :**
Responsibilities : • Provides core data and functionality to business application. • Maintains data in consistent state in data warehouse. • Handles Extract-Transform-Load process. • Informs the components about the data modification.	• View • Controller

Table 6. Class-Responsibility-Collaborator Card for View component with respect to Business Intelligence Architecture

Class : View	**Collaborators :**
Responsibilities : • Displays information of data warehouse to the analyst. • Provides user flexibility to display the information in different formats.	• Model • Controller

Table 7. Class-Responsibility-Collaborator Card for Controller component with respect to Business Intelligence Architecture

Class : Controller	**Collaborators :**
Responsibilities : • Accepts user input (event handling). • Translates user events to requests for the model.	• Model • View

4. CONSEQUENCES

The benefits of Model-View-Controller implementation in Business Intelligence architecture includes following points:

1. **Implementation of Multiple views for the same Business Intelligence model.**

As the Model-View-Controller architecture separates the model from the view, multiple views can be created for a single model. While implementing Business Intelligence architecture for any organization, there is a requirement of multiple views for representing data in a desired form (such as graphs, pie charts etc.). Different type of stakeholders (managers, analysts, developers) may require different views of data in different formats. Model-View-Controller architecture helps to create multiple views by keeping the model and controller intact.

2. Maintainability of synchronized views:
The change-propagation mechanism of the model ensures that all related observers are notified of changes to the application's data at the same time. This synchronizes all the dependent views and controllers. Synchronized view ensures simplicity in Business Intelligence architecture.

3. 'Pluggable' views and controllers.
The conceptual separation of Model-View-Controller allows you to exchange the view and controller objects of a model. User interface objects can even be substituted at run-time for different types of stakeholders.

4. 'Look and feel' approach in User interface. [1]
Look and feel approach refers to the creation of different types of views in the user's perspective. While viewing the interface of Business Intelligence, any stakeholder may find simplicity of handling objects. Look and feel approach however increases the understanding of user interface of any business model.

Liabilities of Model-View-Controller are as follows:

1. Complexity:
Complexity is the degree at which software model and its construction is difficult to understand [1]. For any interactive application Model-View-Controller is not always a good option. For the Business Intelligence architecture it may be suitable as we can create multiple views of same pattern. Designing the code which includes core data and functionality for controller and model might be complex.

2. Controlling excessive number of updates:
If a single analyst's action in user interface results in many updates, the Model component should skip unnecessary changed notifications. The Business Intelligence Architecture, may require controlling of the unnecessary updates.

3. Tight coupling of views and controllers to a model:
Both view and controller components make direct calls to the model. This implies that changes to the model's interface are likely to break the code of both view and controller. This problem is magnified if the system uses a multitude of views and controllers.

5. FUTURE SCOPE
Future scope of this paper may be the alternative solution to Model-View-Controller architectural style. As we know it is difficult to implement multiple views of same model with proper synchronization, we can implement other architectural interactive system pattern such as Presentation-Abstraction-Control (PAC) [1]. PAC uses hierarchy of interactive agents. It completely separates human interaction facility from interactive agents.

Another modification that we can do in Business Intelligence architecture is to implement Document view [1]. We can integrate the responsibilities of the view and the controller from MVC in a single component by sacrificing exchangeability of controllers.

6. CONCLUSION
Model-View-Controller ensures the separation of core data processing and functionality, consistencies among different views, synchronization of views, 'look and feel' approach of user interface, flexibility in handling different views, and representation of an information in different form. However, the objective of interactive system is to provide high usability to the business application. Using Model-View-Controller (which is an interactive system) we can have multiple views as per the requirements of our Business Intelligence system.

7. AUTHORS
First Author – Medha D. Kalelkar is currently pursuing Master's program (M.S.) in Management Information System from Syracuse university school of Information Studies, United States. She has completed her Graduate program in Computer Engineering from Vidyalankar Institute of Technology affiliated to University of Mumbai, India.

Second Author – Prathamesh P. Churi is currently working as a lecturer in Vidyavardhini college of Engineering and Technology affiliated to University of Mumbai, India. He is also pursuing Master's degree (M.E) in Information Technology from Shah and Anchor Kutchhi Engineering College affiliated to University of Mumbai, India. He has completed his Graduate program in Computer Engineering from Vidyalankar Institute of Technology affiliated to University of Mumbai, India.

Third Author – Deepa D. Kalelkar has completed her Graduate program in Electronics Engineering from Sardar Patel Institute of Technology affiliated to University of Mumbai, India. .

E-mail: deepakalelkar7@gmail.com

8. REFERENCES
[1] The format for defining Business Intelligence Architecture in terms of Model-View-Controller architecture pattern we have reference book : Frank Buschmann, Regine Meunier ,Hans Rohnert ,Peter Sornmerlad , Michael Stal, Pattern Oriented Software architecture, volume 1, February 2001.

[2] Business Decisions: http://ijircce.com/upload/2013/june/26_BUSINESS.pdf.

[3] About Model-View-Controller http://en.wikipedia.org/wiki/Model-view-controller

[4] Concept of Class responsibility collaboration card: http://en.wikipedia.org/wiki/Class-responsibility-collaboration_card

[5] http://www.b-eye-network.com/view/7105

[6] Five layered Business Intelligence architecture: http://www.ibimapublishing.com/journals/CIBIMA/2011/695619/695619.pdf.

[7] concept of structured and unstructured data: http://www.brightplanet.com/2012/06/structured-vs-unstructured-data/

[8] Concept of internal and external data: http://chdexpert.wordpress.com/2012/03/30/database-fundamentals-internal-and-external-data-sources/.

[9] Concept of Customer Relationship Management: http://en.wikipedia.org/wiki/Customer_relationship_management

[10] Concept of Enterprise Resource planning: http://en.wikipedia.org/wiki/Enterprise_resource_planning

[11] Concept of data noisy data: http://www.mimuw.edu.pl/~son/datamining/DM/4-preprocess.pdf

[12] http://en.wikipedia.org/wiki/Business_rules_approach

[13] concept of data staging: http://data-warehouses.net/architecture/staging.html

[14] for data warehousing fundamentals and characteristics: Data Warehousing 2.0 by Inmon

[15] Non-functional properties of software architecture: Software Architecture: Foundations, Theory, and Practice by Richard N. Taylor.

[16] http://www.garlic.com/~lynn/secure.htm

[17] concept of stakeholders: http://en.wikipedia.org/wiki/Stakeholder

[18] operational system in data warehouse : http://data-warehouses.net/architecture/operational.html