Constructing a Plug-In Algorithm Generic Database for Future Enterprise

Syed Rehan Zafar and Muhammad Nadeem
SZABIST
Karachi, Pakistan

Abstract: *An important function of a data engineering system is to support incremental and corporative construction of Plug-In Algorithm in an orderly way. Basic idea for this paper is to identify and explore a new vision for generic database modeling and its implementation. In this paper, we focus on what is the importance and how an analytical framework works in this research to give a new concept of developing a generic schema after a continuous learning, and practices of various domains.*

Keywords: *Design pattern, object oriented programming, inheritance, composition, aggregation, framework, MMIS*

1. INTRODUCTION

The advancement in the computer technology (both hardware and software) has led us to see its use and application in the varied diverse existing, new and emerging areas. Through this paper we basically focus on the process of implementing a plug-in algorithm and integrating that algorithm into Analysis Services, including with the stub code and develop the testing phase of a "shell" plug-in algorithm integrated into Analysis Services [1]. It means that knowledge developers of data mining algorithms can concentrate their efforts on their algorithms, as opposed to worrying about integration with Analysis Services.

This paper also identifies and explores a new vision for generic database modeling, development of new concepts of generic schema after continuous learning and practices of various domains e.g.: variety of choices for employee records in textile industry and automobile industry etc. Customer/Domain experts will provide information, this will increase the learning of the database using the learning tier; and will be responsible for checking existing information in the generic schema, generic patterns and standards that are required for those domain experts. In this way, schema will update itself and in the end it will become a generic system that can fulfill any Enterprise and any other these types of systems. So, it will eventually lead towards a very generic Database schema which acts like a brain of the system.

Schema is responsible not only to store the field or data required in that domain, but also stores the behavior of those domains and maintains metadata which will also be helpful to use this system to any existing system. In this way, this metadata tier will act as communicator and negotiator between the database creational tier and the database of another system.

Schema also holds the behaviors of the domain. In this way, it will store the best practices and standards, which are used by those domains. Schema also helps to develop the architectural patterns that contain enterprise level patterns layers and other functional and structural patterns information; that can be used at any level of complete enterprise applications.

Using patterns and services, repository system can identify where/what service(s), what operation(s) and which architecture is/are required through the metadata; and generic data types information.

1.1 Framework

A framework is a reusable design defined by a set of abstract classes and the ways their instances collaborate with each other. This set of cooperating classes makes up a reusable design for a specific class of enterprise software [2,3]. For example, a framework can be geared towards building graphical editors for different domains like artistic drawing, music composition and mechanical CAD [4], [5]. Another framework can help in building compilers for different programming languages and target machine [6]. Yet another might help in building financial modeling applications [7]. A framework can be customized to a particular application by creating application specific subclasses of abstract classes from the frameworks.

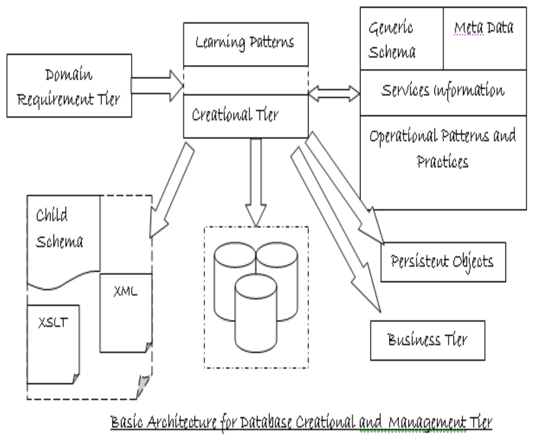

Fig. 1. Generic Database Architecture

By definition, a framework is an object-oriented design. It does not have to be implemented in an object-oriented language, even though it usually is. According to [8], it is useful to classify framework by their scope, although the benefit and design principle underlying frameworks are largely independent of which domain they are applied to. There are three categories of framework according to their scope-system: infrastructure framework, middleware integration framework and enterprise application framework [9].

Further, white-box frameworks require application developers to have comprehensive knowledge of the frameworks' internal structure; so that developers can extend and reuse the existing framework through inheritance or generalization. It relies heavily on object-oriented language features, like inheritance and dynamic binding, to achieve extensibility.

In contrast, black-box frameworks support extensibility by defining interfaces for components that can be plugged into the framework via object composition. They are structured using object composition and delegation more than inheritance. As a result, black-box frameworks are generally easier to use and extend than white-box frameworks [4, 9]. However, black box frameworks are more difficult to develop since they require framework developers to define interfaces and hooks that anticipate a wider range of potential use-cases [10].

1.2 Framework for Plug-in Algorithm for Generic Database

A framework states the architecture or blueprint of any application. It defines the overall structure, the partitioning into classes and objects, how the classes and objects collaborate [9], what their functions and controls are. A framework predefines these design parameters in an abstract level, so that the application designer or implementer can concentrate on the specifics of any application. Plug-in Algorithm Framework Generic Database developed, hence, captures the design decisions that are common to the various domains. For example, the database connections, table schema, triggers, behavioral procedure, DDL and DDM capture data from user and display output to screen [1]. These are the common elements that the system will learn and stores this information. We are using our Algorithm with framework to define a generic way of learning and searching mechanism. This will help us in all domains, and then emphasize design reuse over code reuse.

1.3 Design Patterns

"Each pattern describes a problem which occurs over and over again in our environment and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" [9]. Focusing at structures that solve similar problems, one could distinguish similarities between designs that were high quality or efficiency. These similarities are called patterns. Therefore, software design patterns actually arose from architecture and anthropology [11]. So, design pattern names, abstracts and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design.

2. PLUG-IN ALGORITHM FRAMEWORK ARCHITECTURE

Plug-in Algorithm Framework Architecture is best-expressed using class diagram. According to [5], three perspectives can be used in drawing any UML model, but this breakdown is most noticeable in connection with the class diagram.

The first perspective is conceptual and conceptual diagrams should represent the concepts related to classes that implement them; but there is often no direct mapping. In fact, a conceptual model should be drawn with little or no regard for the software that might implement it. Thus it can be considered language-independent. The second perspective is specification. This step focuses on the interfaces of the software but not the implementation. Object-oriented development puts great emphasis on interface and implementation. However, this may not be true in practice because the notion of class in an object-oriented language combines both interface and implementation. Last is the implementation perspective. This is the most often used perspective because the software developer will follow the blue print to write the actual code.

3. CONCLUSION

The result of the research paper is a flexible and a generic framework for plug-in algorithm architecture, which includes the framework. It is a very robust framework, which is reusable and extended able. The existing frameworks that are in use lack reusability and has performance problem.

The architecture is hardware independent and also software independent. The front end of the architecture could be desktop application or the web application; it is flexible enough to encompass both of them quite easily.

Another point of interest is the database independence. The framework is designed to handle all type of database. There are many features that are specific to database and it is quite reasonable to use them; as they are designed specifically to that database system. All of these handlings are done in separate database controller class. The framework does know about what database is used. The database factory is responsible for creating the appropriate database class. The framework talks to a base class.

3.1 Future Work

The main work now begins by implementing the framework in any language, as required. Whether it is

Java/C# or any language that supports object oriented programming, this framework can be implemented. However, one thing that might be considered while implementing the framework is that the language in which it is implemented must be able to support all the new features like mobile communication/database connection and language; itself has a framework that is extensible otherwise compatibility issues may rise.

3.2 Challenges

According to the Gang of Four theory and concepts, the hard part about object-oriented design is decomposing a system into objects. The task is difficult because many factors come into play: encapsulation, granularity, dependency, flexibility, performance, evolution, reusability etc. They all influence the decomposition in the conflicting ways. This robust and extensible framework structure is very hard to design without experience in software architecture design.

REFERENCES

[1] Fernando Guerrero, Microsoft SQL Server 2005, Database essential step by step.

[2] Deutsch, L. P. (1989). *Design reuse and frameworks in the Smalltalk-80 system*. In Biggerstaff, T. J., & Perlis, A. J., editors, *Software Reusability, Volume II: Applications and Experience*, pages 57-71. NY, USA, ACM Press, Addison-Wesley.

[3] Johnson, R. E., & Foote, B. (June / July 1988) *Designing reusable classes*. Journal of Object-Oriented Programming, 1(2): 22-35.

[4] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Pattern: Element of Reusable Object-Oriented Software*. Boston, MA, USA, Addison Wesley.

[5] Johnson, R. (October 1992). *Documenting frameworks using patterns.* In Object-Oriented Programming Systems, Languages, and Applications Conference Proceedings, Pages 63-76, Vancouver, British Columbia, Canada. ACM Press.

[6] Johnson, R. E., McConnell, C., & Lake, J. M. *The RTL system: A framework for code optimization*. In Giegerich, R., & Graham, S. L., editors, *Code Generation—Concepts, Tools, Techniques*. Proceedings of the International Workshop on Code Generation, pages 255-274, Dagstuhl, Germany. Springer-Verlag.

[7] Birrer, A., & Eggenschwiler, T. (July 1993). *Frameworks in the financial engineering domain: An experience report.*

[8] Campbell, R. H. & Islam, N. (1993). A Technique for Documenting the Framework of an Object- Oriented System, Journal of Computing Systems, 6(4).

[9] Fowler, M. & Rice, D. (2003). Patterns of Enterprise Application Architecture, Addison-Wesley, USA.

[10] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). *A Pattern Language*. New York, Oxford University Press.

[11] Inmon, W. H. (2002). Building the Data Warehouse. John Wiley & Sons, third edition.