Malaysian Journal of Computer Science, Vol. 13 No. 1, June 2000, pp. 56-64

OBJECT-ORIENTED APPLICATION FRAMEWORK ON MANUFACTURING DOMAIN

Sai Peck Lee, Siew Khim Thin, Hong Song Liu
Faculty of Computer Science and Information Technology
University of Malaya
50603 Kuala Lumpur
email: saipeck@fsktm.um.edu.my

ABSTRACT

The problematic issues in the development of manufacturing software systems are related to the various nature of manufacturing systems, which are wide, dynamic and complex. The purpose of this paper is to provide a solution by using framework-based software engineering to solve these problems. Framework-based software engineering is the idea of constructing software systems based on the integration of reusable components rather than developing software from scratch. The manufacturing application framework proposed in this paper is to be developed as a set of integrated reusable components, which can be adapted to suit specific manufacturing applications. This paper focuses on two subdomains of the manufacturing domain, i.e., Production Management (PM) and Statistical Quality Control (SQC). A generic model is defined based on the structure and behavior of each of the PM and SQC subdomains, from which the design infrastructure of the application framework is derived, based on the concept of design pattern. The details of the application framework development by integrating object-oriented technology and component-based development to achieve large-scale software reuse for manufacturing application develop-ment projects are also discussed.

Keywords: ***Application Framework, Component-Based Development, Manufacturing Domain, Object-Oriented Technology***

1.0 INTRODUCTION

Over the last two decades, software development has changed significantly. Its evolution has been motivated largely by the quest to help software developers produce software faster and to deliver more value to end-users. The limitations of traditional programming and today's system software environments have motivated the software industry to start to embrace systematic software reuse due to its potential to significantly increase the developer's productivity and encourage innovation. Large-scale software reuse is able to improve quality and reduce cost and time-to-market in software development [1].

Object-oriented application framework (OOAF) is an idea that comes from the combination of object-oriented technology (OOT) and application framework. The focus should not only be on OOT or application framework, but on how OOT should be delivered in order to fully realize the benefits that it can offer in the development of application frameworks. An OOAF, which is an extensible set of object-oriented reusable components that are well integrated to execute well-defined sets of computing behavior on a certain application domain, is a good answer to a dramatic improvement in software development. This paper discusses the notion of the OOAF and how it can be used to assist software development in the domain of manufacturing.

2.0 MANUFACTURING SYSTEM DOMAIN

2.1 Problematic Issues

Manufacturing is a wide, complex and dynamic domain. As such, to develop a flexible and good quality manu-facturing software is not an easy task.

Manufacturing is considered as a wide-range domain, due to its numerous types of manufacturing operations [2]. Three principal manufacturing operations are Job Shop, Repetitive Manufacturing and Batch Manufacturing. A Job Shop manufacturing business conducts Make To Order (MTO), which only makes when customers place order, and the design of the products is according to the design supplied by the customers. Some of these businesses engineer and build items based on performance requirements specified by the customer. Job Shop generally deals with products of high variety and low volume in quantity. For example, scientific equipment and medical equipment are the products of Job Shop. Repetitive Manufacturing, repetitive production and production lines are terms used for mass production facilities that produce a high volume of the same or similar units of products that follow the same path through the production steps. Repetitive Manufacturing generally deals with products of low variety and high volume in quantity. An example would be an automobile assembly line. A Batch Manufacturing facility makes some intermediate variety of products and produces intermediate volume of each product. Batch Manufacturing is a hybrid of Job Shop and Repetitive Manufacturing. An example would be a company that makes small hand tools like hand mixers, electric screwdrivers, etc. As such, the requirements in software utilities for these different types of manufacturing

are quite different. There is no one kind of basic structure like an application framework that can be specified to the similar but non-identical types of applications to suit the similar but non-identical manufacturing operations. An application framework is highly adept to the above-mentioned differences of the manufacturing environment.

Manufacturing software is one of the most complex software. There are a lot of applications in this domain, such as production planning, inventory control, work-in-process tracking, scheduling, production operation control, capacity planning, process capability analysis, process control, control chart plotting, acceptance sampling, statistical analysis, etc. A problem facing this industry is that these applications are not designed to work together and are difficult to integrate. Also, it is very difficult and expensive to develop an integrated manufacturing software from scratch. Instead of developing the manufacturing software from scratch, a better choice is to reuse some generic design and implemention built into an application framework.

Manufacturing is a dynamic domain. A lot of changes happen from time to time such as the change of technology, standard of production regulation, customer's specification, report's format, etc. These changes will cause the change of process in manufacturing as well as the software that assists its process. For example, when a new technology is invented, from CD-ROMs to DVD players or Pentium II to Pentium III processors, the process and the software that keep track of the products in the production line might have to be modified. As such, the manufacturing software must be as flexible and as extensible as possible.

2.2 General Requirements

The manufacturing domain can be divided to include two subdomains, Production Management (PM) and Statis-tical Quality Control (SQC).

Production Management is designed to control the production and inventory in a manufacturing firm by conducting several activities such as production planning, inventory control, capacity planning, etc.

For Statistical Quality Control, SQC methods are useful tools for appraising and monitoring quality performance and are key ingredients to successful application of quality control. Quality control relies on the continuous monitoring of quality of the input and output of the processes producing products and services. For example, a control chart is used to monitor the quality of the products in the process of the production line.

2.2.1 Production Management

There is a widely used generic version of production management system, called Manufacturing Resource Planning (MRP II) System [2]. The common structure of the MRP II system is shown in Fig. 1.

Each of the boxes represents a separate subsystem. Each subsystem performs certain functions. The arrows in the diagram indicate information flows. This system is divided into three levels. The first level is the Top Management Planning level, which consists of production planning and resource planning subsystems. Some manufacturing companies also include business planning and marketing planning subsystems into this level. At this level, there are long planning horizons (about five years) and broad decisions. The second level is the Operations Management Planning level, which consists of Master Production Scheduling (MPS), Rought-Cut Capacity Planning (RCCP), Material Requirement Planning (MRP) and Capacity Requirement Planning (CRP) subsystems. At this level, there are short planning horizons and more specific decisions. The last level is the Operations Management Execution level, which consists of the Production Activities Control (PAC) subsystem. This level is to make sure productions and jobs are in schedule.

The object-oriented application framework proposed in this paper, is to be built based in part on the common structure of the MRP II system, in such a way that, it is able to be used to create applications of the MRP II system by adapting the generic component systems provided by the framework. For example, the generic forecasting component system of the framework can be used to extend to any specific forecasting application method as needed, such as moving average method, exponential smoothing method, Box-Jenkins method, etc.

2.2.2 Statistical Quality Control (SQC)

SQC is the application of statistical techniques to ensure satisfactory quality. There are three major subgroups of techniques: statistical process control, acceptance sampling and traditional statistical techniques, as shown in Fig. 2. In Statistical Process Control (SPC), tools called control charts are used primarily to prevent or detect production of defective products (finished goods, assemblies, components) or service. This allows the process to continue or stop and inspection is carried out to find the cause. There are two types of control charts: control charts for attributes and control charts for variables. Acceptance sampling is the application of statistical techniques to determine whether a quantity of material should be accepted or rejected, based on the inspection or testing of a

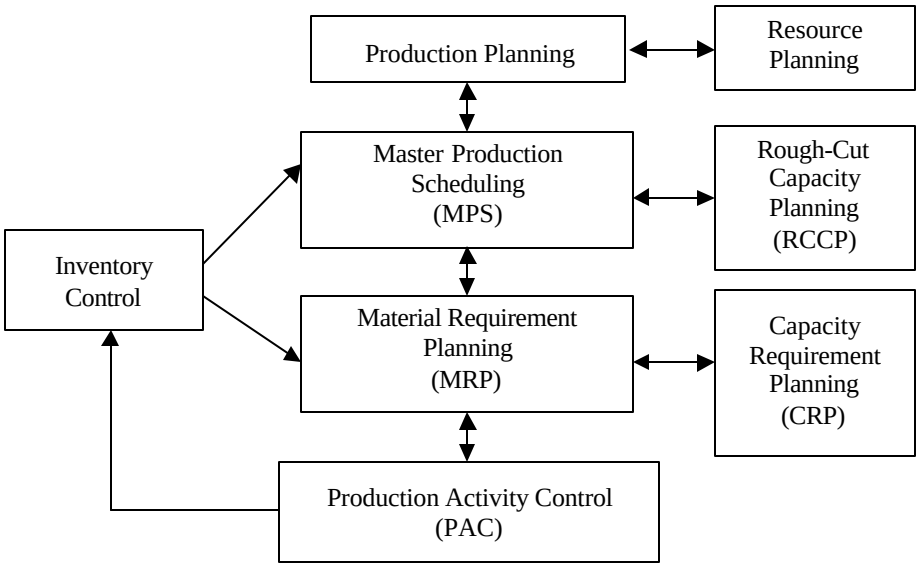

Fig. 1: Generic model of Manufacturing Resource Planning (MRP II) System

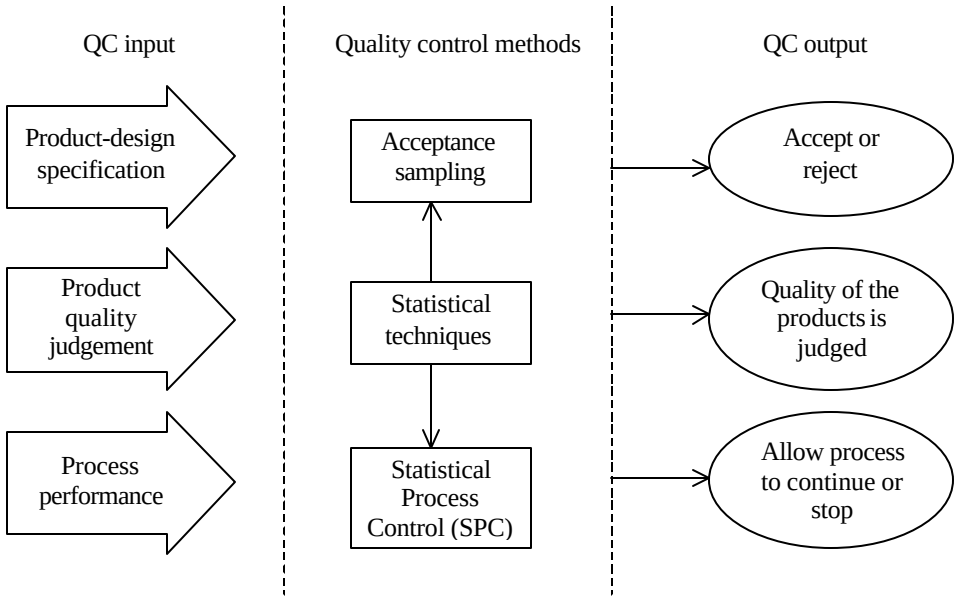

Fig. 2: The basic structure of a statistical quality control system

sample. To judge the quality of products or service, traditional statistical techniques such as frequency distribution, measures of central tendency and dispersion, regression and probability distribution can be used [3].

The development of the object-oriented application framework is based also on the common application systems of the SPC, acceptance sampling, and traditional statistical technique which are widely used in quality control in manufacturing. The construction of a control chart application system basically involves implementing a reusable production control component in the framework for the purpose of adaptation or configuration. A generic production control component will be provided by the application framework to support a variety of control charts such as R-chart, X-chart, n-chart, etc, in one of its variability points meant for adaptation according to the specific needs of the reuser.

3.0 OBJECT-ORIENTED APPLICATION FRAME-WORK (OOAF)

3.1 Fundamental Principles

Achieving widespread reuse of complex software components requires a concerted focus on the fundamental design of the application framework that underlies software

systems on a certain application domain. An application framework aids the development of software by expressing the structure and collaboration of components to developers at a level higher than source code or object-oriented design models that focus on individual objects and classes. It facilitates reuse of software architecture [4].

If object-oriented (OO) software is to become an engineering discipline, the successful practices and design expertise must be documented systematically and disseminated widely. An application framework is an important tool for documenting these practices and expertise, which traditionally existed in the minds of expert software architects or buried deep within the source code of complex systems. Without a thorough understanding of the framework underlying a domain-specific architecture, design, and implementation, OO software reuse will remain a largely unfulfilled promise.

An OOAF is a skeleton implementation of an application or application subsystem in a particular problem domain. It is technically composed of concrete and abstract classes and provides a model of interaction or collaboration among the instances of classes defined by the framework [5]. A family of similar, though not identical, applications can be built out of a single framework. A framework is also a methodology and a set of software libraries that allow software developers to build applications. The following summarizes the major advantages of a framework [6]:

- **Provide an infrastructure and an architectural guidance:** By virtue of the interconnections among the class libraries, much of the needed functionality already exists in the framework, thus reducing coding, testing, and debugging. In addition, object-oriented frameworks encourage better design in the code in the sense that developers are provided an "example" to guide them to more effectively utilize object technology. Applications developed from frameworks tend to be smaller, as well as more maintainable and reusable.
- **Provide a mechanism for reliably extending functionality:** While objects and object classes provide interfaces for extending functionality at a fine-grained level, frameworks provide this flexibility at a higher level. In this way, applications can be developed by using the framework as a starting point and writing smaller amount of code to modify or extend the framework's behaviour. These extensions can be added without sacrificing compatibility and interoperability because the interfaces are well defined.
- **Reduce maintenance:** Due to inheritance, when a framework bug is fixed or a new feature is added, the benefits of those changes become available more quickly to the derived classes or components. Also, changes are made only in one place, thus, the chance of introducing additional errors in the code is minimized.

3.2 Literature Review

A review of the literature on frameworks in general, reveals that most of them have their subjected advantages for programmer productivity and code reliability. Much research has been done on this topic for more than a decade. Frameworks are important, and continue to become more important. There are quite a number of available frameworks such as Taligent, Model View Controller (MVC), Domain-Specific Software Architecture (DSSA), Java Bean, OLE, etc.

Taligent's framework is an independent joint venture of Apple Computer, Inc., IBM Corporation, and Hewlett-Packard Company. This framework defines the behavior of a collection of objects, providing an innovative way to reuse both software design and code [6]. Taligent has designed frameworks for system software functions, such as networking, multimedia and database access. Taligent's framework provides two kinds of Application Programming Interface (API). The first API is the client API which is used by other frameworks or application developers. This API is for developers who just want a simple abstraction and to simply be a user of the framework's services. The second API is known as the framework API and is for developers who want to take advantage of the flexibility and extensibility that the framework provides. This API provides all the hooks to the developer's codes that modify the framework's behavior to provide the desired software- or hardware-based solution.

Model View Controller (MVC), as a user-interface framework, is actually a design pattern which was developed using the Smalltalk programming environment for the creation of user interfaces [7, 8]. The MVC paradigm is a way of breaking an application, or even just a piece of an application's interface, into three parts: the model, the view, and the controller. The advantage of the MVC paradigm is that it effectively limits and defines the interaction between the interface components and the underlying problem-domain classes. The *Model* is the problem-domain class on which an interface is to be created. There is an actual class Model for subclassing to create any problem domain class to be affected by an interface or to display to an interface. The *View* is the class that receives input from the user (e.g., button clicks, key presses, etc.) and displays output to the user (e.g., display boxes, colored symbols, visualizations, etc.). The *Controller* is the class that deals with the physical input devices and translates those signals (mouse movement, keyboard input) into messages to the appropriate view.

A good framework facilitates application system development, promotes achievement of functional requirements, and supports system reconfiguration. The Domain-Specific Software Architecture (DSSA) provides, a reference architecture designed to meet the functional requirements shared by applications, principles for decomposing expertise into highly reusable components,

and an application configuration method for selecting and configuring components within the architecture to meet a particular application requirement [9].

4.0 METHODOLOGY IN THE DEVELOPMENT OF THE OOAF

The methodology used in this project is based on the integration of some existing concepts of Object-Oriented Domain Engineering (OODE) modeling notation and method, which makes it easy to compose reusable components [10, 11, 12, 13]. It is a systematic technique to establish good system architecture to help in application framework development at the same time controlling the complexity of the system. The develop-ment process supported by the methodology for the framework is referred to as Application Architecture Engineering (AAE) as shown in Fig. 3.

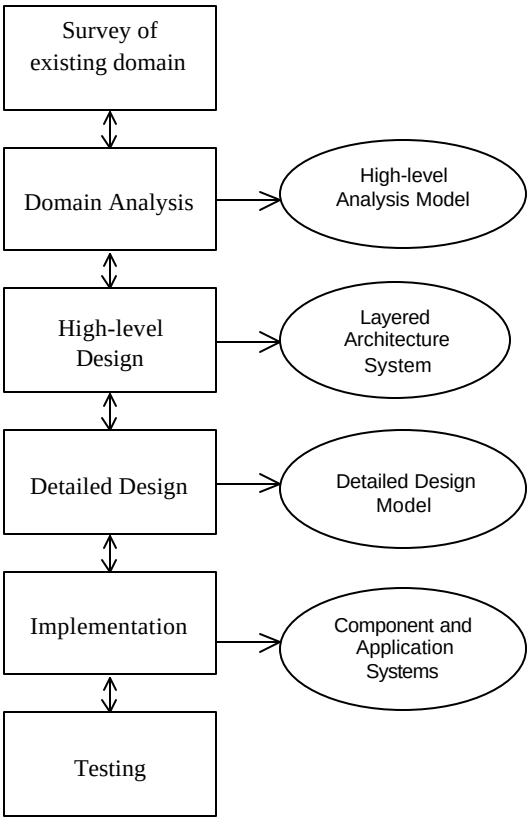

Fig. 3: Stages of Application Architecture Engineering (AAE) process

AAE starts from domain analysis through to implementation and as the way to decompose the overall standardization set of applications into a suite of application systems and supporting component systems. The process architects the Layered Architecture System which consists of the interfaces of the subsystems and component systems that support the complete architecture of related applications. There are five iterative stages of the process described as follows:

- **Survey of existing domain systems:** The survey lays the foundation of knowledge about the domain. In the survey, the requirements of each domain system's external actors, business processes and models are captured.
- **Domain analysis:** Identifies the functionalities and understands the variable and non-variable functionalities in the domain. Non-variable function-alities are good candidates for generic component systems. The main aim is to identify the candidates for application and components systems for constructing a high-level analysis model.
- **High-level design:** The high-level analysis model serves as a foundation for producing a prototype design model that defines the layered architecture system in terms of application systems and component systems. The interfaces of the application systems and component systems are defined as part of the detailed design of the layered architecture system.
- **Detailed design:** The detailed design model, which contains several design patterns as well as a library of class inheritance hierarchy used by various component systems, is built based on the layered architecture system. The library includes concrete and abstract classes with variation points. Most attributes and methods of each class are also defined in this phase.
- **Implementation:** The application and component systems will be implemented as defined in the detailed design of the layered architecture system.
- **Testing:** Each application system and component system is tested individually and also tested as part of the layered architecture system as a whole.

5.0 APPLICATION FRAMEWORK DEVELOP-MENT STRATEGY

The manufacturing application framework to be developed is not just an integrated set of software components on manufacturing domain, but an application ready to be assembled, completed with some needed tools, training, user guide and instructions. The application framework is divided into two layers, which are the application layer and the component layer. Fig. 4 shows a part of the application framework in the Production Management subdomain.

Both layers in the framework are interoperating with each other in the same layer with some dependencies of the application layer on the component layer. In the application layer, the application systems that can be constructed from the framework are known beforehand. In the component layer, the component systems that are needed to construct a particular application system are also known beforehand. As illustrated in Fig. 4, the types of application systems that can be constructed from the framework are the *Forecast* application systems and

Production Schedule application systems. To develop a specific *Forecast* application system, the two component systems named *Historical Data Management* and *PP_item Forecast Management* will be imported automatically by the system for customization by the software developer. The same idea applies to develop a specific *Production Schedule* application system.

Fig. 5 shows a part of the application framework in the SQC subdomain. Two types of application systems that can be constructed from the framework are the *Attribute Chart* application systems and *Process Capability* application systems. A software developer can customize and develop a specific *Attribute Chart* application system by importing the component systems such as *SPC Data Management*, *SPC Process Management*, *Control Chart Management* and *Basic Statistical Management*. On the other hand, to develop a specific *Process Capability* application system, *Basic Statistic Management* and *Process Capability Management* component systems need to be imported.

In the component layer, the component systems are designed to be customizable, adaptable and configurable in order to enable the development of a specific application system from the application framework much easier. Documents for the application and component systems interface as well as their variation points are also provided in the application framework in order to assist the reuser to fully utilize the features of the framework.

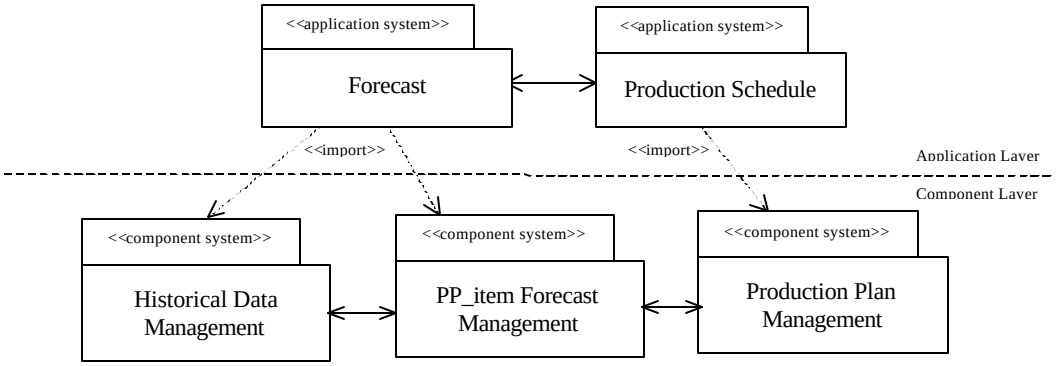

Fig. 4: A part of Production Management in the manufacturing application framework.

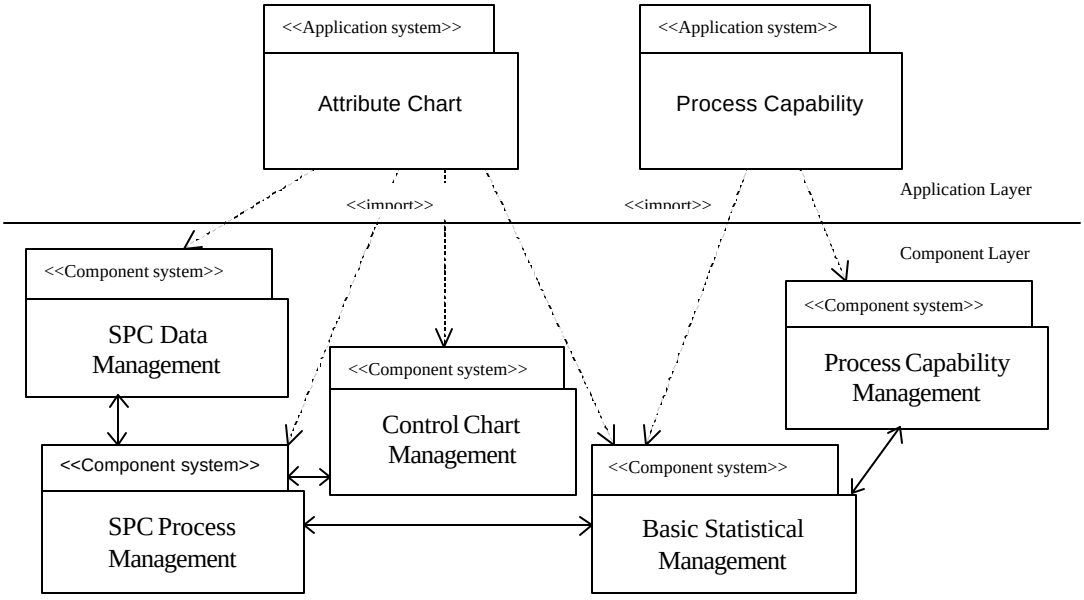

Fig. 5: A part of SQC in manufacturing application framework

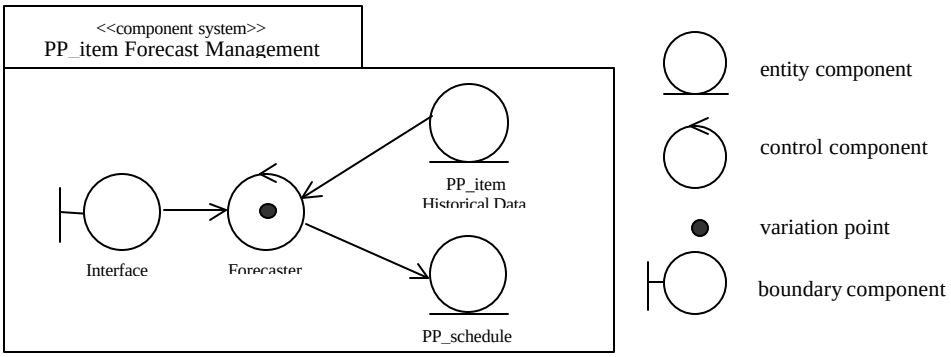

Fig. 6: The BCE diagram of *PP_item Forecast* Management component system

For example, the *PP_item Forecast Management* component system can be configured into different forecasting methods in order to develop a specific *Forecast* application system. Following the notation of Jacobson et al [12] for defining design pattern, the Boundary-Control-Entity (BCE) diagram of a part of the *PP_item Forecast Management* component system is shown in Fig. 6. There is a control component called *Forecaster*, which is interacting with two entity components called *PP_item Historical Data* and *PP_schedule*. The *PP_item Historical Data* component supplies the historical data of the sales of production planning item to the *Forecaster* control component in order to forecast the future sales orders and record in the production planning item schedule, called *PP_schedule*.

The forecasting method of the generic component can be adapted to different types of methods, such as moving average, exponential smoothing, Box-Jenkins, etc. according to the reuser's needs through the variation point of the *Forecaster* control component in order to develop a specific *Forecast* application system.

The design pattern used to implement the application framework is based on the Template Method pattern as defined by Larman [14]. This pattern is the core of the system. The idea is to define a template method in a superclass that defines the skeleton of an algorithm, with its varying and unvarying parts. The Template Method invokes other methods. These methods are concrete methods and abstract methods, which may be overridden in a subclass. Thus, subclasses can override the varying methods in order to add their own unique behavior at various points of variability.

Fig. 7 shows the design pattern of the *PP_item Forecast Management* component system. The method named *forecast* is a template method of the superclass called *Forecaster*. This template method consists of two concrete methods, named *historyDataRetrieval* and *scheduleUpdate*, and one abstract method, named *forecastMethod*. At first the template method calls *historyDataRetrieval* to retrieve the history data of a certain item. The method *historyDataRetrieval* takes two arguments, which are the identification number (*item_ID*) of the item to be forecasted and the duration of the history data needed to be retrieved (*period*). This method will return an object of *PP_item_Historical_Data*, which contains the history data in a certain period of the item.

The second method called by the template method is *forecastMethod*. The method *forecastMethod* is an abstract method in the *Forecaster* abstract class, which can be overridden in each of the subclasses as redefined methods. For example, in the *MovingAverageForecaster* subclass, *forecastMethod* is overridden to perform the moving average forecasting method; and in the *ExponentialSmoothingForecaster* subclass, *forecastMethod* is overridden to perform the exponential smoothing forecasting method. The method *forecastMethod* takes an object of *PP_item_Historical_Data* as an argument and returns an object of *Forecast_result*, which contains the result of the forecasting method. The last method called by the template method is *scheduleUpdate* for updating the production schedule (*PP_schedule*) of the item. It takes two arguments, which are *PP_schedule* and the object that contains the result of the forecasting (*Forecast_result*).

6.0 CONCLUSION

This paper has described a framework-based approach to the development of application systems by reusing framework design in the domain of manufacturing. The focus of the paper is on the manufacturing application framework development with the aim of providing the same ideas for future reuse efforts. The purpose of the project is to provide an integrated application framework, operating environment, common databases and interface standards for developing manufacturing software applications. The manufacturing application framework aims to enable large-scale software reuse to reduce cost and time of developing specific manufacturing application systems.

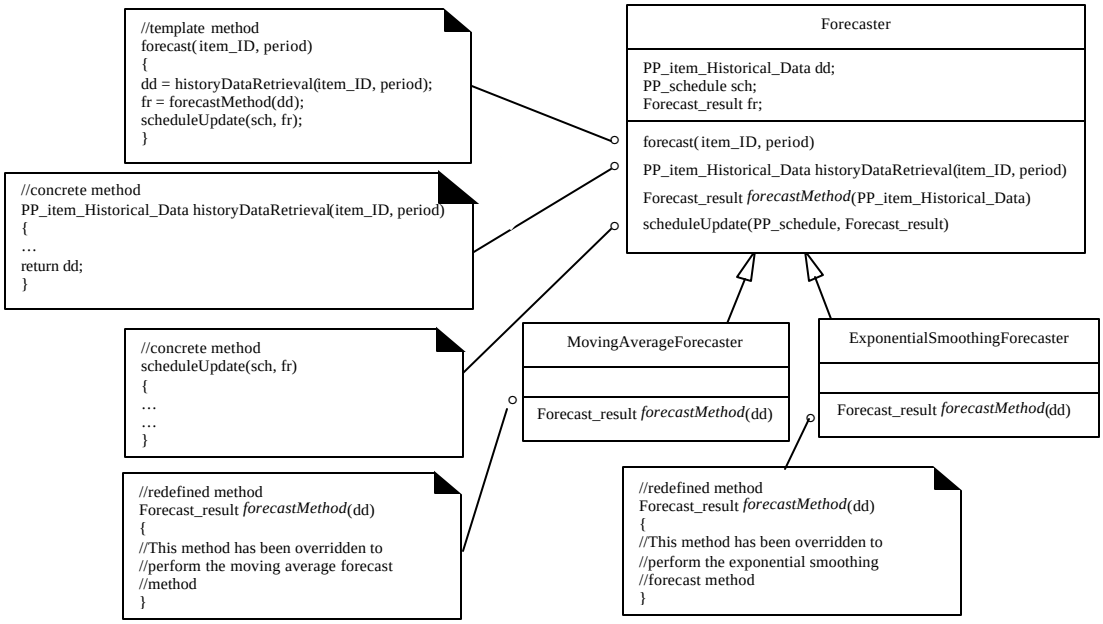

Fig. 7: Design pattern of the *PP_item Forecast* Management component system

The integration of the application framework with OOT provides a unique solution for the development of an object-oriented application on the manufacturing domain, which is wide-range, dynamic and complex. The object-oriented application framework will serve as a standard manufacturing domain design infrastructure for the development of specific manufacturing application systems.

REFERENCES

[1] Emmanuel Henry and Benoit Faller, "Large-Scale Industrial Reuse to Reduce Cost and Cycle Time", *IEEE Software*, September 1995, pp. 48-53.

[2] James B. Dilworth, *Operation Management, Design, Planning and Control for Manufacturing and Services*, McGraw-Hill, Inc., 1992.

[3] William S. Messina, *Statistical Quality Control for Manufacturing Managers*, John Wiley & Sons, Inc., 1987.

[4] Douglas C. Schmidt, "Using Design to Develop Reusable Object-Oriented Software", *http://www.wustl.edu/~schmidt/OOWG-statement.html*.

[5] G. Froehlich, H. J. Hoover, L. Liu, and P. Sorenson, "Designing Object-Oriented Frame-works". In, *S. Zamir, editor, Handbook of Object-Oriented Technology*. CRC Press, Boca Raton, FL., 1997.

[6] "Leveraging Object-Oriented Technology Frame-work", h*ttp://www.ibm.com/java/education/ooleveraging /index.html*.

[7] Glenn E. Krasner and Stephen T. Pope, "A Cookbook for Using the Model View Controller User Interface Paradigm in Smalltalk-80". *Journal of Object-Orientated Programming*, August/September 1988, pp. 26-49.

[8] T. J. Biggerstaff, and C. Richter, "Reusability Framework, Assessment, and Directions". *IEEE Software 4*, March 1987, pp. 41-49.

[9] Barbara Hayes-Roth, Karl Pfleger, Philippe Lalanda, Philippe Morignot, and Marko Balabanovic, "A Domain-Specific Software Architecture for Adaptive Intelligent Systems", *IEEE Transactions on Software Engineering*, Vol. 21, No. 4, April 1995, pp. 288-295.

[10] Wing Lam, "The Development of Very High-Level Components for Software Engineering", In, *IASTED International Conference Software Engineering*, November 1997, pp. 49-52.

[11] Roberto Bellinzona, Politecnico di Milano and Maria Grazia Fugini, Reusing Specifications in OO Applications, *IEEE Software*, March 1995, pp. 65-75.

[12] Ivar Jacobson, Martin Griss and Patrick Jonsson, *Software Reuse Architecture, Process and Organization for Business Success*, Addison-Wesley, 1997.

[13] Sally M. Chan and Terence L. Lammers, "The OODE Method – Designing, Building and Delivering Domain Frameworks", *Boeing Commercial Airplane Group – Information Systems*, June 1997.

[14] Craig Larman, *Applying UML and Pattern*, Prentice Hall, Inc., 1997.

BIOGRAPHY

Sai Peck Lee obtained her Master of Computer Science from University of Malaya in 1990, her D.E.A of Computer Science from University of Paris VI Pierre et Marie Curie in 1991 and her PhD of Computer Science from University of Paris I Panthéon-Sorbonne in 1994. She is an Associate Professor in the Faculty of Computer Science and Information Technology, University of Malaya, Malaysia. Her research interests include Software Engineering, Object-Oriented Methodologies, Software Reuse, Information System and Database Engineering.

Siew Khim Thin is currently pursuing his Master degree in the Faculty of Computer Science and Information Technology, University of Malaya, Malaysia.

Hong Song Liu is currently pursuing his Master degree in the Faculty of Computer Science and Information Technology, University of Malaya, Malaysia.