A Paradigm for the Integration of Real-time Monitoring and MRP II

by

Mr. Kwok-Cheung Lee

Department of Manufacturing Engineering

The Hong Kong Polytechnic University

A thesis submitted to

The Hong Kong Polytechnic University

in accordance with the regulation

for the degree of Master of Philosophy

February 1999

Abstract of thesis entitled 'A Paradigm for the Integration of Real-time Monitoring and MRP II'

submitted by Mr. Kwok-Cheung Lee, Department of Manufacturing Engineering

for the degree of Master of Philosophy

at The Hong Kong Polytechnic University in February 1999.

Abstract

The objective of this research is to design a methodology for integrating Manufacturing Resource Planning (MRP II) systems and Supervisory Control and Data Acquisition (SCADA) systems. The aim is to enhance data integrity, data consistency and data accuracy of the execution phase of MRP II applications through the use of real-time data provided by real-time data capturing systems.

Although there are numerous methods established to enable real-time data in MRP II applications, these methods are application dependent and none of them can be considered as a general methodology. This is further hindered by the unlimited hardware and software standards in the past; it is difficult to provide a solution in an infinite, diverse, dynamic and unpredictable environment. With the advancement of software engineering technology, a universal standard (i.e. OLE for Process Control), for data interchange between field devices and software was available in the late 90's. This evolution created a finite environment for this research that has not been possible in the past.

Early studies indicate that integration is not only difficult to construct properly for the first time, but more significantly, that it is often difficult to specify correctly the first time. To overcome these pitfalls, the proposed integration methodology is based on both theoretic and pragmatic premises. The integration methodology contains three key elements: a data model, a collection of modeling heuristics, and a method of implementing the designed model. The model is a data representation of real world objects and operations. It employs the Hierarchical Object-Oriented Design (HOOD) tree as the core data structure. The HOOD tree is devised from the HOOD method that is a proven technology for designing real-time mission-critical application in both the academic and the industry field. The complexity of the model is mirrored in the size and structure of real-world entities. Hence, the proposed methodology provides a consistent manner to form the data model. Modeling heuristics are apparent to transform the HOOD tree from a physical model to a logical model. Various optimization strategies that are based on the rule of inheritance and polymorphism are also introduced in this research for the effective use of a data model.

To validate the proposed methodology, case studies were carried out with the help of a collaborated company as a Teaching Company Scheme project. The results indicated that the proposed methodology is a feasible solution to integrate MRP II systems and SCADA systems. Two industrial projects are successfully implemented based on the integration methodology developed in this research.

Acknowledgements

The author expresses his gratitude to Dr. W.H. Ip of the Hong Kong Polytechnic University, and to Dr. K.L. Yung of the Hong Kong Polytechnic University, who were directly involved in and supported this project.

Special thanks are due to Mr. Simon Lee of GRD Engineering (H.K.) Ltd. for his many useful technical discussions and recommendations.

Table of Contents

List of Figures

Glossary

APS	Advanced Planning and Scheduling
AS/RS	Automated Storage/Retrivel System
CASE	Computer Aided Software Engineering
CIM	Computer Integrated Manufacturing
CORBA	Common Object Request Broker Architecture
DDE	Dynamic Data Exchange
DFD	Data Flow Diagram
DOOD	Distributed Object-Oriented Database
EDI	Electronic Data Interchange
ERP	Enterprise Resource Planning
ESA	Eeupoean Space Agency
HOOD	Hierarchical Object-Oriented Database
JIT	Just In Time
LTR	Langage Temps Reele
MPS	Master Production Schedule
MRP	Material Requirement Planning
MRP II	Manufacturing Resource Planning
OLE	Object Linking and Embedding
OO	Object-Oriented
OOAD	Object-Oriented Analysis and Design
OOD	Object-oriented Design
OPC	OLE for Process Control
PLC	Programmable Logic Controller

POC	Plant Operation Control
RTU	Remote Terminal Unit
SA&D	Structured Analysis and Design
SCADA	Supervisory Control and Data Acquisition
STD	State Transition Diagram

1. Introduction

1.1 The Problem

To complete effectively in the global marketplace, manufacturers are being faced with challenges including:

- Lowering operating costs
- Integrating with the supply chain
- Reducing product time-to-market
- Improving product quality
- Decreasing cycle times
- Increasing productivity
- Improving customer service
- Improving regulatory compliance

In order to meet today's manufacturing challenges, decision makers throughout the enterprise must have accurate and timely production information to allow them to make effective decisions. All too often decision support systems do no provide timely information. Most of existing applications (e.g. Material Requirement Planning (MRP) software, Manufacturing Resource Planning (MRP II) software, Enterprise Resource Planning (ERP) software, etc.) are batch driven, giving reports on the status of production as of the last shift, day or even week. Obviously, this is not an appropriate way to manage a manufacturing enterprise. The need of a real-time manufacturing management system is clear.

Furthermore, real-time data provides a critical element of supply chain integration. MRP II software, Electronic Data Interchange (EDI) standards and Advanced Planning and Scheduling (APS) software have gone a long way toward linking the supply chain. Manufacturing enterprises can plan their business effectively with the above software systems. However, they are weak in executing their plan without the help of real-time data.

1.2 Identification of the Problem Area

Today's manufactures face the task to integrate real-time shop floor data into their business systems in order to have a better control to their business. The task of integrating shop floor data into their business systems is made difficult, since integration strategies, standards and tools are hardly available in the market. Moreover, the task itself is not an all-in-one product. Investment in knowledge, time, cost and tools is essential to achieve the integration goal.

From a technical point of view, basic components of a real-time manufacturing management system are available from various manufacturers. However, those components are working in their own domain and they are isolated from each other for more than ten years. The problem can be identified into three main areas:

First, the original design and implementation of MRP II system does not facilitate the capture of real-time data from shop floor automatically. Instead, MRP II system is

targeted on the planning stage of the manufacturing cycle. The execution stage of the manufacturing cycle is usually omitted in the scope of MRP II applications.

Secondly, data capture systems such as Supervisory Control and Data Acquisition (SCADA) system has made data available electronically, but they are not designed for manufacturing management. SCADA system is a comprehensive data capturing application. It is a computerized system for monitoring and for controlling shop floor equipment. SCADA system is widely used in process manufacturing. However, SCADA system concentrates on process control instead of shop floor management.

Thirdly, there are different types of shop floor devices utilized to communication with other systems. This technical aspect is the main barrier to implement real time data in the manufacturing management application. In the past, manufacturers have to develop a special driver for each shop floor device to communicate with their enterprise information system. This approach is ineffective because every solution will be a proprietary system, system components cannot be reused and the solution cannot be generalized.

1.3 Industrial Survey and Results

A survey on the need of a real-time manufacturing system was performed in this research. The survey was prepared and organized by the Hong Kong Polytechnic University and the collaborated company – GRD Engineering (H.K.) Ltd. under the Teaching Company Scheme project in which I am the research assistant. The target

of this survey is to identify the importance of real-time data from the view of Hong Kong's manufacturers.

The survey was carried out in the 4th quarter of 1997. One thousand manufacturers in Hong Kong were randomly selected from the customer database of the GRD Engineering (H.K.) Limited. The mode of survey was by mail and followed up by telephone interviews. A total of 238 out of 1000 questionnaires were returned. The targets of this survey are the executives and senior managers of Hong Kong's medium-size manufacturers. The questionnaire can be seen in Appendix 11.1.

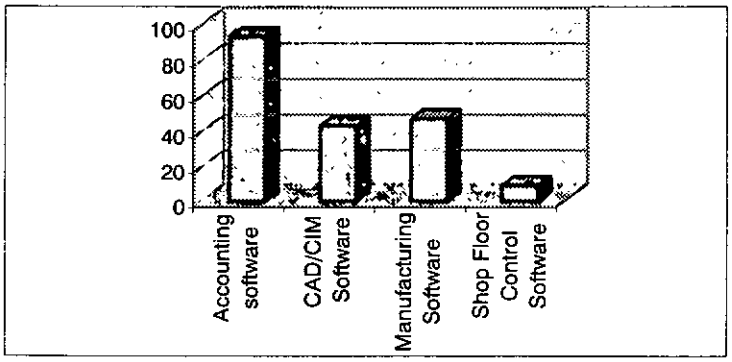


Figure 1: Distribution of software applications.

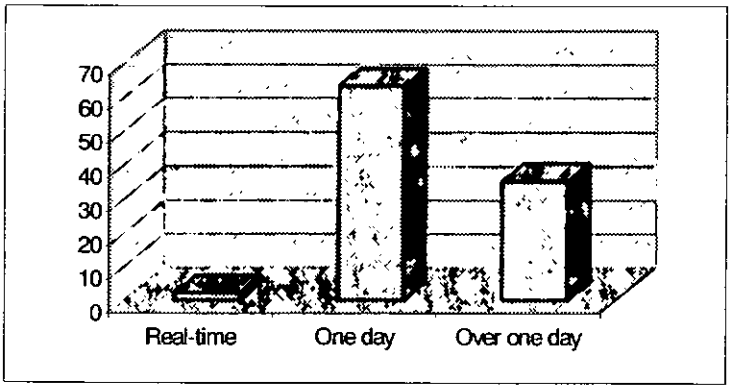


Figure 2: Average system response time

From the analysis, we can see that half of the manufacturers have implemented manufacturing applications and 9% of them have data capturing systems (see figure 1), but only 2% of them can obtain real-time data through their system (see figure 2). Over one third of them have to take more than one day to identify their production status. It shows that manufacturers are aware of the importance of planning, but they are weak in keeping track of their production plan. This may owing to the reason of around 70% of manufactures have their factories in main land China and communication across the border is the major problem. The result of survey shows that manufacturers are well aware the need of accuracy and timing of data. They are prepared to effectively make use of production information.

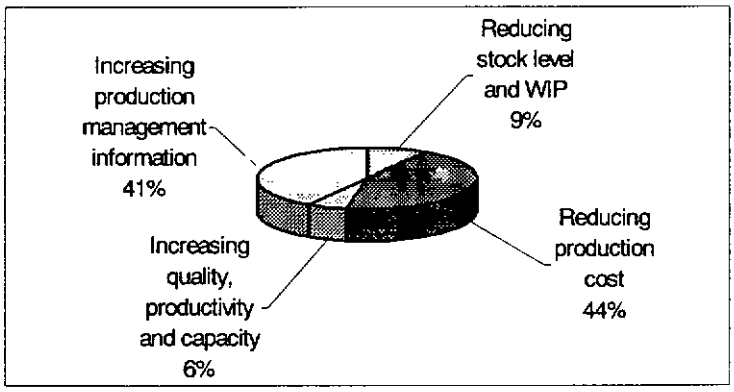


Figure 3: The most important target of the organization

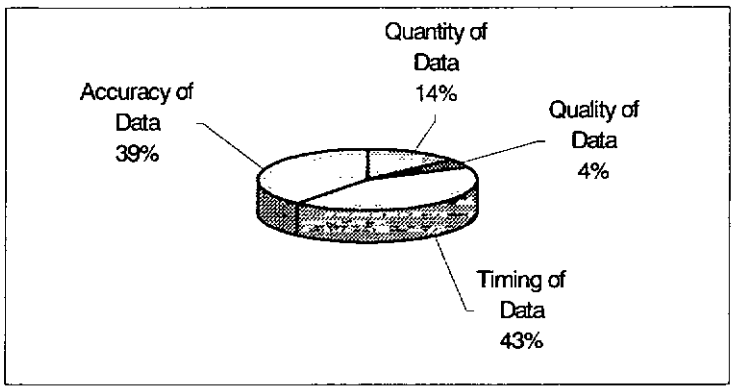


Figure 4: The importance of data

From figure 3, we know that manufacturers want to have more production data to manage their business in order to reduce the production cost. Their need of accurate and up-to-date production data is shown in figure 4. However, from the result of figure 2, it shows that most of the manufacturers are not achieving what they want. According to the answers of the open question in the questionnaire and through the telephone interview, we identified that the lack of real-time production data in the management level is the main reason behind the problem. Problems are not being able to report to the management in time and this aspect delay the management to make the correct decision. This research targets to close the gap between the production level and the management level through the integration of MRP II systems and real-time data.

1.4 Overview of the Thesis

This thesis is organized in nine chapters. Chapter one gives an introduction to the problem and the problem area is identified with an analysis on the survey results. Literature survey on previous researches and nowadays technologies are provided in chapter two. The proposed methodology for integrating MRP II systems with real-time data is presented in chapter three. Chapter four describes the approach to implement the integration methodology. Two case studies are presented in chapter five. Research results are stated in chapter six. Chapter seven discusses the advantages and limitations of the proposed methodology. Chapter eight concludes works and results in this research. A statement of originality and contribution to knowledge is given in chapter nine. Finally, bibliography and appendixes are attached at the end of this report.

2. Literature Survey

Five major areas have been investigated in the literature review. The five areas are Manufacturing Resource Planning (MRP II), Supervisory Control and Data Acquisition (SCADA), existing integration solutions, Hierarchical Object-oriented Design (HOOD) and OLE for Process Control (OPC). This research first identifies the nature, needs and weakness of both the MRP II system and the SCADA system. Then, existing integration solutions are reviewed and evaluated. Finally, HOOD method is introduced as the core design concept and OPC is applied as a proven implementation method.

The review on MRP II aims to identify problems in the implementation stage and the execution stage of a manufacturing software system. The discussion on SCADA systems stress on their impacts on changing the mode of software automation. Existing integration solutions are then described and their inadequacies are identified. The review on HOOD illustrates the hierarchical design concept and usage of object-oriented technology in real-time applications. Literatures on OPC demonstrate that it is a feasible technology to implement the integration.

2.1 Manufacturing Resource Planning (MRP II)

Material Requirement Planning (MRP) is a calculation technique for planning purchase orders and manufacturing orders (Takana, 1992). MRP II is an extension of MRP. By linking other business management applications to MRP, MRP II becomes a method for effective planning of all resources of a manufacturing company. MRP II is a systematic approach to plan production in complex multistage manufacturing systems. Most of the manufacturing application software is based on the concept of MRP II as the central application. MRP II is the dominant application software system for today's manufacturing management (Turbide, 1995).

The major inputs of MRP II are the master production schedule (MPS), the product structure records and the inventory status records. The MPS is a statement of the orders for finished goods. It indicates the quantity of each finished good that needs to be produced by a time period. The MPS usually involves the purchase order released from the customer and the work order to be issued to the shop floor. The product structure records or the bill of material (BOM) is a list of components and material that required to produce a finished good. It contains information on the relationships of parts, components and assemblies. It also shows the quantity of each component and the sequence to assemble a finished good. The inventory status records contain the data of on-hand inventory, inventory location and on-order inventory.

The outputs of an MRP system includes purchases orders for raw material, work orders for shop floors, reschedule notices for job review and capacity requirements for labor arrangement. Figure 5 illustrates a typical MRP II system.

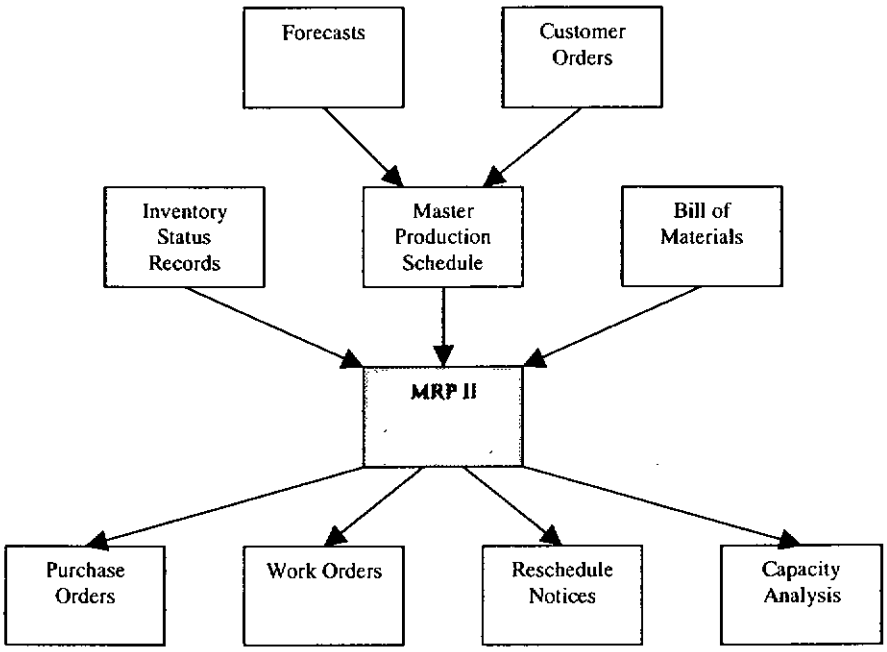


Figure 5: A typical MRP II system.

The study of MRP II has received a lot of attention in the field of manufacturing management. Many literatures (Woodgate, 1989) (Villa, 1990) (Poon, 1991) (Muller, 1990) dealt with defining MRP II and integrating MRP II with new functions and other systems. In recent years, the execution and feedback phase has been described as a weakness in many MRP II systems (Murthy, 1991). The following section describes some researches that identified the problems of the execution and feedback phase in MRP II.

Sampson (Sampson, 1985) concentrates on integrating MRP with flexible manufacturing systems such as material handling systems. The main idea is to computerize the shop floor area and storage area to gain better control such as Automatic Storage/Retrieval System (AS/RS). Although his paper is based on material

handling system only, the idea of computer control and feedback in shop floor level has been raised.

Piciacchia (Piciacchia, 1989) discusses the importance of shop floor control and defines the Plant Operations Control (POC) function that integrates all shop-floor activities and maintains control over the execution of the Just In Time (JIT) manufacturing schedule in a Computer Integrated Manufacturing (CIM) Plant. The idea of POC is similar to SCADA systems in the 90's. However, SCADA is a more advanced and well-established expert system for real-time control and monitoring.

Browne (Browne, 1995) focuses on shop floor control for discrete parts cellular manufacturing system. He proposes a functional architecture for managing the flow of work through the system in quasi real time. His work is based on the conventional structured design methodology.

The lack of well-defined data structure and the piecemeal development approach of traditional MRP II systems had posed a number of problems (Correll, 1995). We noted the following shortcomings inherent in the traditional approach:

- Data does not represent an abstraction of reality
- Pending data affects estimation
- Real-time data is not available
- Production data is not taken into account in time

The manufacturing technology, modes of production and hardware devices have advanced dramatically over the past several decades, the network and computer

systems have got faster and more standardized (Loose, 1995). For example, the speed of the communication interface of field devices has advanced from 2400bps serial link to 100Mbps fast Ethernet. Unfortunately, the manufacturing software has not kept up with the changes in hardware. This problem is usually related to the quality of software. Inflexible software makes the whole system very difficult or even impossible to modify. Unsupported software and hardware are a great barrier for the growing of MRP II applications. Moreover, task specific MRP type applications do not support linkage to other enterprise information systems for data sharing. Therefore, new technology for developing flexible and scalable manufacturing software is required to support the modern manufacturing enterprises.

2.2 Supervisory Control and Data Acquisition (SCADA)

Collecting shop floor data and entering it into a computer manually is an error-prone, time-consuming and inefficient process. More importantly, this approach cannot provide up-to-date information and it is not realistic for time critical applications. For the above reasons, data capturing systems such as SCADA systems became common since the late 60's (Nicoloro, 1994).

The term SCADA was defined by the Bonneville Power Administration, Portland, Oregon, as comprehensive definition for a combination of technologies. Supervisory control is the process of looking at current activity from a distance and passing operating directives to the remotely located controllers. Data acquisition is the process of obtaining information from remotely located devices or systems. In its basic form this implies a one-way transfer of data from remotes to central hosts. However the state-of-the-art devices now allow transfers in the opposite direction and downloading of configuration parameters (Szelke, 1994). Earlier examples of control systems based on reed relays and tone telemetry such as railways control systems. General examples of modern SCADA systems include breweries, sewage treatment plants, welding plants, hydroelectric power stations, automotive transfer systems and mains water treatment plants, etc. (Rhodes, 1994).

In the early 90's, SCADA systems were widely used for real-time data capturing. A SCADA system is a high level framework for building real-time control and monitoring application (Mazumdar, 1991). A typical SCADA system consists of shop floor equipment, communication protocols, data networks, logistic control

software and Man-Machine Interface (MMI). A sample SCADA system is demonstrated in figure 6. Since a SCADA system operates in real-time, it requires some mechanisms for synchronization of its active components. An object-oriented model is proposed for handling the interaction between each component in such an active system (Jorgensen, 1994).

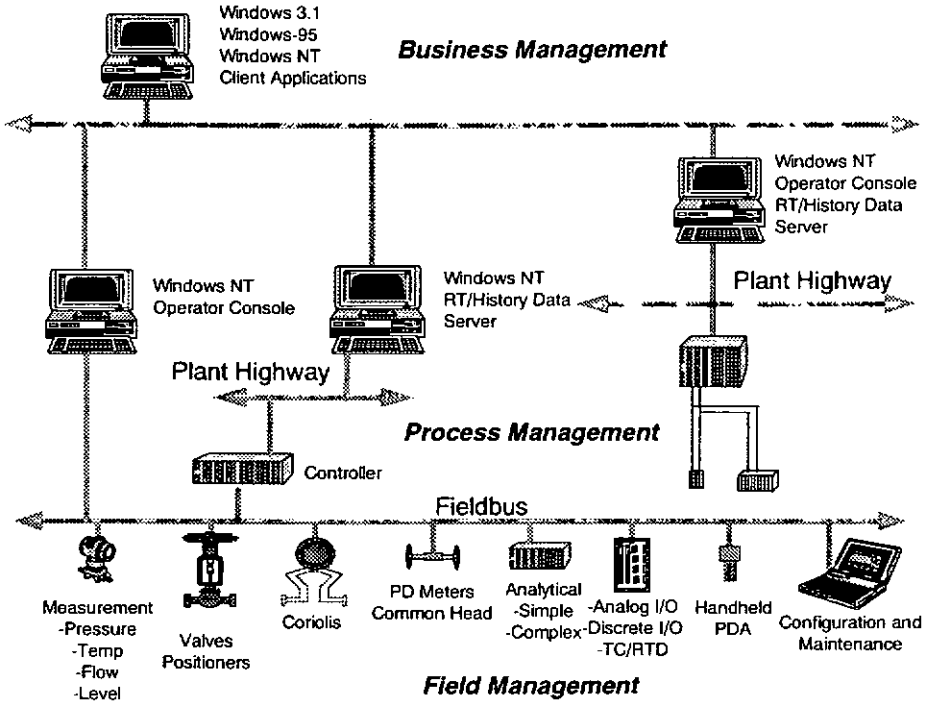


Figure 6: A typical SCADA system.

Gay (Gay, 1994) pointed out the hardware required to support a SCADA system has changed dramatically over the past several years, the network and computer systems have got faster and more open. The software unfortunately has not kept pace with the changes in the hardware. There have been more analysis tools available but the SCADA software has remained proportionally closed and proprietary. Because these systems have been built with speed and data integrity as the main concern, the ability to share data is usually omitted in the design stage. This has generally lead to only one group or department having access to highly valuable data. Since that department is charged with maintaining the speed and integrity of the SCADA system, they are usually reluctant to allow other departments access to the SCADA system.

Common deficiencies associated with existing SCADA systems were identified by Brinkler (Brinkler, 1997) as follows.

- Inflexible software made it impossible to modify.
- Aging/unsupported hardware/software.
- No electronic linkage to corporate system for data/application sharing.
- Multiple systems existed with no electronic linkage among them.
- No efficient way to store/retrieve data and perform ad hoc queries.
- Remote Terminal Units (RTUs) and Programmable Logic Controller (PLCs) (Peshek, 1993) were at maximum capacity and were not capable of storing/processing information.
- Limited communication diagnostic capabilities.

2.3 Existing Integration Solutions

Dynamic changes in manufacturing shop floors introduce an infinite development environment. Changing hardware and their configurations likely will result in modifying the software interface as well. The success of implementing a real-time MRP II system has been limited usually because of the lack of flexibility in its software layer to adopt the changes. This is a common pitfall of all existing solutions (Grimble, 1996).

Russell (Russell, 1990) stresses real time data is available at the process level, but it is difficult to access and associate with schedules and management functions such as Enterprise Resource Planning (ERP) systems (Fitzgerald, 1994) or MRP II systems. However such data contains vital information as to the actual machine performance on the shop floor at any moment in time, and when processed correctly can be used to monitor and indicate production problems in time to make calm adjustments to schedules. Russell (Russell, 1990) address how a PLC is used to collect shop floor data and is interfaced to a low cost super-micro computer for production monitoring and control.

Common methods to integrate SCADA systems and MRP II systems are as follows and they are illustrated from figure 7 to figure 13:

- Build a proprietary system.
- Use Electronic Data Interchange (EDI) for information interchange.
- Use Application Program Interface (API) calls provided by MRP II systems.
- Use common communication protocols, e.g. OLE 2.0, DDE, etc.

- Access MRP II's database directly.
- Use transaction files to update information.
- Use object technology, e.g. global objects, message passing, etc.

A proprietary system is a closed system. Data cannot be shared or to share with the built system.

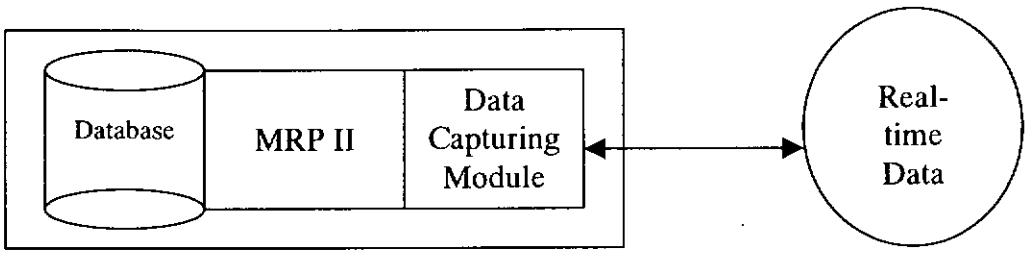


Figure 7: System Integration – A proprietary system.

EDI is a common method to exchange data between two devices. However, the data format of each device must be compatible to the other.

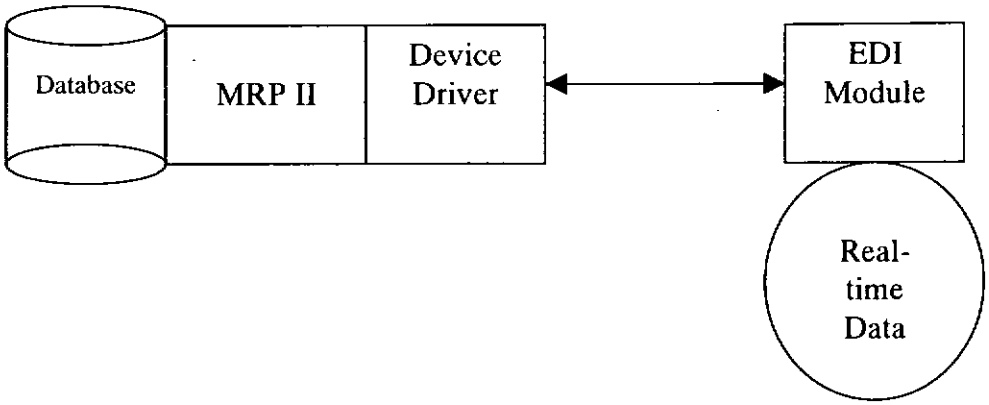


Figure 8: System Integration – Electronic Data Interchange (EDI).

API is a code level implementation of EDI. The use of API is limited by the compatibility of programming languages and development platforms.

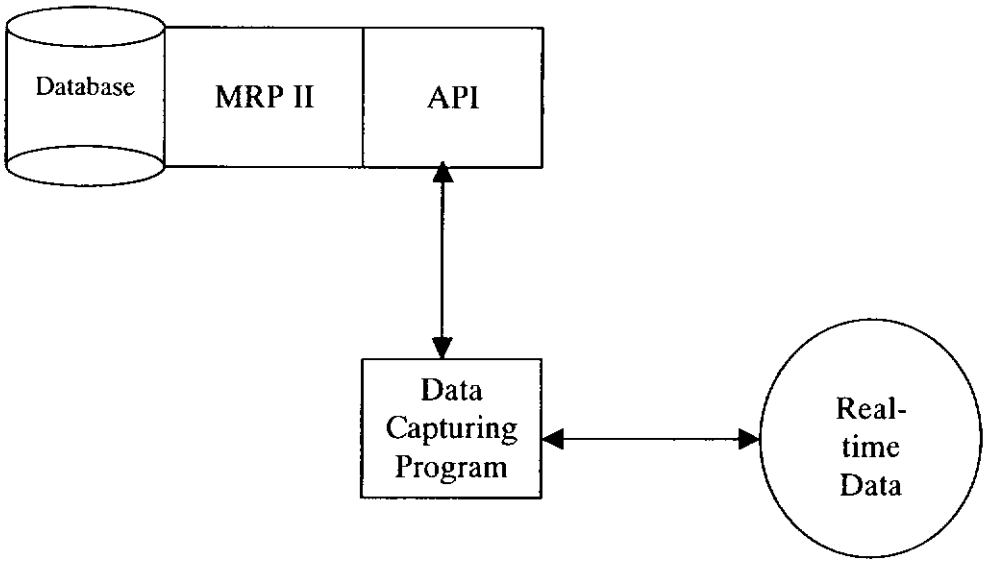


Figure 9: System Integration – Application Program Interface (API).

DDE communication is widely adopted in desktop computer. Unfortunately, the speed of communication is ineffective.

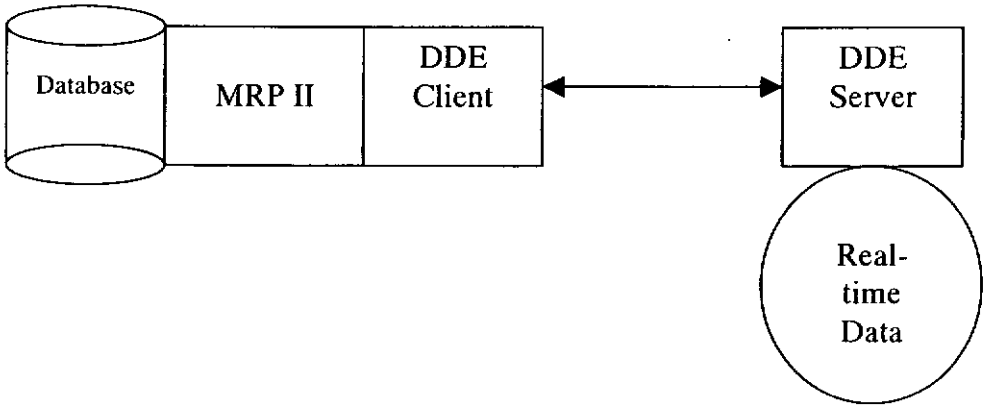


Figure 10: System Integration – Dynamic Data Exchange (DDE).

Directly write to MRP II application's database may introduce the problem of security, data integrity and maintainability.

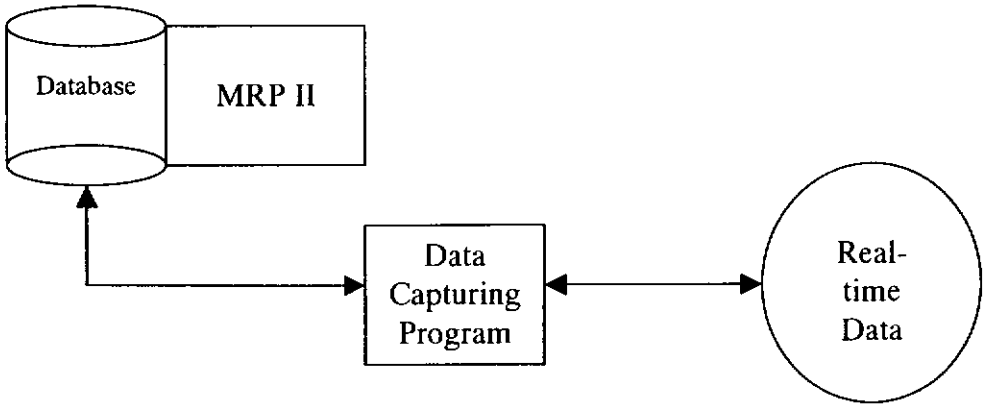


Figure 11: System Integration – Direct access to database.

The use of transaction files to update MRP II database is a batch processing. This method cannot process real-time data.

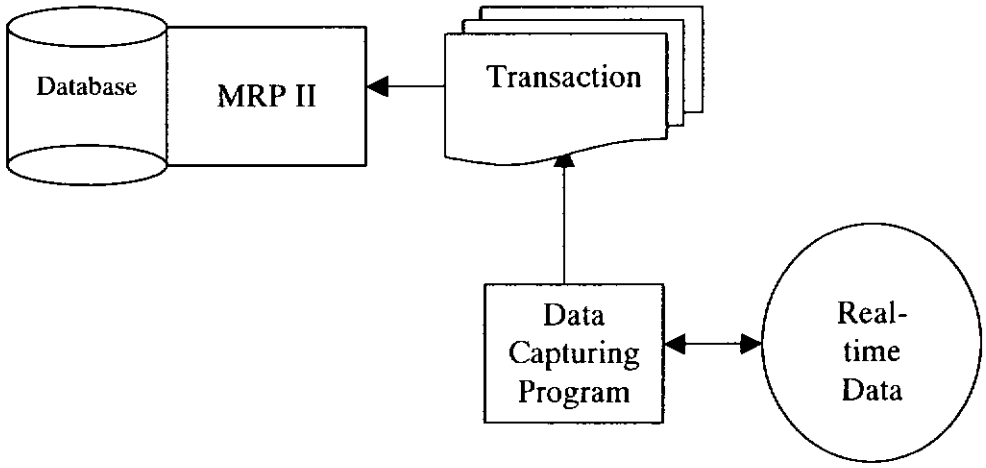


Figure 12: System Integration – Transaction File.

Object technology is a feasible solution to integrate MRP II systems with real-time data. However, object definition for each component has not been standardized.

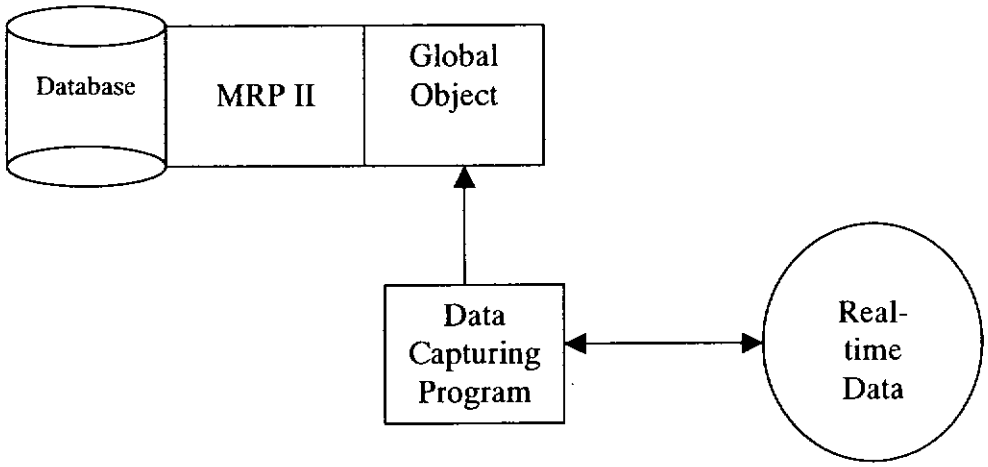


Figure 13: System Integration – Global Object.

The above methods, however, are designed for integrating particular systems only. They have a lack of flexibility and fail to account for the requirements of dynamic changes in manufacturing shop floors and information technologies (Dongarra, 1996). Thus it is necessary to introduce and develop a methodology to formalize the integration of SCADA systems and MRP II systems.

2.4 Hierarchical object-oriented design (HOOD)

The object-oriented approach is an evolving paradigm in the computer industry. Many object-oriented methods have been proposed over the years (Meyer, 1988) (Booch, 1994) (Poo, 1994) (Sodhi, 1996). The object-oriented method represents a model of a system that is based on real world entities. Objects are the basic elements in the object-oriented paradigm and a system is defined in terms of objects. An object represents a real-world entity, such as a table, car, house, etc. Each object is completed and self-contained with well-defined attributes and operations. In traditional software development approaches, operations determine the structure of the system. However, in an object-oriented system, operations and data are combined together to form objects. Structure of the system is determined by a set of objects instead of operations (West, 1996). Figure 14 illustrates the typical composition of an object, its attributes and operations

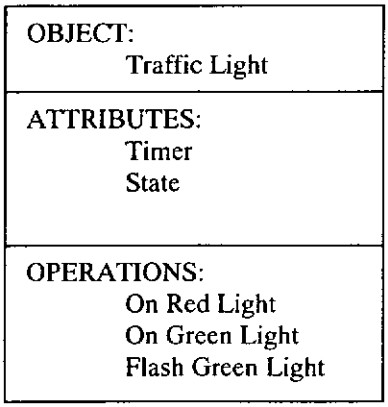


Figure 14: A typical object model.

Although object-oriented development has caught many researchers attention and various object models are proposed, only a little of them has focused on the real-time domain. There remains a certain reluctance (Haban, 1990) to apply the object-oriented approach in real-time application in conjunction with a manufacturing system. In the real-time domain, it requires a mechanism for synchronization of its active components (Lee, 1993). An object-oriented model is purposed for handling the interaction between each component in such an active system (Mitchell, 1996).

There are numerous object-oriented design methods and standards in the world, where Coral was developed and used in the UK, Common Object Request Broker Architecture (CORBA) in USA, Pearl in Germany and Language Temps Reele (LTR) in France, etc (Robinson, 1992). However, these design methods were not widely accepted in other countries. The Hierarchical Object-oriented Design (HOOD) method (HOOD User Group, 1992) was recommended for the design of real-time applications and the following describes the advantages of the HOOD. The HOOD method has been adopted by several military projects in Europe, European Fighter Aircraft, nuclear power plants, electricity projects and large communication network projects (Robinson, 1992). HOOD is considered for most major mission-critical projects as an established method well supported by Computer Aided Software Engineering (CASE) tools.

HOOD was developed as an architectural design method for real-time applications. The HOOD method was developed by CISI Ingenierie and commissioned by the European Space Agency (ESA). The original version of HOOD focuses on

modularity, data abstraction, information hiding, hierarchically structured abstract machines and supports for real-time applications.

HOOD is based on software engineering principles of abstraction, encapsulation and modularity. It is basically a top-down method, particularly suitable for a project development environment. Besides, it provides useful features for maintenance, both enhancement and correction. In short, HOOD is a combination of object-oriented design (OOD) and hierarchical decomposition of abstract machines. Software engineering features of HOOD includes abstraction, information hiding, cohesion and modularity.

Abstraction facilities to deal with complexity by focusing on the important elements of the problem. At the top level of the design, the objects are at a higher level of abstraction. These objects relate to each other and provide a complete solution to the problem without a great deal of detail. The lower-level objects are not visible. The design is shown in such a way that the structure and interfaces are clear and separated from the details of the processing. Emphasis is made on the control flows and data flows at high-level design. Hence this is hierarchical decomposition.

Information hiding benefits to reduce the complexity of an object. The essence of an object is that the data is encapsulated inside the object, and are accessible only through external operations. Details of the implementation are hidden in the body of the object. A common example is a stack object that provides operations to pop and push data onto the stack while maintaining the stack itself hidden in the stack object.

Cohesion aims to aid maintenance, abstractions that are logically in the same object or package. Thus if a change is required by a change of the environment, then the effects on the software are localized. For example, if the version of hardware is changed and requires changes to the internal software, the software interface of the object may remain unchanged such that other components and systems are not affected.

2.5 OLE for Process Control (OPC)

HOOD is a design method for building real-time applications. On the other hand, OPC is the technology to implement the application. Detailed description of the OPC technology are not included in this thesis, for more information, please refer to the OPC specification (OPC Taskforce, 1996).

Before the age of OPC, to design the integration of shop floor data with MRP II systems was a difficult task, to implement such a solution can be described as impossible. This is because MRP II systems and real-time data capturing systems are designed and developed by various companies. These companies gain access to the data by independently developing device drivers and databases for their own packages. Each company has their own design methods and standards. It is quite impossible to generalize the solution. Figure 15 illustrates the basic architecture of OPC.

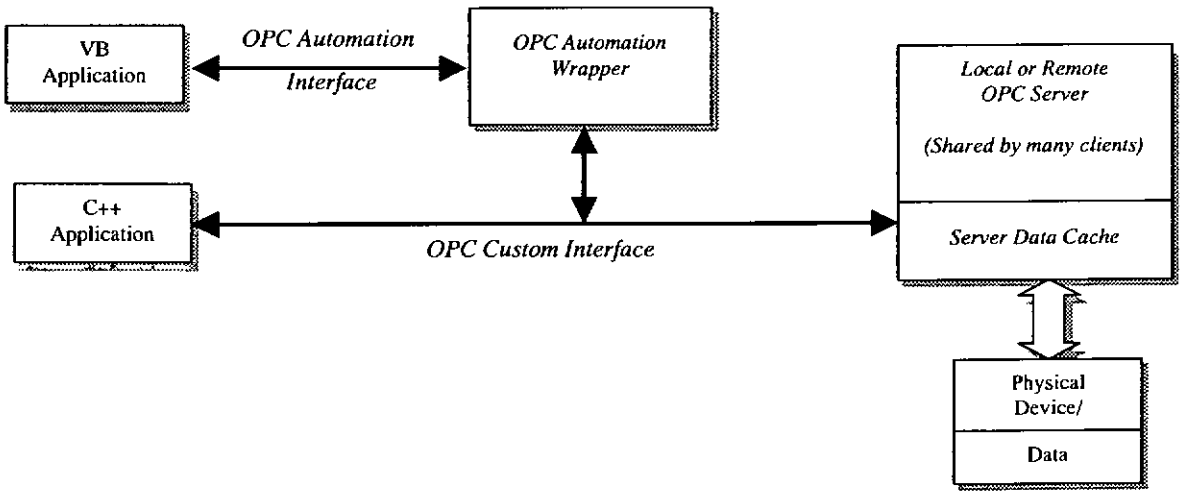


Figure 15: OPC Architecture.

OPC is an open and effective communication enabler concentrating on data access. It is designed as a common way for applications to access data from any device on the shop floor, allowing compliant applications to seamlessly access data in a manufacturing environment.

With OPC, hardware manufacturers only have to make one set of software components for customers to utilize in their applications. On the other side, software developers won't have to rewrite driver programs because of feature changes or additions in a new hardware release. Last but not least, end users will have flexibility to build their own manufacturing system with different components. System integration in a heterogeneous computing environment will become simple.

OPC consists of an OPC Client and OPC Servers provided by different vendors. The code written by the vendor determines the devices and data to which each server has access, the way in which data items are named and the details about how the server physically access that data. Figure 16 illustrates the relationships between OPC servers and OPC clients.

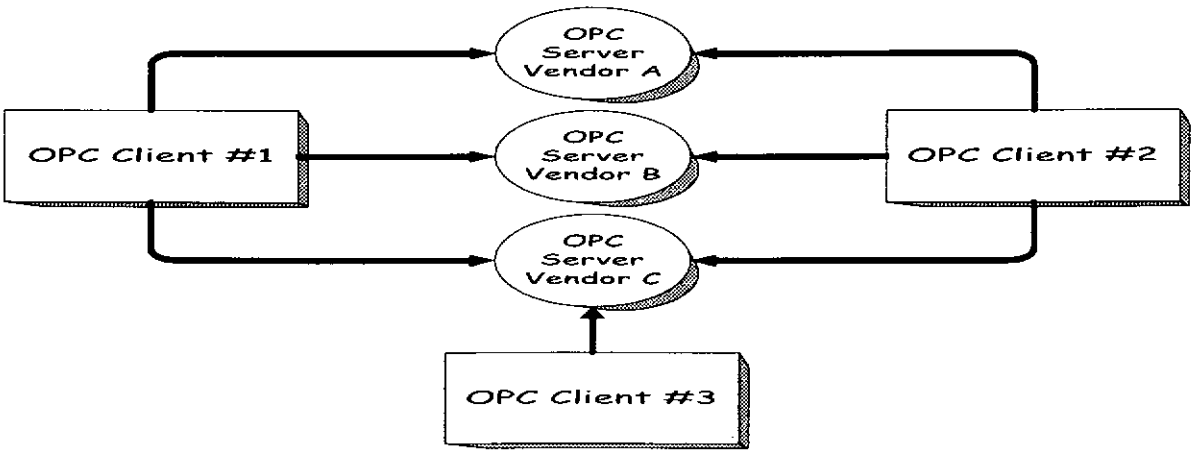


Figure 16: OPC Servers and OPC Clients.

3. The Proposed Methodology

3.1 Design Hypothesis

From our earlier discussion of the problem and literature review, we can see that there is a need for integrating MRP II systems and SCADA systems. The primary hypothesis is that real-time data is essential in today's manufacturing application. This point is also confirmed by the industrial survey. In the proposed integration methodology (see Figure 17), MRP II systems will be integrated with SCADA systems in order to provide real-time data. This research aimed to provide real-time data for MRP II systems, but not changing existing MRP II applications to real-time basis.

In order to achieve the integration, a systematic approach is necessary. There are many methodologies available in the field of system engineering that can be used to develop a model of the integration. However, the object paradigm is particularly well suited to handle the real-time domain and it was selected as the design tool after consideration (Selic, 1994). A list of considered methodologies are as follows:

- Structured Analysis and Design (SA&D)
    - Functional decomposition
    - Hierarchy presentation
    - Data flow diagrams (DFD)
    - State transition diagrams (STD)

- Data-Oriented Analysis and Design
    - Data driven approach
    - Data flow diagrams
- Events-Oriented Analysis and Design
    - Event driven approach
    - Event specification
- Module-Based Analysis and Design
    - Package specification

In the past, structured analysis and design was the most common methodology to study MRP II and other information systems (Senn, 1989). Conventional analysis and design of an information system involves the identification of data structures, information flow and task decomposition, etc. However, researches in software engineering and intuitive science (Sommerville, 1992) (Vlient, 1993) have shown that structured model may not be the best methodology for understanding a system. This is simply because a human does not think in this way.

Therefore, many people have moved to the object-oriented approach in the 90's. Object Oriented Analysis and Design (OOAD) becomes a new stream for system analysis and design. In OOAD, we identify real objects in the system as well as the relationships between objects. This is much closer to the human way of thinking than traditional analysis techniques.

The object-oriented paradigm claims to promote reuse, reduce development time, and improve analysis, design and software quality. Hence the cost and the risk that is

associated with software development is reduced. The introduction of object-oriented technology brings simplification and better understanding of requirements analysis.

The HOOD method is selected as the core design for our integration, because it suits the needs of complex and sophisticated real-time systems. Although other object-oriented methodologies support some of HOOD features to various degrees, there is a shortcoming that they were not designed for building real-time applications. Instead, most of them are general-purpose methodologies and not specializing in the real-time domain.

3.2 System Integration Methodology

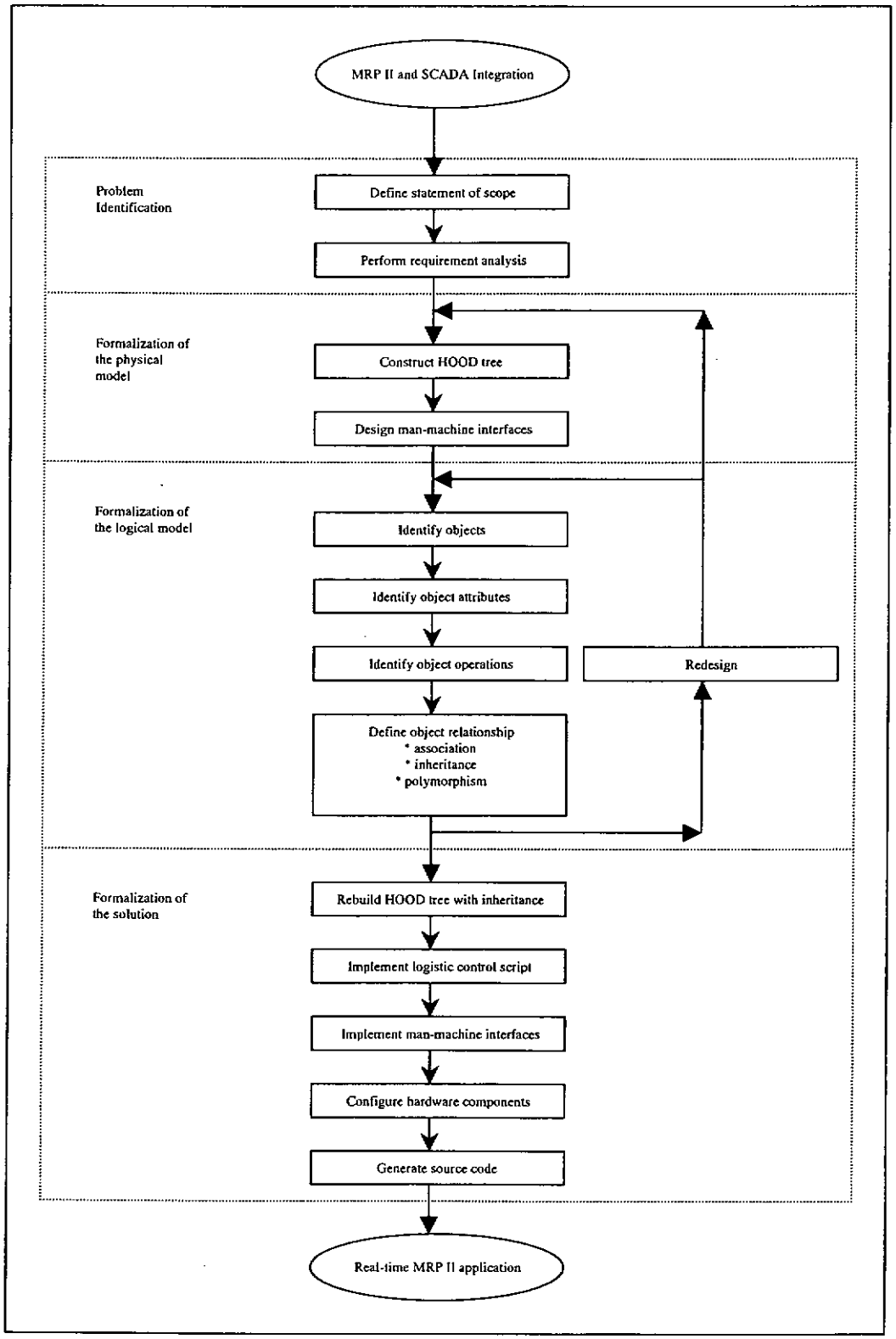


Figure 17: System Integration Methodology.

The proposed integration paradigm is illustrated in figure 17. It is a combination of top-down and bottom-up design approach. The method makes use of top-down hierarchical decomposition approach to formulate the physical model and then works out its logical model by the bottom up approach. Physical model consists of the shop floor equipment layout, equipment specification, and the communication media. Logical model contains the logistic control software, communication protocols and man-machine interfaces.

Before we move on to describe the complete integration methodology, in this section, we first have a look at the concept of a model in our methodology. The model is a key element in the proposed methodology and it distinguishes our methodology from other design methodologies.

According to the integration methodology, formalization of the model can be divided into four major steps. In step 1 (i.e. Constructing HOOD tree), a list of shop floor equipment and their layout is drawn out with the help of the HOOD tree. The HOOD tree forms the first version of the model - the physical model. Two types of objects are identified in this step, i.e. production objects and machinery objects. Production objects contain process level data required by MRP II systems. Machinery objects contain hardware level data required by SCADA systems. A physical model is represented by two HOOD trees (see section 3.3 and 3.4 for more information). The first one is the hierarchy of shop floor equipment with machinery objects as its nodes. The second one is the inventory hierarchy with production objects as its nodes.

The bottom up object-oriented approach is then used to define attributes and operations of each object defined in step 1. In Step 2 (Identifying object attributes and operations), attributes and operations for each object are then identified by considering the function of each object required supporting and manipulating in the system. Objects and their relationships are incrementally identified through the first two steps. All those information are represented by the HOOD trees.

Step 3 (Defining object associations) is a consolidation step. It examines the identified objects and produces a consolidated set of rules that would appropriately analysis objects according to their instances. The target is to identify linkage objects to establish bridges between machinery objects (real-time data) and production objects (MRP II data). Hence, this is the major step for linking real-time data to MRP II systems. For detailed description, please refer to section 3.4 – Data Model.

In step 4 (Applying inheritance and polymorphism), optimization strategy is applied to the HOOD tree by the nature of object inheritance and polymorphism. A complete combination of the physical and logical model is finalized after several refinement steps. Real-time application is finally generated according to the model created in this stage.

An important concept of the proposed integration methodology is to model the application by HOOD tree instead of building task-specific functions to accomplish the goal. The preliminary output of the integration methodology is not an executable system; indeed, it is a data model that contains all necessary knowledge to execute an application. The execution system is fully described in the next chapter. This

approach creates a model-driven application, such that the design and implementation of any system can be generalized.

3.2.1 Problem Identification

Similar to the other design models, the first stage of system design is problem identification and it contains two basic activities:

- Define the scope in terms of statements
- Perform requirement analysis

The statement of the problem is a clear and precise definition of the problem and the context of the system to design. Scope of work, constraints and available resources should be included in the statement. The purpose of requirement analysis is to verify and validate the terms of the statement. Flexibility study is an essential task to make sure the problem has been well understood, and more important, the possibility of the proposed solution.

Problem identification involves understanding the problem domain such as system specification and requirement. This is a general stage for all kinds of system development and does not involve any design paradigm. There are numerous methods to analyze a problem (Harhalakis, 1991), however, stepwise refinement is recommended to deal with its complexity. The principle of this method is to break down the problem into smaller parts down to a manageable level. Feasibility study and project planning is also included in this phase.

3.2.2 Formalization of the physical model

Formalization of the physical model is the process to encapsulate real world entities (Berrisford, 1994). The physical model provides a standard tool to formalize real-world entities. This stage contains two events:

- Construct HOOD tree
- Design man-machine interfaces

The HOOD tree is a diagram of the whole MRP II system and the SCADA system decomposed down to component level. Nodes in the HOOD tree could either be a physical component (a hardware component) or a logical component (e.g. a group of components). The bottom nodes of the HOOD tree are either physical I/O points that connect to field devices or logical I/O points for data processing. The nodes at the bottom are either physical I/O points that connect to field equipment or logical I/O points for data processing.

The constructions of HOOD tree facilities the identification of real-world objects. To form the HOOD tree, a complete shop floor layout is required. There is no need to identify active and passive objects in the application like the other object-oriented design methodology, because an object in the proposed model is the same as the physical equipment can be seen on the shop floor.

After defining the physical objects, definition of how to present those objects is required. According to user requirement specified in the problem identification stage, a set of graphical user interfaces (GUIs) could be built. The rule of thumb is that if an object has been defined in the man-machine interfaces, it usually will have its definition in the HOOD tree as well. If this is not the case, the physical layout design and the man-machine interface design might have compatibility problems in terms of system completeness. The design of man-machine interface is an option in the development cycle. This step is performed if graphical representation of the data is required in the application. The graphical representation here is considered as the interface for process control.

3.2.3 Formalization of the logical model

The goal of this stage is to create an online solution of the problem that has been defined. This solution is described by using the HOOD Tree. The steps are as follows:

- Identify objects
- Identify object attributes
- Identify object operations
- Define object relationship
  - Association
  - Inheritance
  - Polymorphism
- Redesign

The purpose of this stage is to convert the HOOD tree skeleton defined in the physical model to a workable HOOD tree with attributes and operations, i.e. the logical model. Then optimization strategies such as inheritance and polymorphism are applied to the logical model to reduce the overall complexity. After several refinements, a formal model of the solution can be elaborated. Details of inheritance and polymorphism are provided in the section 3.7 and 3.8.

Formalization of the logical model is the procedures to build a system conceptual model by converting physical entities to logical entities. The first step is to identify potential objects. An object is a model of a real-world entity that consists of attributes

and operations of those attributes (i.e. nodes in the HOOD tree). After that, identification of private and public attributes for each object is required. Services (operations) provided by objects have to be defined as well. Finally, objects are linked together either in cohesive or loose coupling mode. As a result, a HOOD tree with attributes and operations associated with each tree node is formed. Inheritance is then applied to introduce a super-class for grouping similar objects together. This technique simplifies the creation of similar classes and restricts unauthorized access to any devised classes. Polymorphism is another technique that facilities system simplicity. Polymorphism enables a single service call to give different outcomes from performing the operation on different objects. More details about inheritance and polymorphism are described in the next section. Walkthrough and redesign processes are also essential to assure system quality and completeness. All the steps defined in this stage will be carried out repeatedly until a finalized design is made. Further discussion on inheritance and polymorphism are provided in section 3.7 and 3.8.

3.2.4 Formalization of the solution

This stage performs the following steps to generate the final system:

- Rebuild HOOD tree with inheritance
- Implement logistic control script
- Implement man-machine interfaces
- Configure hardware components
- Generate source code

This part of the methodology can be described as a universal engine to implement the solution. This engine will take the logical model as the main and deduce a result table to map the data from SCADA system with the data from MRP II system. The system is all driven by the HOOD data model, but not the function of any particular system. A sample code generation engine is presented in figure 18.

Formulation of the solution is the implementation stage of the designed system. Hardware configuration such as I/O points and device addresses are associated with the appropriate component in the HOOD tree. Control logic can be implemented by OPC calls. The advantage is to manipulate I/O points by object abstraction instead of non-memorable I/O addresses. Once the man-machine interfaces are built and all the hardware components are installed, source code and executable programs for the specified platform could be generated and the execution system is ready for testing.

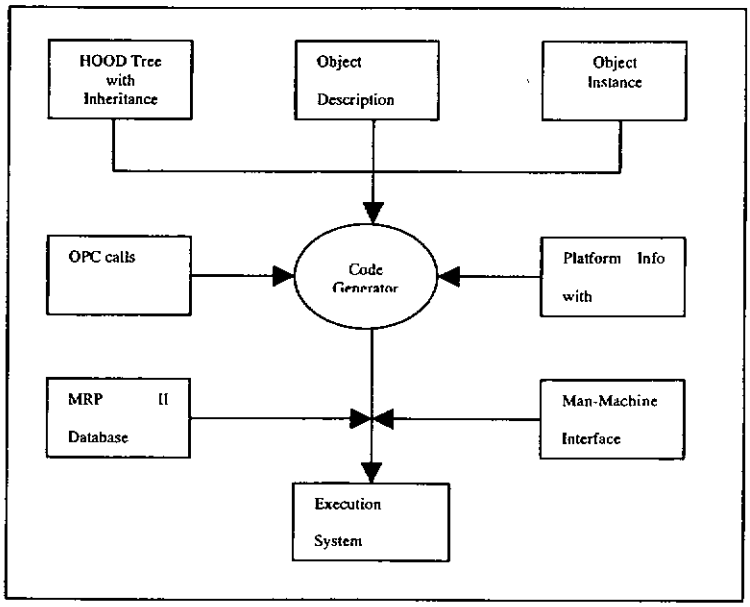


Figure 18: Code Generation.

3.3 Knowledge Representation

Before we can develop the physical and logical model to represent the MRP II data and the shop floor data, the information flow between each component must first be identified. Like the conversion of energy, knowledge in a real-time manufacturing application forms a recursive cycle. The knowledge required performing the integration between MRP II system and SCADA systems could be classified in three basic levels (see Figure 19):

- Level 1 – MRP II Data
- Level 2 – Interchange Interface
- Level 3 – Shop Floor Data

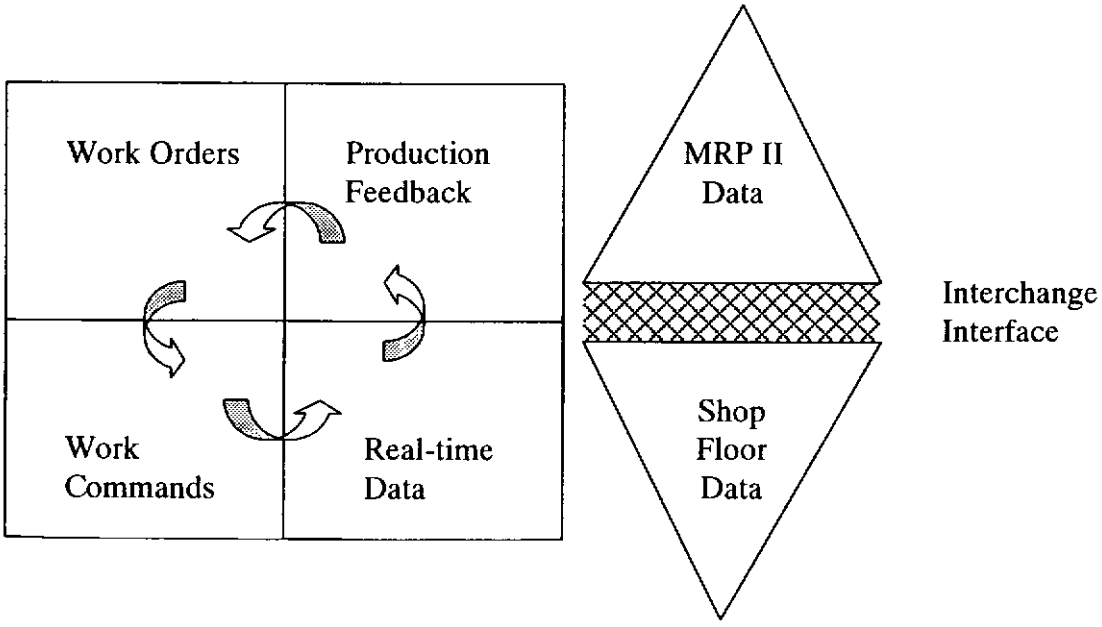


Figure 19: Knowledge model to integrate MRP II and SCADA.

MRP II data represents the work order to shop floor and the feedback data from the production line. A work order provides information on the production schedule, input material data, work procedure and the quantity of finished goods required to meet the ordered production requirements. The feedback data from the shop floor is used as the basis to reschedule production schedule according to the actual production situation. Essential data includes machinery statuses, the information of WIP, the number of surplus and scrap items; and the amount of finished goods.

The interchange interface deals with the conversion from work orders to work commands and the propagation of real-time data back to the MRP II system. This part of knowledge does not exist either in the MRP II system or the SCADA system. Further elaboration of this area is presented in the next section.

Shop floor data includes operating commands and real time production status. Operating commands can either be in the form of procedures on paper or computer program instructions. Production status reflects the actual execution of a work order. Equipment health status and production throughput rates are the main pool of knowledge.

3.4 Data Model

Based on the knowledge representation, we can further expand the model in section 3.3 in terms of data. The data model is the detailed implementation of the knowledge flow diagram. It represents the knowledge to integrate MRP II systems with SCADA systems in terms of data format. The data model is basically a combination of the MRP II HOOD tree and the SCADA HOOD tree. The bottom nodes of both trees are either physical I/O points or logical I/O points. Linkage I/O points are used to linkup the physical and logical I/O points. Figure 20 illustrates the data model.

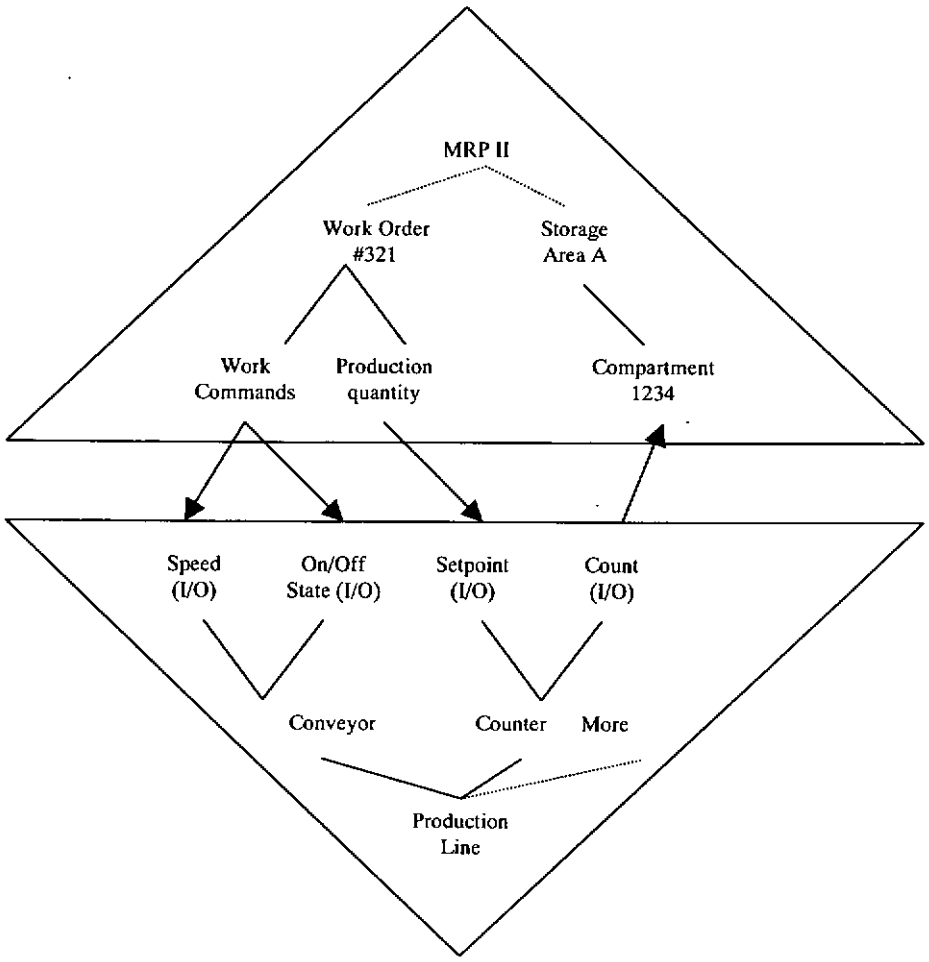


Figure 20: Data Model to integrate MRP II and SCADA.

3.4.1 Physical I/O Points

Physical I/O points are data points captured from data capturing devices directly. All of the real-time data from shop floor are physical I/O points. They are in the form of electronic pulses or analog signals. Numeric data can be gathered through SCADA devices such as communication port, Programmable Logic Controller (PLC) and Remote Terminal Unit (RTU), etc. In the proposed data model, they are classified as attributes of a data object.

3.4.2 Logical I/O Points

Logical I/O points are data points stored in the system as internal variables. All of the MRP II data are logical I/O points. They are usually stored in the format of databases. Logical I/O points can represent numeric data like quantity. Moreover, it can also represent a set of commands and a sequence of jobs. For example, command number '3' can imply to start the third production line. In the proposed data model, they are classified as attributes of a data object.

3.4.3 Data Linkage Points

Data linkage points are all those objects not locating at the bottom of the HOOD tree. They can be classified as passive objects, because they will not take part in the integration directly. The purpose of data linkage points is to organize physical I/O points and logical I/O points in a hierarchical manner. Although it does not take part in the integration, it facilitates the implementation of inherence and polymorphism theory on the HOOD tree, such that the overall complexity of the HOOD tree can be simplified.

3.5 Sample Physical Model

The physical layout of any manufacturing plant can be represented by a HOOD tree. The use of a hierarchical structure has been suggested by many researchers (O'Grady, 1989) in the field of shop floor monitoring and control. The typical hierarchy consists of four levels: factory, shop, cell and equipment. However, the number of hierarchical level is application dependent and even in the same application, each branch could have its own number of hierarchical levels. Therefore a more generic hierarchy is required and the n-level HOOD tree is employed by the proposed methodology. A sample physical model is given in figure 21.

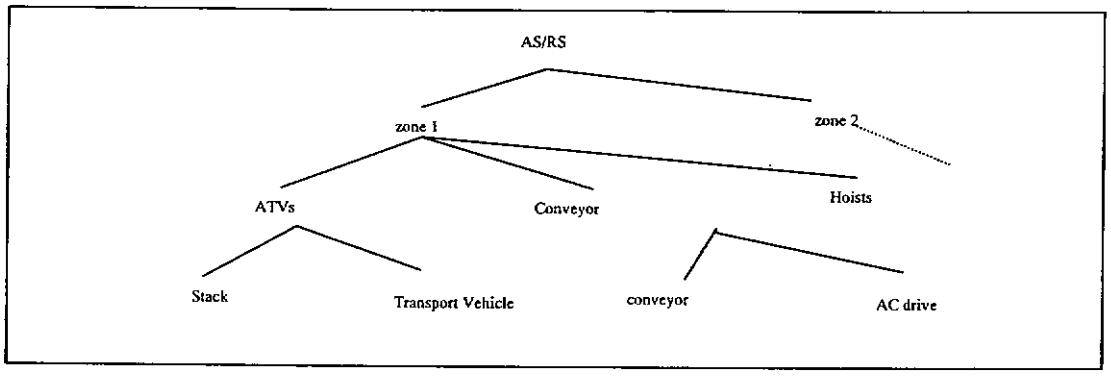
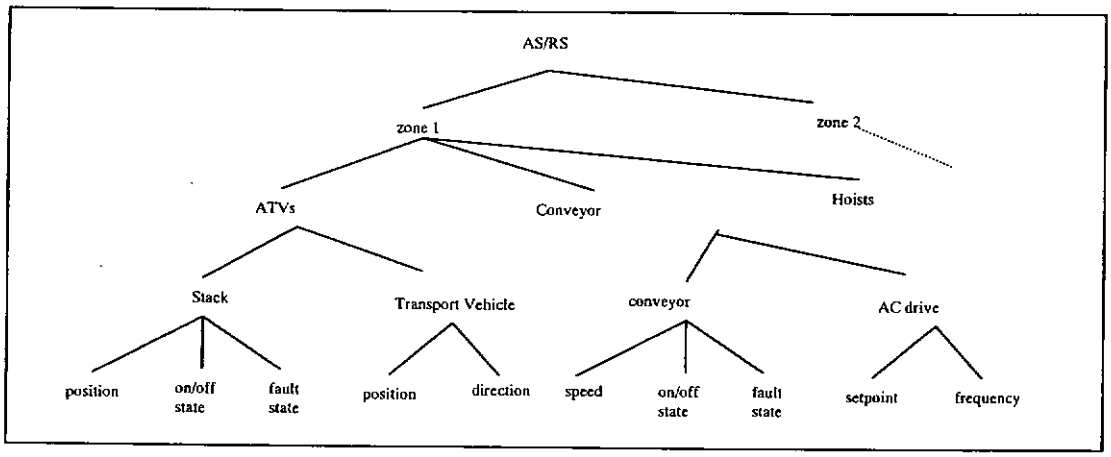


Figure 21: Core HOOD Tree – A physical model.

3.6 Sample Logical Model

The logical model is the optimized HOOD tree derived from the physical model. The physical model represents shop floor layout. It targets to minimize the implementation information required to compile client applications. The logical model is the combination of the physical model with object features. It aims to minimize the implementation information programmatically available to the client.

In short, the physical model is a data model that facilitates the design phase. Whereas the logical model is a data model that facilities the implementation phase. A logical model is illustrated in figure 22.



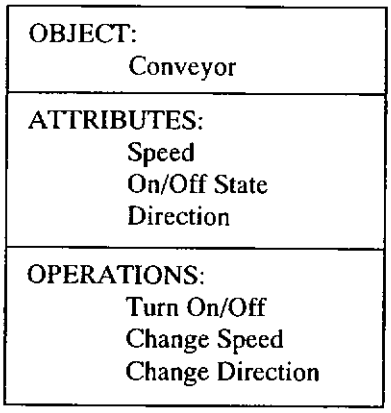


Figure 22: HOOD Tree with attributes and operations – A logical model.

3.7 Inheritance

This section further elaborated the inheritance theory applied to the logical model stated in section 3.2.3.

In our integration methodology, an n-way HOOD tree is formed after the formation of the physical model. A class is a set of objects that share a common structure and a common behavior that is the same type of equipment in the HOOD tree. Inheritance is used to reduce the complexity of the HOOD tree by eliminating duplicate and similar objects. There are many different relationships possible between classes (Booch, 1994). The most important of all these relationships is inheritance. Inheritance is appropriate between classes only when we can have "is a" relationship between the objects. By the inheritance law, if a subclass (or derived class) inherits from the superclass (or base class) then an object of the subclass is an object of the superclass, but not vice versa. In single inheritance, one subclass inherits from one superclass. In multiple inheritance, one subclass inherits from more than one superclass. No matter in single or multiple inheritance, inheritance relationships cannot form a cycle. Therefore inheritance relationships either form a tree or a one-way D-graph (see Figure 23).

As a design strategy, inheritance introduces an intuitive design. Objects constitute the fundamental building block for modeling. This approach enables the RAD (rapid application development) technology. After the HOOD tree is formed, the inheritance relationships between each node could be identified. There are three main types of class relationships in a HOOD tree.

- Parent-to-Child Association (e.g. conveyor contains rollers). The most common relationship between objects is that of association (or aggregation). In association relationship, objects relate to one another in a whole part hierarchy. That is, one object contains another object.

- Parent-to-Child Inheritance (e.g. thermometer is a sensor). The parent-to-child inheritance relationship exists when the child node is inherits from the parent node. In this case, the parent node is described as an abstract base class and it is a pure logical node in the HOOD tree.

- Node-to-Node Inheritance Hierarchy (e.g. motor A and motor B are both inherit from the motor class). The node-to-node inheritance hierarchy is a kind of inheritance as well. However, the abstract base class does not exist in the HOOD tree. A separate n-way 3D graph is required to build on top of the existing HOOD tree.

By combining the n-way HOOD tree with the inheritance hierarchy, we could have an n-way 3D-graph. Hence this is the complete logical model. A sample HOOD tree with the above inheritance relationships is demonstrated in figure 23.

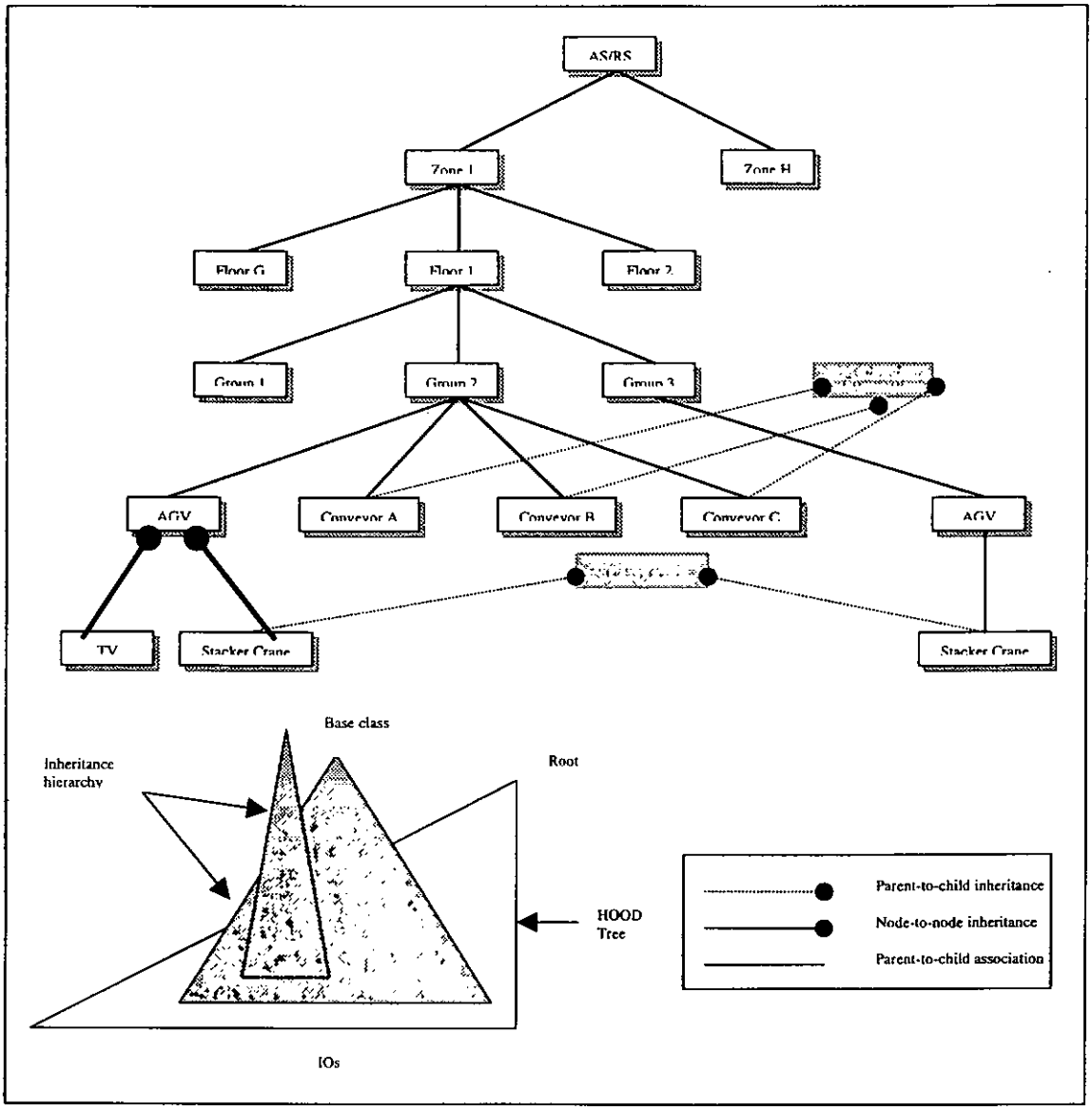


Figure 23: Sample of inheritance and polymorphism.

As an implementation strategy, inheritance fully facilitates re-use of code. Thereby minimizing total coding effort needed. Especially in a SCADA system, each type of component usually has more than one piece and the layout for each subsystem is expected to be similar, inheritance reduces the total implementation effort.

3.8 Polymorphism

This section further elaborated the polymorphism theory applied to the logical model stated in section 3.2.3.

Another special feature of our proposed design paradigm is its multi-platform development capability. Polymorphism is a Latin word that means many (poly~) objects of such form (~morphism). Basically, polymorphism is a concept in type theory such that a name may denote instances of many different classes as long as they are related by some common superclass. Any object denoted by this name is therefore able to respond to some common set of operations (Booch, 1994). In short, different objects could be treated in the same way by the law of polymorphism. The nature of polymorphism is employed as an open system (Nicoloro, 1994) architecture for handling different forms of implementation such as design methodology, programming language, performance requirement and operating system.

In the presence of polymorphism, new design and implementation strategies could be employed. The following are four derived techniques that based on the law of polymorphism.

- Design Polymorphism
- Implementation Polymorphism
- Performance Polymorphism
- Platform Polymorphism

Design polymorphism is the most common technique to be employed in OOAD. This kind of polymorphism separates the detail implementation level from the abstract design level. It also facilities system designer to combine a mountain of similar operations to a single operation for easy handling. For instance, the operation to turn the light on and the operation to turn a heater on are both described as a "turn on" operation in the point of view of a system designer. Therefore the total number of different operations could be highly reduced (Gay, 1994).

Followed by the design stage, implementation polymorphism is the actual phase to built polymorphism by means of coding. This type of polymorphism is language dependent. For example, in C++, virtual function is the feature to support polymorphism. By declaring a base class to be virtual, it implies that classes derived from the base class may have their own version of operations in the base class and these operations are invoked based on the actual object types at run-time.

Performance polymorphism refers to the concept of maintaining and selecting appropriate implementation approach. The selection criteria is usually constrained by execution time, memory space, system configuration, result precision, and so on. In a real-time system, this type of polymorphism is essential if timing constraint is important.

For enabling the feature of multi-platform development, the technique of platform polymorphism is employed. This type of polymorphism is usually integrated with the code generator. With platform polymorphism, the code generation operation could then generate appropriate application according to the specified platform. The most

important aspect in here is that adding new platform generating operation will not affect the existing platforms. A sample polymorphism model is shown in figure 23.

4. Implementation

The proposed methodology as stated in chapter 3 is a data model to integrate MRP II systems and SCADA systems. This chapter further describes the formalization of solution in the proposed methodology stated in figure 17.

In the generic design paradigm, physical aspects of the SCADA system are implemented by encapsulated objects, the functions of the system are modeled by object operations, and the interaction between components are represented by a class and object relationship diagram. The design approach then make use of the nature of object inheritance behavior to associate common components and their characteristics to a super-class object for reducing the complexity during system design and implementation. Polymorphism in this case, is demonstrated by acting a single operation on different equipment to obtain different results according to the equipment type. Emphasis is put on the operation instead of the type of equipment. This approach enables open system architecture for integrating any hardware devices and software programs produced by different manufacturers. Practically, this approach is implemented by the OPC technology.

The proposed implementation strategy is considered as a system configuration tool that will generate an application by the given logical and physical model. This approach gives a significant advantage when building multi-platform applications and it ensures system consistence and data integrity.

The physical aspects of the system are described by the object description and the object instance. The logical linkages between objects are represented by the HOOD tree. Control logic for individual components and subsystems are implemented as the operations acted on the object. With the above information and platform specific data, source code and executable files can be generated. Relational databases and object-oriented database (OODB) (Zhou, 1995) are both suitable to store real-time data gathered from field devices, since both of them can implement the hierarchical structure. For better performance, the main memory database (Cornelio, 1993) could be used. After integrating with the man-machine interface, the system prototype is ready for testing.

The execution of the integration is driven by five main processes: the controller, the dispatcher, the executor, the monitor and the intelligent agent. Figure 24 illustrates the proposed execution system to integrate a MRP II system with real-time data.

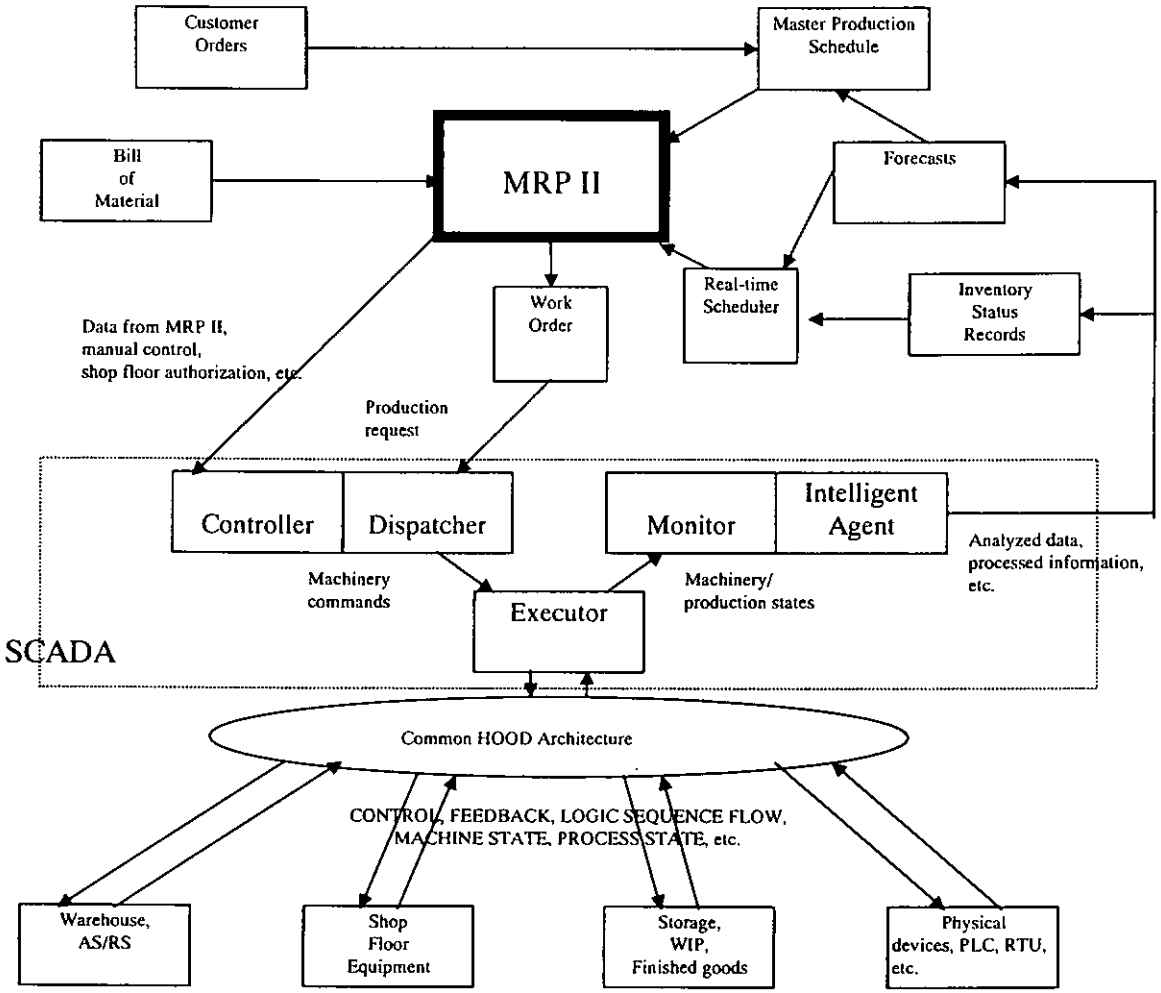


Figure 24: The execution system.

4.1.1 The Controller

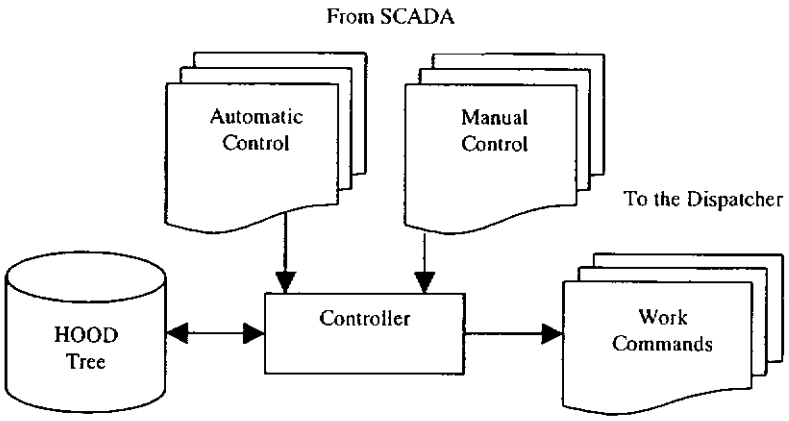


Figure 25: The Controller.

The controller is used to translate automatic control commands and manual control commands from the SCADA application to work commands in HOOD representation. For example, a command to start a conveyor is zone ABC may be interpreted as "Factory/ZoneABC/ConveyorSystem/ConveyorA/Start". Since the HOOD representation is associated with both the shop floor layout and the object representation. Commands issue by the dispatcher is human readable and they can be converted to any object-oriented language.

Work commands are forwarded to the dispatcher instead of directly to the executor. This is an interlocking mechanism to prevent the controller shutting down a machine that is in production.

The controller is an optional process of the whole system. If the overall system does not support supervisory control, then the controller can be removed from the system without affecting other processes.

4.1.2 The Dispatcher

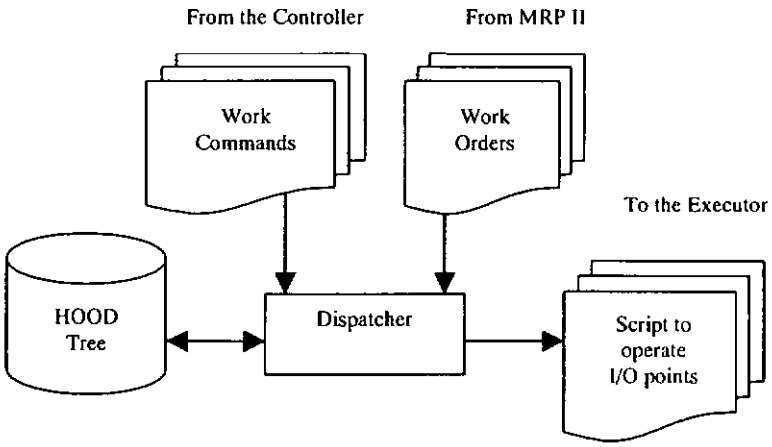


Figure 26: The Dispatcher.

The dispatcher is a multipurpose converter to interpret a work order in terms of raw material information, production quantity, production schedule, etc. All of the above items will be converted into a sequence of operation scripts like the work command from the controller. For example, the operation of retrieving items from compartment #123 through the conveyor may be:

"Factory/ZoneABC/ConveyorSystem/ConveyorA/Start"

"Factory/ZoneABC/StorageArea/Compartment/123/Release"

In the system without supervisory control, the dispatcher will issue worksheet to the shop floor instead of creating computer instructions. For semi-automated processes, a combination of worksheets and computer instructions can be used. If an operation has not been defined in the HOOD tree, then it is defaulted as a manual operation.

4.1.3 The Executor

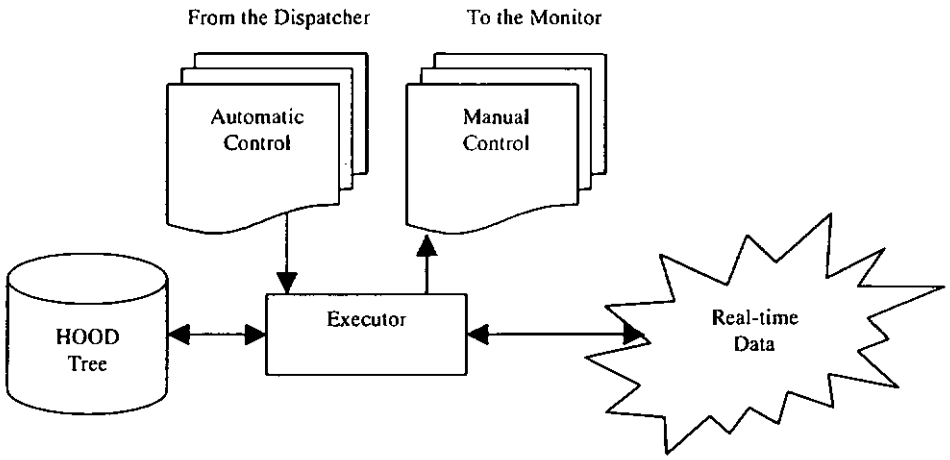


Figure 27: The Executor.

The executor is the bridge between the shop floor equipment and the proposed execution model. The executor is implemented by the OPC technology, such that it can access different equipment produced by different manufacturers in a consistence manner.

Supervisory control is issued through the executor to the shop floor equipment. The executor will convert the work command to the OPC operation by looking up the HOOD tree. Since the work command is implemented as a hierarchical object, the executor can convert the script to OPC object call directly.

By using the same mechanism, the executor will call up field equipment to obtain their current status via the OPC object call. Collected real-time data will be forwarded to the monitor.

4.1.4 The Monitor

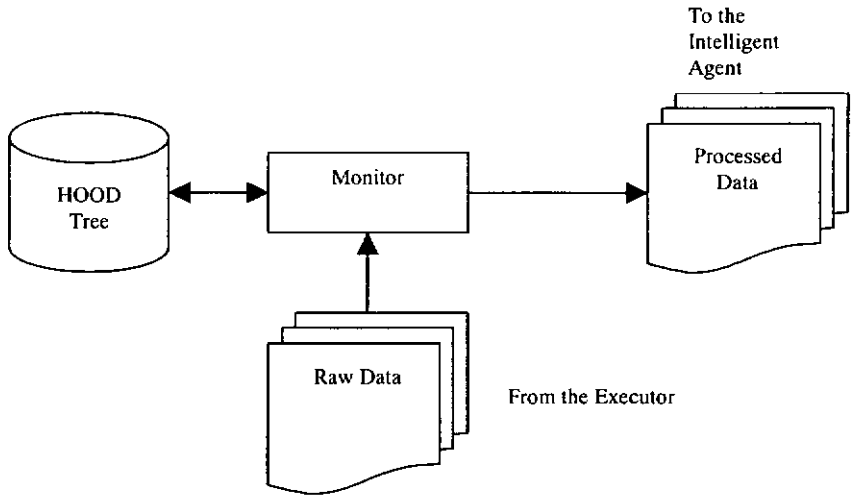


Figure 28: The Monitor.

The monitor is a data processor. Data received from the executor is raw data. Sometimes encoded raw data needs to be processed before their actual value can be extracted. The following is a list of the core functions of the monitor:

- A scale turner. Analog signals are usually measured in 4-20mA. Scaling helps to obtain the numeric reading in the floating-point format.
- An accumulator. The monitor will add up pulse signal and return the sum.
- Logical Analyzer. The monitor can derive logical points from physical points, for example, detecting the high-level alarm limit from a pressure reading.
- An Adder. The monitor can sum up the total value from a set of I/O points, e.g. total flow rate.

4.1.5 The Intelligent Agent

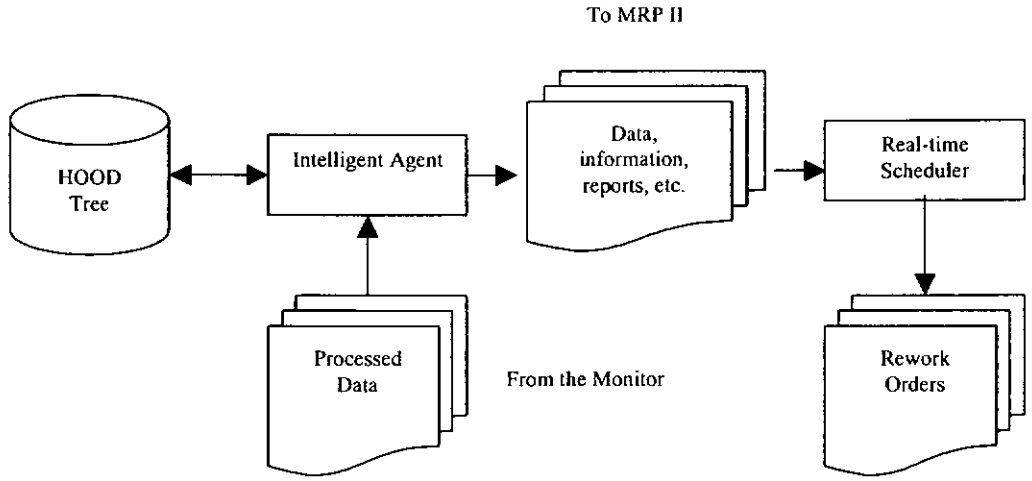


Figure 29: The Intelligent Agent.

The intelligent agent is an analysis tool to produce information from data. Both the data and the information will be feedback to the MRP II system. Processed data is used as the measurement ground for the MRP II function to compare the difference between the schedule plan and the actual performance. Raw data can reflect machinery health status. It can act as an estimation tool for the condition-based maintenance and the preventive maintenance.

Moreover, with the help of processed data and a real-time scheduler, rework order can be issued by the MRP II system automatically on a real-time basis. MRP II systems now gain the ability to reschedule work orders according to the actual shop floor performance. Hence, the integration between MRP II systems and SCADA systems facilitate the forming of a fully automated factory.

A list of the report and information for MRP II system is as follows:

- Real-time production Gantt chart. It provides a real-time production status for the management to govern the production schedule. If the re-scheduling module is enabled in MRP II, real-time data from the shop floor can be used as the rescheduling basis.
- Real-time tooling utilization report. Production efficiency will be affected if production tools are not in a proper condition. With the help of tooling utilization report, preventive maintenance and condition-based maintenance can be performed according to the tooling condition reported from the shop floor.
- Real-time WIP report. This is useful for the management to identify the bottleneck on the shop floor immediately.
- On-line alarm reporting. This report will alert the supervisor if any of the production procedure has a problem.
- On-line inventory report. Real-time inventory data guarantees the MRP II system has accurate and up-to-date inventory information. Moreover, it provides raw data for the warehouse system to keep track of the stock.

5. Case Study

Case Study I illustrates how the proposed integration methodology is used in an industrial oven system. This case study concentrates on demonstrating how can the proposed methodology be applied in the field of manufacturing. This case study was implemented by the collaborated company with the author through the Teaching Company Scheme, however, because of the limitation of space and company policy, detail of the whole system has been simplified.

Case Study II illustrates how the proposed integration methodology is used in a huge AS/RS of an air cargo terminal. This case study concentrates on showing the flexibility and reusability of the proposed methodology in large-scale systems. This case study was implemented by the collaborated company with the author through the Teaching Company Scheme as well.

5.1 [Case Study I] Statement of the Problem

According to the proposed methodology in Figure 17. The first stage is problem identification. A precise statement of the problem is given.

The industrial oven system controls two conveyors, a gas valve, a safety valve, a counter and a burner. The system has to control the temperature by adjusting the valve opening. Figure 30 illustrates the physical components of the industrial system.

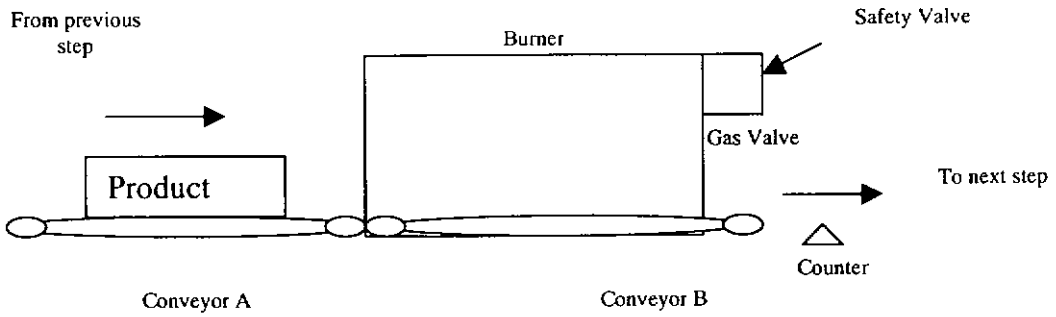


Figure 30: Case Study I: Shop Floor layout.

5.2 *[Case Study I] Analysis and Structuring of Requirement Data*

After defining the problem, we have to declare the system requirements. The following statements are the basic control philosophy for this demonstration.

- "If the temperature is too high, slightly close the gas valve."
- "If the temperature is too low, slightly open the gas valve."
- "If the temperature excesses the high-level limit, shutdown the system (i.e. close the safety valve)."
- "The counter is used to determine number of finished product."

5.3 [Case Study I] The Physical Model

The first step is to construct the physical model. A physical model is established according to the physical layout. Figure 31 illustrates the physical model in HOOD tree format. The physical model in figure 31 is based on the shop floor layout specified in figure 30.

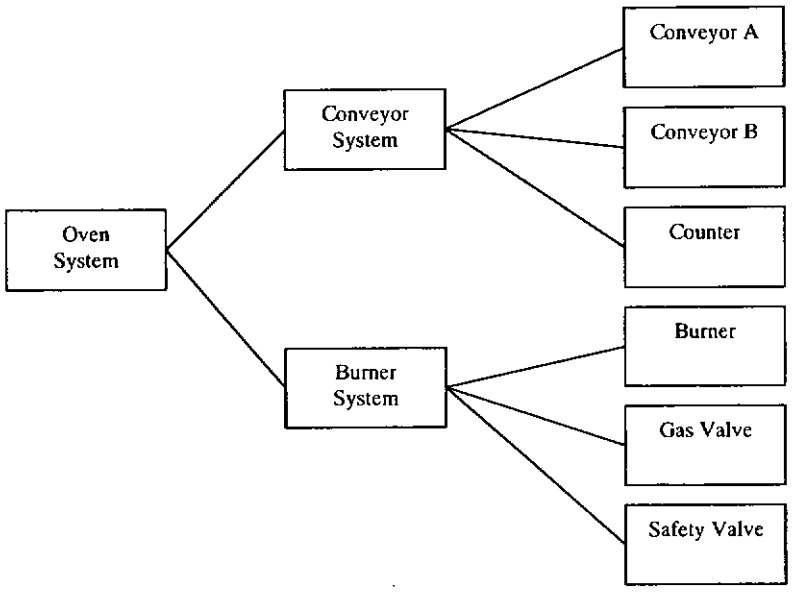


Figure 31: Case Study I: The Physical Model.

5.4 [Case Study I] The Logical Model

5.4.1 Objects Identification

Based on the integration methodology in figure 17. We now identify the logical model. The active objects are conveyor, burner, gas valve, safety valve and counter

and the passive objects: oven system, conveyor system and burner system. Objects, their attributes and object methods are illustrated in figure 32.

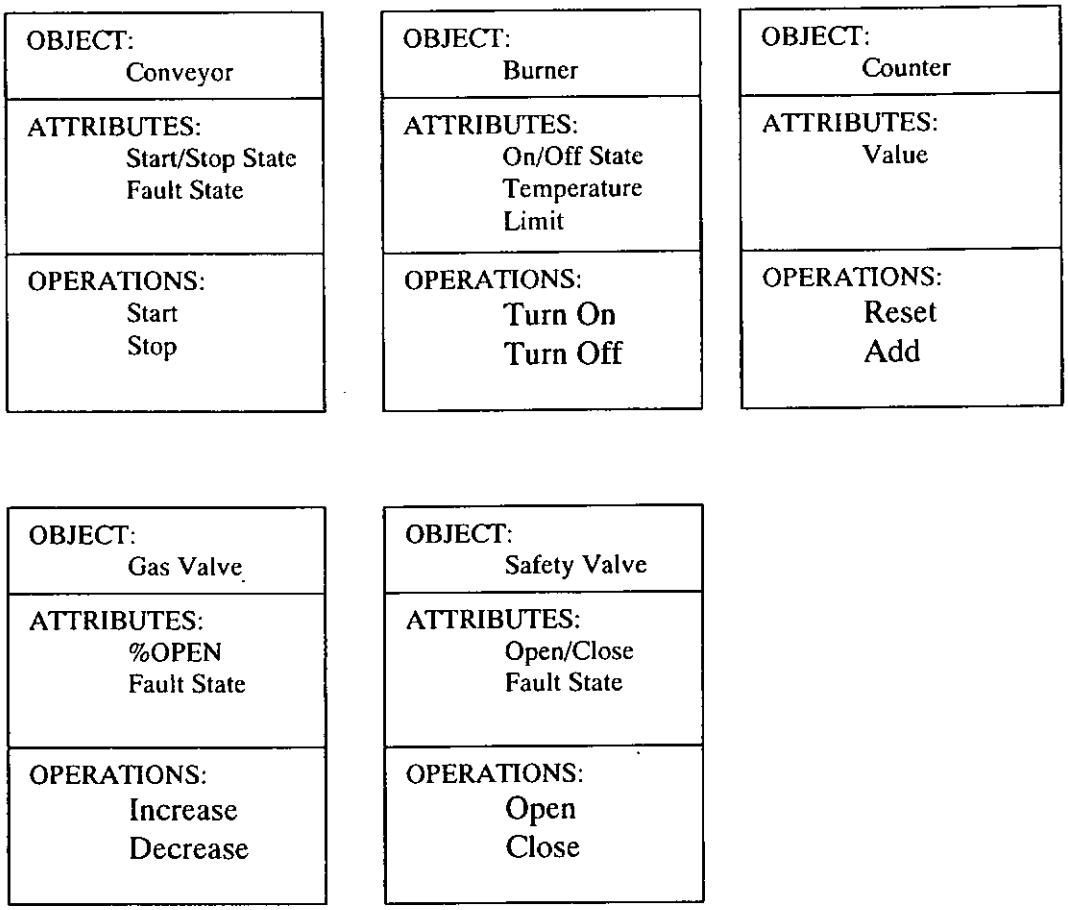


Figure 32: Case Study I: Object Definition.

5.4.2 A Logical Data Model

After identifying the object attributes, we can construct the logical model by adding object attributes to the physical model as shown in figure 33.

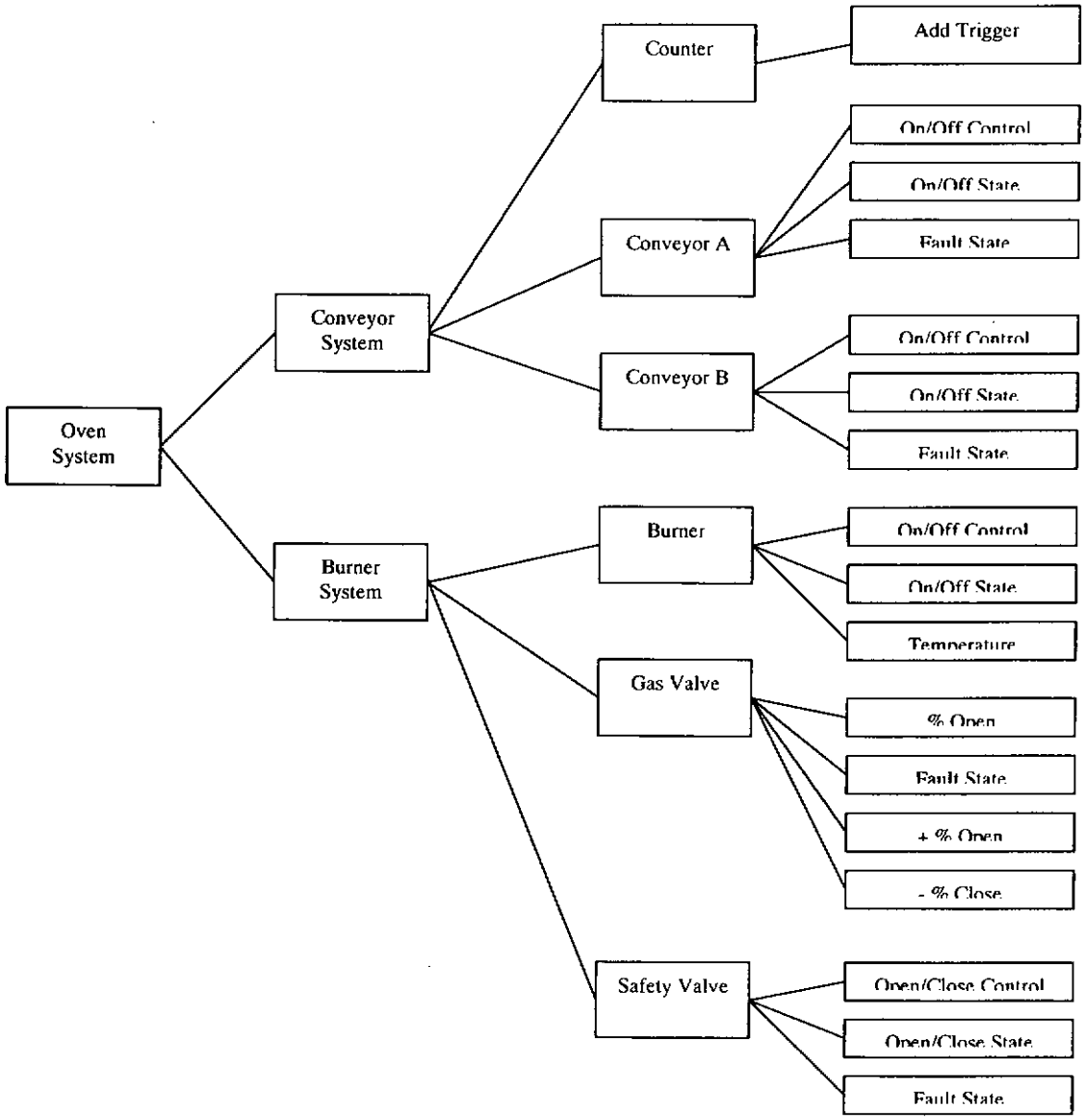


Figure 33: Case Study I: Logical Model (I).

5.4.3 A Logical Data Model with Inheritance and Polymorphism

To optimize the implementation effort, we now apply the rule of inheritance and polymorphism according to the guidelines specified in the integration methodology. An optimized logical model is shown in figure 34. With the help of inheritance and polymorphism, the complexity of conveyors and valves are reduced. Conveyor A and conveyor B are now unified as inheritance objects of the conveyor system. The gas valve and the safety valve are simplified as polymorphism objects of the common valve.

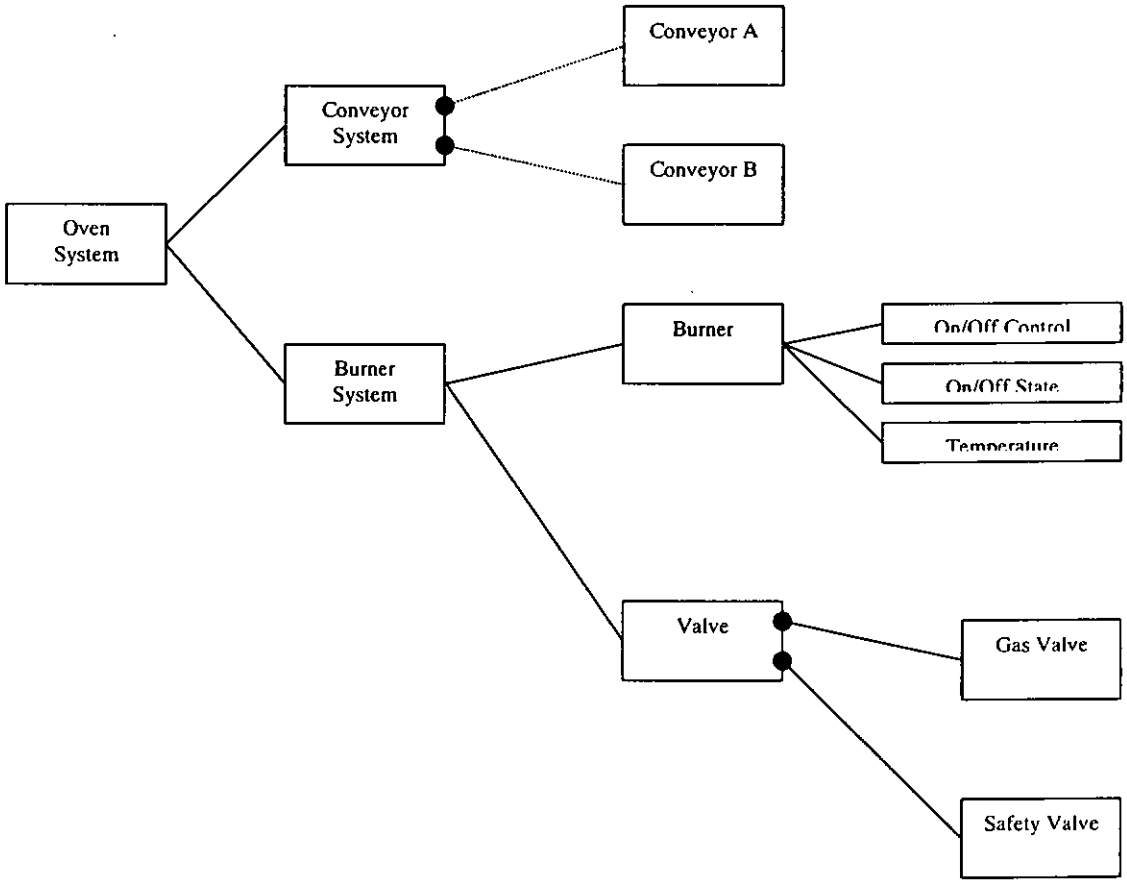


Figure 34: Case Study I: Logical Model (II)

5.4.4 Sample Command Script

This example illustrates the case of section 5.2 (high-level limit). The main idea of the command script is to implement shop floor controls and operations in a human readable and program-ready language. The most important to bear in mind is that the syntaxes and operations of the command script are derived from the HOOD tree. There is no need to memorize or to learn any programming language before writing the script. Hence, the HOOD tree is the language enabler.

```
If OvenSystem.BurnerSystem.Burner.Temp>OvenSystem.BurnerSystem.Burner.Limit
        OvenSystem.BurnerSystem.Valve.Close
        OvenSystem.ConveyorSystem.Stop
        OvenSystem.BurnerSystem.Burner.Off
endif
```

5.4.5 Sample OPC Implementation

The following segment of C++ program illustrate how can the script in 5.4.4 be converted to programming language. This part of work is done by the controller and dispatcher automatically in the execution system.

```
If OPCServer1.Burner->Temp > Burner->Limit
{
        OPCServer2.Valve->Close();              // Close Gas Valve
        OPCServer3.Valve->Close();              // Close Safety Valve
        OPCServer4.Conveyor->Stop();            // Stop Conveyor A
        OPCServer5.Conveyor->Stop();            // Stop Conveyor B
        OPCServer6.Burner->Off();               // Turn Off the Burner
} /* end-if */
```

5.4.6 Execution of the Model

The factory in this case is still under developing and has not been a fully automated factory. Once a work order is released by the MRP II system. The dispatcher will analysis the work order and forward the work command to the shop floor in paper format, since it cannot find the existence of any automated device on the HOOD tree.

After the workman received the job command, he/she will pick up raw materials from the warehouse and prepare them for production. When the system startup button is pressed, the SCADA will issue a start command to the controller. The controller will then translate the start command to a set of equipment startup sequence in the form of HOOD script. Through the dispatcher, HOOD scripts are downloaded to the shop floor equipment by the executor.

When a finished good passes through the counter, a digital pulse will be generated. The executor will report the pulse to the monitor. The monitor acts as an accumulator to store the total number of finished goods. The monitor will then forward the accumulated total to the intelligent agent and finally the total number of finished good feed back to the MRP II system. The actual implementation is through the use of OPC technology. The intelligent agent acted as an OPC server, where the MRP II system is the OPC client.

Sample source code of the developing system is provided in the Appendix.

5.5 *[Case Study II] Statement of the Problem*

According to the proposed methodology in Figure 17. The first stage is problem identification. A precise statement of the problem is given.

The scope of this project is to build a SCADA system to connect with the existing cargo handling system for a cargo terminal. The scope of work is illustrated in figure 35. The system has to monitor and to control all equipment on the shop floor. Figure 36 illustrates the physical components of the cargo system.

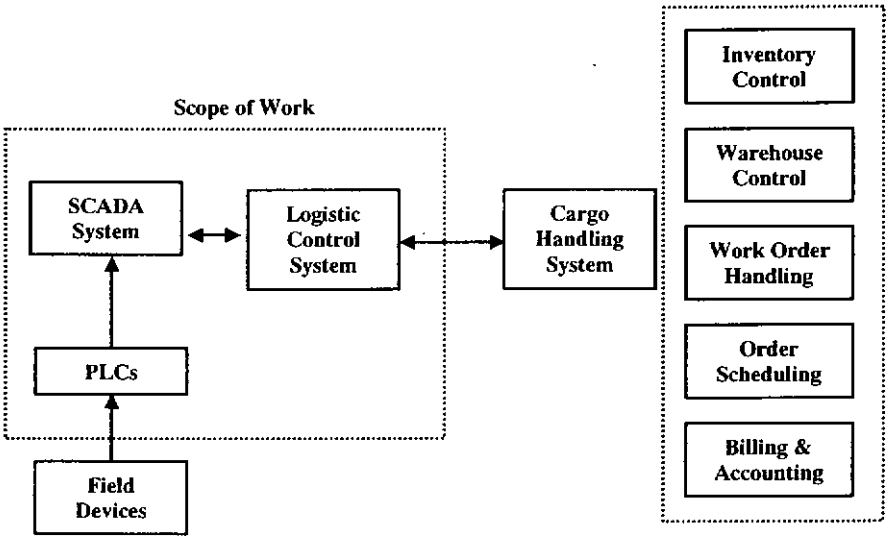


Figure 35: Case Study II: Scope of work.

Figure 36: Case Study II: Shop Floor layout.

5.6 [Case Study II] Analysis and Structuring of Requirement Data

After defining the problem, we have to declare the system requirements. The major significances of this project are as follows:

- The cargo terminal contains two different AS/RSs supplied by two manufacturers. The new system should be able to adopt different hardware and standards used by two systems.
- The system contains over 5000 different components and 400,000 I/O points. The new system should be able to handle that amount of data in both the design and the implementation stage.

5.7 [Case Study II] The Physical Model

The first step is to construct the physical model. A physical model is established according to the physical layout. Figure 37 illustrates the physical model in HOOD tree format. The physical model in figure 37 is based on the shop floor layout specified in figure 36. All hardware components are identified and defined by using the HOOD tree as the below format.

Hood Tree Hierarchy	Logical Name	Objects
Level 1	AREA	Bulk Storage System and Container Storage System
Level 2	WNIG	North Block and South Bock
Level 3	ZONE	Zone E, Zone F, etc.
Level 4	FLOOR	Floor 3, Floor 4, etc.
Level 5	EQUIPMENT	Conveyor, Crane, etc.
Level 6	I/O	Devices I/O points

Figure 37: Case Study II: The Physical Model.

5.8 [Case Study II] The Logical Model

To optimize the implementation effort, we now apply the rule of inheritance and polymorphism according to the guidelines specified in the integration methodology. With the help of inheritance and polymorphism, the complexity of conveyors and cranes are highly reduced. More than 3000 conveyors are now unified as inheritance objects of the conveyor class. 48 cranes are now unified as inheritance objects of the ATV class. As a result, 5000 components are unified to less than 20 object classes. Hence, the overall complexity of the project is highly reduced.

6. Results

6.1 Industrial Projects

A generic methodology to integrate MRP II with real-time data has been developed in this research. The academic research result is justified in the industry through the Teaching Company Scheme that co-operated by the Manufacturing Department, Hong Kong Polytechnic University and the GRD Engineering (H.K.) Limited. The collaborated company is a system integrator mainly involved in developing real-time applications in the manufacturing industry. The integration methodology has been adopted by the collaborated company as the integration strategy in two of their projects during this scheme. With the proposed methodology, the collaborated company has successfully implemented a manufacturing oven system with 800 I/O points and an Automatic Storage and Retrieval System (AS/RS) with over 400,000 I/O points. This result is only possible with the joint research in the Teaching Company Scheme. The integration barriers were solved by the proposed methodology. Moreover, the advantage of the model-driven approach and the possibility of application generation highly reduce the development time on programming, debugging and testing. The collaborated company employs the integration methodology in their ongoing projects.

6.2 System Complexity

For the AS/RS, although there are more than 5,000 different components in the system, the abstract type of equipment could be classified into a few classes. The

main identified classes are bi-directional conveyor, unidirectional conveyor, transport vehicle, stacker crane, workstation and buffer. By the help of inheritance and polymorphism theory, the problem size in BigO analysis (Sommerville, 1992) is highly reduced from BigO[>5000] to BigO[<30].

7. Discussion

7.1 Advantages and Limitations of the Methodology

With the advance of computer technology, many limitations in the past have been broken through. In the early 90's, the client/server technology with computer networks enables data to be interchanged between different software systems, even different platforms and operating systems. Programs and data can be freely distributed all around an enterprise. Information sharing is a major milestone towards a modern manufacturing system.

Secondly, the problem of hardware and software incompatibility has been solved by the OPC technology since 1996. We now are able to design any hardware system without concerning the compatibility of software system. In the past, this aspect limited many research works such as CIM (Computer Integrated Manufacturing) (Adam, 1993) on the conceptual level. Many difficulties to implement the concept in the field of manufacturing can now be overcome. The OPC technology has broken the barrier and it concurs with our methodology in both the academic and the industrial community.

Last but not least, the open connectivity of field devices facilities real-time data capturing. Nowadays, field devices, sensors and controllers are widely opened equipment. They can share captured data through common communication media such as the Ethernet, serial network and the profi bus, etc. Without the above

hardware devices, the integration of MRP II systems with SCADA systems can never be realized.

However the proposed methodology is still facing certain limitations. The major limitation of the methodology depends on the level of automation. This is the core problem in the topic of manufacturing integration. This problem can be classified into three catalogues:

- The process must be handled manually (e.g. artwork).
- The process can be automated, but does not involve any automation components (e.g. packing).
- The process involves automation components, but they are not able to communicate with each other (e.g. proprietary hardware and software).

Although there are many ways to improve the above cases, the efficiency and the effectiveness of the proposed methodology will be highly reduced owning to the possibility of automation. In contrast, more pre-works and modifications on the existing system will be required if we apply the integration methodology to such a system.

Another limitation is concerned with the compatibility with the existing MRP II system. Most of the worldwide software vendors designed their MRP II products based on the concept of object-oriented technology. Their products have open connectivity and they are well prepared to interface with other systems. Multinational enterprises are affordable to employ these products. However, these products may be

too expensive for Small Manufacturing Enterprise (SMEs). If the low-end MRP II products employed by SMEs do not support open connectivity, it will create a difficulty for SMEs to apply the integration methodology.

The integration methodology introduces two modeling techniques:

- Formalization of the physical model
- Formalization of the logical model

The physical model can be built from the shop floor layout and the equipment specification. There is no prerequisite knowledge required to complete this task except the knowledge on the manufacturing process itself. The physical model can be described as a data representation of the shop floor. However, the logical model is not straight forward as the physical model. Based on the core architecture built from the physical model, optimization strategies are integrated to the physical model to form the logical model. Certain knowledge on object-oriented theory, inheritance, polymorphism and complexity analysis is required to construct an effective and efficient model.

To implement the integration methodology, there are two major constraints. The first constraint requires the manufacturer to have a systematic stock keeping system. Since the integration is through the linkage between the item data in MRP II systems and I/O points in SCADA systems, a well-defined stock control procedure is essential to provide adequate information. The second constraint requires the manufacturer to have a complete set of components before defining the data model. Since the

proposed system in figure 17 does not support prototyping, all of the components must be well defined at the first time. Missing a component in the design stage may affect the formalization of our solution.

7.2 Benefits of the Methodology

In this research, the integration methodology was not only served as a design tool for enabling real-time data in MRP II applications, it is a complete framework guides the analysis, design, optimization and the implementation of the integration towards a general solution that was not available in the past.

As stated earlier in section 1.2, the primary objective of this research is to have better control of the execution stage of MRP II systems. Moreover, the real-time data also facilities the management of equipment. With real-time data, condition-based maintenance technique can be applied. As a result, the number of equipment failure is reduced. Hence, the overall productivity can be raised. Some of our preliminary research relating to the application of SCADA in a maintaining environment has been summarized in an article entitled "SCADA in Maintenance System: A Case Study" and is being under reviewed by the Journal of Quality Maintenance Engineering. Because of the scope has been targeted on our integration methodology, it has not been elaborated in here.

Design a system from data models is the main characteristic of the proposed methodology. The use of physical model simplifies the stage of object identification in other object-oriented methodologies. The use of logical model reduces the system

complexity significantly. Moreover, applying inheritance and polymorphism theory in the logical model have enriched the inadequacy in the original HOOD method. In conclusion, this approach provides a consistent manner in all stages of development.

Another state-of-the-art is the HOOD tree. The HOOD tree is simple to build and easy to modify. More importantly, the HOOD tree represents both object attributes and relationships between objects in the same data model. This feature enables the conversion between the design and the implementation. In other word, with the HOOD tree, the design is the implementation and the implementation is the design; this characteristic is essential for code generation solutions. Moreover, the advantage of the model-driven approach and the possibility of application generation highly reduce the development time on programming, debugging and testing.

7.3 Future Work

The proposed methodology may be adapted to support other object-oriented databases, especially Distributed Object-Oriented Databases (DOOD) (Zhou, 1995). The use of DOOD facilities to construct a data model in the global manufacturing environment is necessary. Because of the historical development of MRP II since 1790's, relational databases are used in the existing MRP II applications. Nevertheless, relational theory has no conflicts with the object-oriented approach, applying relational databases on an object-oriented model may affect the overall performance. Besides, relational databases do not support object-oriented features such as inheritance and polymorphism. This aspect limits the design flexibility and future expansion. Therefore research on replacing relational databases with DOOD in

MRP II applications and the integration model can facilities the implementation of a distributed solution.

With the help of DOOD, the hierarchical approach can be upgraded to the network architecture. The integration methodology employs the hierarchical approach as the core data model, since a hierarchical data structure is compatible with relational databases. However, the hierarchical data structure supports one-to-many structure only. It can not cater many-to-many relationship. Even if the use of a hierarchical model or a network model does not make any different in the integration methodology, replacing the core data structure with a more flexible data structure is the trend to innovate knowledge. Inheritance and polymorphism theories can be further explored by using the network data model. Therefore, research on upgrading the hierarchical model to the network model is recommended to explore more optimization strategies through the use of network model.

The integration methodology has been implemented in two industrial plants. However, the generalization of the result needs further applications and conclusions. It is anticipated that further work will be done in different manufacturing industrial by using the integration methodology.

8. Conclusion

In this research, we have proposed an integration methodology (Figure 17) as a general solution to integrate MRP II systems with SCADA systems. The ultimate goal of the proposed integration is to enhance the consistency, and the accuracy of MRP II systems.

There is no doubt that manufacturers are facing an environment that requires more efficient production as well as build-to-order environments. From the survey, it shows that manufacturers want to have more production data to manage their business in order to reduce the production cost. However, most of the manufacturers are not achieving what they want. The survey identified that the lack of real-time production data in the management level is the main reason behind the problem. This research provides the execution layer that connects MRP II systems with shop floor data collection systems. Traditionally, MRP II vendors have embedded process specific knowledge in monolithic systems that are unable to communicate with other systems. The proposed integration methodology falls outside the realm of traditional MRP II systems. It allows manufacturers to seamlessly integrate MRP II systems with SCADA systems and provides a total solution to close the gap between the production level and the management level.

The integration can be summarized with four main steps. First, a physical model is constructed according to the physical shop floor layout to form the basic HOOD tree. Then object attributes and operations are added to each of the objects on the basic HOOD tree. Afterward, inheritance and polymorphism theory is applied to the

HOOD tree to reduce system complexity. A logical model of the HOOD tree is then formed. This is a universal way to construct the logical data model of the proposed methodology.

In the execution stage, the controller will coordinate all of the shop floor equipment. The dispatcher will interpret work orders into work commands. The executor will download the machinery command generated by the controller and the work command interpreted by the dispatcher to the SCADA system based on the shop floor layout defined on the HOOD tree.

During the production stage, real time data are captured by the SCADA system and transmitted to the executor according to the HOOD tree. The executor will forward real-time data to the monitor. The monitor is used to convert the received data to readable format. It also derives logical data from physical data, for example, determining the high-level alarm limit from a hardware device reading. The purpose of the intelligent agent is to translate raw data into meaning information, for example, SPC chart, trend diagram, etc.

Finally the production data and information are forwarded to the MRP II system. MRP II functions will evaluate the difference between the planned schedule and the actual performance. Rescheduling will be performed if necessary. If so, MRP II system will download a new set of work orders and machinery commands to adjust the production environment.

With the integration methodology, the design framework for integrating MRP II systems with real-time data is now available. The success of the methodology in the industry shows that our model-driven approach is a breakthrough of the integration technology. Moreover, real cases indicate that the developing time required on programming, debugging and testing are reduced significantly with the proposed methodology.

To validate the proposed methodology, case studies were carried out with the help of a collaborated company as a Teaching Company Scheme project. The results indicated that the proposed methodology is a feasible solution to integrate MRP II systems and SCADA systems. Two industrial projects are successfully implemented based on the methodology developed in this work.

9. Statement of Originality and Contribution to Knowledge

The major contribution of this research is the design of methodology to enable the integration of MRP II systems and SCADA systems, this work enable both the planning and the execution phase of manufacturing advances from batch system to real-time system.

In addition, innovated knowledge such as HOOD tree with numerous types of inheritance and polymorphism is introduced. HOOD tree is a data model derived from the HOOD method. HOOD tree is the data structure to represent real-world entities in the manufacturing environment. Both the shop floor layout and the material distribution can be transform to a HOOD tree. The core integration concept is through the mapping of the shop floor layout tree and the material distribution tree. Traditional integration methods usually integrate two systems by their functions. Instead of using this conventional approach, we proposed the integration by data models. This approach can identify the problem and provide the solution.

Two unique features of object-oriented technology, inheritance and polymorphism are not supported by the original HOOD method. Our research work has applied the theory of inheritance and polymorphism to the HOOD tree.

Furthermore, deriving from the object theory, three new inheritance laws are introduced in our methodology; they are:

- Parent-to-Child Association
- Parent-to-Child Inheritance

- Node-to-Node Inheritance Hierarchy

and again, Deriving form the object theory, four new polymorphism laws are introduced in our methodology:

- Design Polymorphism
- Implementation Polymorphism
- Performance Polymorphism
- Platform Polymorphism

As a result of our research, several international publications have been completed. They are summarized as follows:

Published:

Lee, K.C., Ip, W.H. and Yung, K.L.

"Hierarchical Object-Oriented Supervisory Monitoring and Control for Manufacturing Enterprises"

Advances in Industrial Engineering Applications and Practice II, Volume Two, pp.1241-1246 (1997)

Lee, K.C., Ip, W.H., Yung, K.L.

"Intelligent Supervisory Monitoring and Control System for Automated Manufacturing"

4th International Conference on Manufacturing Technology in Hong Kong, CDROM. (1997)

Lee, K.C., Ip, W.H., Yung, K.L.

"Data Acquisition and MRP II"

Proceedings of the Management & Technology '98 Seminar, pp. 33-36 (1998)

Accepted (to be published):

Lee, K.C., Ip, W.H., Yung, K.L.

"Inheritance and Polymorphism in Real-time Monitoring and Control Systems"

Journal of Intelligent Manufacturing.

Under Review:

Lee, K.C., Ip, W.H., Yung, K.L.

"SCADA in Maintenance System: A Case Study"

Journal of Quality in Maintenance Engineering

10. Bibliography

1. Aarsten A., Brugali, D. and Menga, G.

   "Designing Concurrent and Distributed Control Systems"

   Communication of the ACM, October, Vol.39, No.10 (1996)

2. Adam, N.R. and Gangopadhyay, A.

   "Integrating Functional and Data Modeling in a Computer Integrated Manufacturing System"

   Proceedings of the 9th International Conference on Data Engineering, April, pp.302-309 (1993)

3. APICS

   "APICS dictionary - Sixth Edition"

   The American Production and Inventory Control Society, Inc. (1987)

4. Berrisform, G. and Burrows, M.

   "Reconciling OO with Turing Machines"

   The Computer Journal, Vol. 38, No. 10, pp. 888-906 (1994)

5. Billaut, J.C. and Roubellat, F.

   "Significant States and Decision Making for Real Time Workshop Scheduling"

   IEEE, 0-8186-4030-8/93, pp.493-500 (1993)

6. Booch, W.

"Object-Oriented Analysis and Design with Application (2nd edition)"

Benjamin/Cumming (1994)

7. Borangiu, T. and Popescu, A.

"Intelligent Components and Object-oriented Concept for CIM"

IFAC Intelligent Manufacturing Systems, pp. 415-419 (1994)

8. Bostel, A.J. and Sagar, V.K.

"Dynamic Control System for AGVs"

Computing & Control Engineering Journal August, IEE (1996)

9. Brinkler, V.C., Twidell, J. and Crane, M.

"SCADA Systems and their application to wind farm performance"

Institute of Energy and Sustainable Development, De Montfort University (1997)

10. Browne, J.

"Interoperable Control"

Manufacturing Engineer, December, pp. 279-283 (1995)

11. Cha, S.K. and Park, J.H.

"An Object-oriented Model for FMS Control"

Journal of Intelligent Manufacturing, 7, pp. 387-391 (1996)

12. Chung, S.H. and Lin, C.W.

"From an Integration Viewpoint to Build a Short-term Production Planning System for Flexible Manufacturing Systems"

Proceedings of the 1991 IEEE International Conference on Robotics and Automation, April, pp. 2224-2230 (1991)

13. Cornelio, A. and Navathe, S.B.

"Using Active Database Techniques for Real Time Engineering Applications"

Proceedings of the 9th International Conference on Data Engineering, April, pp.100-107 (1993)

14. Correll, J.G.

"Reengineering the MRP II Environment"

IEE Solutaions, July, pp. 24-27 (1995)

15. Dongarra, J.J., Otto, S.W., Snir, M. and Walker, D.

"A Message Passing Standard for MRP and Workstations"

Communication of the ACM July, Vol.39, No.7 (1996)

16. Editor

"Process comes to terms with MRP"

Works Management, June (1993)

17. Fitzgerald, A.

"Enterprise Resource Planning (ERP) – Breakthrough or Buzzword?"

Oracle Corporation UK Ltd, UK (1994)

18. Gay, R.

"Integrating GIS and SCADA Systems"

AM/FM International, pp. 581-587 (1994)

19. Grimble, M.J.

"Control's a Big Gamble"

Computing & Control Engineering Journal, April, p.111-112, IEE (1996)

20. Gunther, O. and Lamberts, J.

"Object-oriented Techniques for the Management of Geographic and Environmental Data"

The Computer Journal, Vol. 3, No.1, pp. 16-25 (1994)

21. Haban, D. and Shin, K.G.

"Application of Real-time Monitoring to Scheduling Tasks with Random Execution Times"

IEEE Transactions on Software Engineering, Vol. 16, No. 12, December, pp.1374-1389 (1990)

22. Haigh, D. and Adams, W.

"Why 64 bit Computers in the Control Room"

Computing & Control Engineering Journal, December, p.277-281, IEE, (1996)

23. Hargrove, S.K.

"A Systems Approach to Fixture Planning and Design"

Internal Journal of Advanced Manufacturing Technology, Vol. 10, pp. 169-182 (1995)

24. Harhalakis, G., Lin, C.P. and Mark, L. and Pedro, M.M.

"Information Systems for Integrated Manufacturing (INSIM)"

IEEE, 0-7803-0233-8/91, pp. 335-341 (1991)

25. Harhalakis, G., Lin, C.P., Mark, L. and Pedro, M.M.

"Implementation of Rule-Based Information Systems for Integrated Manufacturing"

IEEE Transactions on Knowledge and Data Engineering, Vol. 6, No. 6, pp. 892-908 (1994)

26. Harris, W., McClatchey, R. and Baker, N.

"The Use of an Object Repository in the Configuration of Control Systems at CERN"

Proceedings of the 6th International Conference CISMOD'95, pp.153-163 (1995)

27. HOOD User Group

"HOOD Reference Manual Issue 3.1.1"

Prentice Hall International (1992)

28. Horn, F. and Stefani, J.B.

"On Programming and Supporting Multimedia Object Synchronization"

The Computer Journal, Vol. 36, No. 1, pp. 4-18 (1993)

29. Johansson O.

"Using an Extended ER-Model Based Data Dictionary to Automatically Generate Product Modeling Systems"

Proceedings of the First International Conference, ADB-94 Vadstena, June, pp. 42-61 (1994)

30. Jordan, P., Browne, J. and Browne, M.

"Production Activity Control for Small Manufacturing Enterprises"

Knowledge-Based Reactive Scheduling, pp.29-34 (1994)

31. Jorgensen, K.A. and Madsen, O.

"Object-oriented modelling of Active Systems"

IFAC Intelligent Manufacturing Systems, pp. 129-133 (1994)

32. Kehoe, R. and Jarvis, A.

"ISO 9000-3 A Tool for Software Product and Process Improvement"

Springer (1995)

33. Khmelnitsky, E., Kogan, K. and Maimon, O.

"Optimal Flow Control for Scheduling in Manufacturing Systems with Continuous Setup"

IEEE, 0-8186-6510-6/94, pp. 178-183 (1994)

34. Lee, K.C., Ip, W.H., Yung, K.L.

"Hierarchical Object-Oriented Supervisory Monitoring and Control for Manufacturing Enterprises"

Advances in Industrail Engineering Applications and Practice II, Volume Two, pp.1241-1246 (1997)

35. Lee, K.C., Ip, W.H., Yung, K.L.

"Intelligent Supervisory Monitoring and Control System for Automated Manufacturing"

4th International Conference on Manufacturing Technology in Hong Kong, CDROM. (1997)

36. Lee, K.C., Ip, W.H., Yung, K.L.

"Data Acquisition and MRP II"

Proceedings of the Management & Technology '98 Seminar, pp. 33-36 (1998)

37. Lee, W.J, Chen, M.S and Wang, S.P.

"Development of a Real Time Power System Dynamic Performance Monitoring System"

IEE 2nd International Conference on Advances in Power System Control, Operation and Management, December, pp.11-16 (1993)

38. Loose, G.

"Fieldbus and its Impact on the Automation User"

Computing & Control Engineering Journal, December, p259-262, IEE (1995)

39. Mazumdar, S. and Lazar, A.A.

"Modeling the Environment and the Interface for Real-Time Monitoring and Control"

ICC' 91, IEEE, CH2984-3/91/0000-1598, pp. 1598-1603 (1991)

40. Meyer, B.

"Object-oriented Software Construction"

Prentice Hall International, 1988

41. Mitchell, S.E. and Wellings A.J.

"Synchronisation, Concurrent Object-oriented Programming and the Inheritance Anomaly"

Computer Languages, Vol. 22, No. 1, pp. 15-26 (1996)

42. Muller, D.J., Jackman, J.K. and Fitzwater, C.F.

"A Simulation-Based Work Order Release Mechanism for a Flexible Manufacturing System"

Proceedings of the 1990 Winter Simulation Conference, pp. 599-602 (1990)

43. Murthy, D.N.P. and Ma, L.

"MRP with Uncertainty: A Review and Some Extensions"

International Journal of Production Economics, pp.51-64, (1991)

44. Muth., P., Rakow, T.C., Weikum, G., Brossler, P. and Hasse, C.

"A Simulation-Based Work Order Release Mechanism for a Flexible Manufacturing System"

Proceedings of the 9th International Conference on Data Engineering, April, pp. 233-242 (1993)

45. Nicoloro, M.A.

"Commonwealth Gas Company SCADA/GASS Project Implementation and Future Linkage to AM/FM/GIS"

AM/FM International, pp. 59-68 (1994)

46. O'Grady, P.J. and Lee, K.H.

"An Intelligent Cell Control System for Automated Manufacturing"

Knowledge-based Systems in Manufacturing, pp. 151-172 (1989)

47. OPC Taskforce

"OLE for Process Control Standard, Final-Release 1.0"

OPC Foundation (1986)

48. Peshek, C.J. and Mellish, M.T.

"Recent Developments and Future Trends in PLC Programming Languages and Programming Tools for Real-Time Control"

IEEE Cement Industry Technical Conference, May, pp.218-230 (1993)

49. Piciacchia, F.R.

"Plant Operations Control: A Vital Necessity for Executing the JIT Schedule in a CIM Plant"

Proceedings of the Autofact'89 Conference, pp.22/49-22/67 (1989)

50. Poo, C.C. and Lee, S.Y.

"An Object-oriented Systems Modelling Method based on the Jackson Approach"

The Computer Journal, Vol. 37, No. 8, pp. 669-682 (1994)

51. Poon, J.L.

"Integration of MRP and JIT and Its Applicability in Hong Kong's Industries"

IEEE, 07803-0161-7/91, pp. 836839 (1991)

52. Rhodes, R.A.

"Making SCADA a Sustainable Competitive Advantage"

AM/FM International, pp. 225-230 (1994)

53. Robinson P.J.

"Hierarchical Object-Oriented Design"

Prentice Hall, pp. 175-180 (1992)

54. Russell, D.W.

"Integration of PLCs and Databases for Factory Information Systems"

IEEE, TH0309-5/90/0000/0730, pp.730-737 (1990)

55. Sampson, R.A.

"Integration of MRP with the Flexible Manufacturing System"

FMS for Electronics, EE85-130 (1985)

56. Seidman, A.

"Intelligent Information Models for Operating Automated Storage and Retrieval Systems"

Knowledge-based Systems in Manufacturing, pp.279-304 (1989)

57. Selic, B., Gullekson, G. and Ward, P.T.

"Real-Time Object-oriented Modeling"

Wiley (1994)

58. Senn, J.A.

"Analysis & Design of Information Systems"

McGraw-Hill (1989)

59. Shahid, M.S., Saygin, C. and Eskicioglu, H.

"A Frame-Based Object-oriented Product Modelling System for Rotational Components"

IFAC intelligent Manufacturing Systems, pp. 135-138 (1994)

60. Sodhi J. and Sodhi, P.

"Object-Oriented Methods for Software Development"

McGraw Hill (1996)

61. Sommerville, I.

"Software Engineering - fourth edition"

Addison-Wesley (1992)

62. Szelke, E.

"Intelligent Supervisory Control and Reactive Scheduling for Event Sensitive Dynamic Manufacturing Environment"

IFAC Intelligent Manufacturing Systems, pp. 345-351 (1994)

63. Tanaka, Y., Ogawa, F. and Katayama, H.

"Ordering loading operation management systems for MRP: Mathematical programming versus a knowledge-based approach"

International Journal of Technology Management, Vol. 7, Nos 4/5, pp. 290-301 (1992)

64. Turbide, D.A.

"MRP II Still Number One!"

IEE Solutaions, July, pp. 28-31 (1995)

65. Villa, A.

"Production Planning Architectures: A Common Framework for the Comparison of MRP II, OPT, JIT"

IEEE, 0-8186-1966-X/90/0000/0526, pp. 526-530 (1990)

66. Vlient, H.V.

"Software Engineering"

John Wiley & Sons (1993)

67. Ward, R.M.

"Application of COGSYS to Real-time Process Control"

68. West, M.

"Object Technology: An Overview"

BCS Computer Bulletin, Feb, pp.2-3, (1996)

69. Woodgate, H.S.

"MRP III – Material Flow Control in JIT Manufacturing"

Proceedings of the Autofact'89 Conference, pp.22/9-22/24 (1989)

70. Zhou L., Rundensteiner, E.A. and Shin, K.G.

"OODB Support for Real-Time Open-Architecture Controllers"

Proceedings of the 4th International Conference on Database Systems for Advanced Applications, April, pp. 206-213 (1995)

11. Appendixes

11.1 Survey Sample

Remark: This questionnaire was prepared and distributed through the collaboration company of this project in 1997. The following is a list of questions in the questionnaire and a summary of the result. Detailed company information and sensitive data is excluded from this report.

1. Which one best describes the primary business performed at your location?
A. Discrete Manufacturing
B. Product Manufacturing
C. Process Manufacturing
D. Others

Q1	Percentage
A	21%
B	68%
C	8%
D	3%

2. Number of Employees?
A. Under 50
B. 50-99
C. 100-500
D. Over 500

Q2	Percentage
A	6%
B	19%
C	32%
D	43%

3. What is your primary job function?
A. Executive Management
B. Product/Service Management
C. Engineering/Technical Management
D. Operating Management

Q3	Percentage
A	53%
B	22%
C	21%
D	4%

4. What is your organization unit annual turnover?
A. Less than HK$1 million
B. HK$1 million – HK$9.9 million
C. HK$10 million – HK$49.9 million
D. Over HK$50 million

Q4	Percentage
A	2%
B	32%
C	57%
D	9%

5. What is your organization unit annual budget for software products?
A. Less than HK$50,000
B. HK$50,000 – HK$99,999
C. HK$100,000 – HK$499,999
D. Over HK$500,000

Q5	Percentage
A	14%
B	74%
C	11%
D	1%

6. In what areas of the world does your organization have branches or factories?
A. Hong Kong
B. China
C. Asia
D. Other

Q6	Percentage
A	22%
B	69%
C	7%
D	2%

7. Have your organization employ the following software? (check all that apply)
A. Accounting Software
B. CAD/CAM Software
C. Manufacturing Software (e.g. MRP II, ERP)
D. Shop Floor Control Software (e.g. Bar Code, SCADA)

Q7	Percentage (Individual)
A	93%
B	42%
C	47%
D	9%

8. What is your organization unit annual wastage on scrap items?
A. Less than HK$50,000
B. HK$50,000 – HK$99,999
C. HK$100,000 – HK$499,999
D. Over HK$500,000

Q8	Percentage
A	7%
B	20%
C	65%
D	8%

9. What is the time basis for you to tell the production status? (For example, how many items of a particular product have been made?)
A. Yes, I can tell from my computer or the automation equipment.
B. Yes, I can tell on a daily basis.
C. Yes, I can tell, but it takes more than a day to find it out.
D. No, I can't tell. There is no way to identify.

Q9	Percentage
A	2%
B	63%
C	35%
D	0%

10. Which of the following is the most important for your organization? (check one only)

A. Reducing stock level and work-in-progress
B. Reducing production cost (e.g. scrap items)
C. Increasing quality, productivity and capacity
D. Increasing production management information

Q10	Percentage
A	9%
B	44%
C	6%
D	41%

11. Which of the following is the most critical for your organization? (check one only)

A. Machinery Data
B. Production Data
C. Inventory Data
D. Costing Data

Q11	Percentage
A	7%
B	44%
C	12%
D	37%

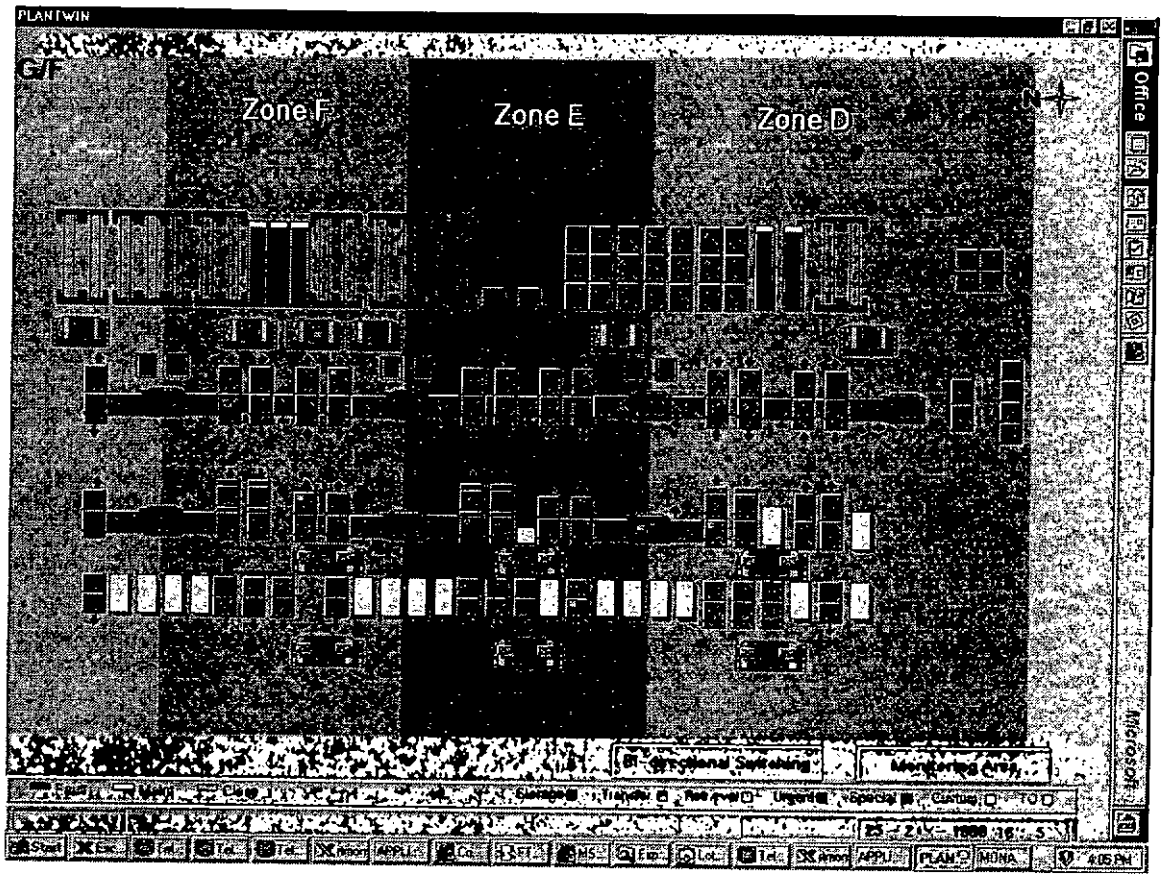
12. Which of the following need to be improved for your organization? (check one only)

A. Quantity of Data (enough data?)
B. Quality of Data (well prepared?)
C. Timing of Data (up-to-date information?)
D. Accuracy of Data (data realistic?)

Q12	Percentage
A	14%
B	4%
C	43%
D	39%

11.2 Sample Shop Floor Layout

The following is a sample man-machine interface of the case study.



11.3 Sample HOOD Tree

The following is a tabular represent of an HOOD tree.

```
ZONE [Zone K]
     STATE [DEFAULT]
     LOGLVL [15]  GINDEX [NONE]

FLOOR [G/F]
     STATE [DEFAULT]
     LOGLVL [10]  GINDEX [NONE]

EQUIP [PL0K1000]
     STATE [DEFAULT]
     LOGLVL [5]  GINDEX [NONE]

     IO [AutoFault]
         STATE [0] [Normal] NORMAL
         STATE [1] [FAULT]
           ALARM [PL0K10000804WBad RAM at Power-up]
         DEVICE [PL0K1000]  ADDRESS [400/0]
         RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20141]
         LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

     IO [AutoFault]
         STATE [0] [Normal] NORMAL
         STATE [1] [FAULT]
           ALARM [PL0K10001201WProcessor in TEST Mode]
         DEVICE [PL0K1000]  ADDRESS [400/1]
         RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20142]
         LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

     IO [AutoFault]
         STATE [0] [Normal] NORMAL
         STATE [1] [FAULT]
           ALARM [PL0K10001202WProcessor in PROG Mode]
         DEVICE [PL0K1000]  ADDRESS [400/2]
         RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20143]
         LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

     IO [AutoFault]
         STATE [0] [Normal] NORMAL
         STATE [1] [FAULT]
           ALARM [PL0K10001203WProcessor burning EEPROM]
         DEVICE [PL0K1000]  ADDRESS [400/3]
         RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20144]
         LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

     IO [AutoFault]
         STATE [0] [Normal] NORMAL
         STATE [1] [FAULT]
           ALARM [PL0K10001204WProgram Download in Process]
         DEVICE [PL0K1000]  ADDRESS [400/4]
         RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20145]
         LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

     IO [AutoFault]
         STATE [0] [Normal] NORMAL
         STATE [1] [FAULT]
           ALARM [PL0K10001206WForces Present]
         DEVICE [PL0K1000]  ADDRESS [400/5]
         RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20146]
         LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

     IO [AutoFault]
         STATE [0] [Normal] NORMAL
         STATE [1] [FAULT]
           ALARM [PL0K10001205WOnline Programming in Process]
         DEVICE [PL0K1000]  ADDRESS [400/6]
         RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20147]
         LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

     IO [AutoFault]
         STATE [0] [Normal] NORMAL
         STATE [1] [FAULT]
           ALARM [PL0K10000807WLow Battery Power]
         DEVICE [PL0K1000]  ADDRESS [400/7]
         RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20148]
         LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

     IO [AutoFault]
         STATE [0] [Normal] NORMAL
         STATE [1] [FAULT]
           ALARM [PL0K10000808FProgram File Corrupted]
         DEVICE [PL0K1000]  ADDRESS [400/10]
         RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20149]
         LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

     IO [AutoFault]
         STATE [0] [Normal] NORMAL
         STATE [1] [FAULT]
           ALARM [PL0K10000809FAddress Corrupted]
         DEVICE [PL0K1000]  ADDRESS [400/11]
         RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20150]
         LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]
```

```
IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10000810FWatchdog Time-out]
    DEVICE [PL0K1000]  ADDRESS [400/12]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20151]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10000811FPLC Hardware Fault]
    DEVICE [PL0K1000]  ADDRESS [400/13]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20152]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10001207WLCS/IS Message Send FIFO Full]
    DEVICE [PL0K1000]  ADDRESS [400/17]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20153]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10000901FEmergency Stop Relay Tripped]
    DEVICE [PL0K1000]  ADDRESS [400/20]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20154]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10000901FEmergency Stop on Main Panel Pushed]
    DEVICE [PL0K1000]  ADDRESS [400/21]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20155]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10000901FEmergency Stop on +MP0K101 Pushed]
    DEVICE [PL0K1000]  ADDRESS [400/22]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20156]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10000901FEmergency Stop on +OM0K103 Pushed]
    DEVICE [PL0K1000]  ADDRESS [400/23]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20157]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10000901FEmergency Stop on +MP0K102 Pushed]
    DEVICE [PL0K1000]  ADDRESS [400/24]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20158]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10000901FEmergency Stop on +OM0K104 Pushed]
    DEVICE [PL0K1000]  ADDRESS [400/25]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20159]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10000901FEmergency Stop on +OM0K105 Pushed]
    DEVICE [PL0K1000]  ADDRESS [400/26]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20160]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10000901FEmergency Stop on +OP0K106 Pushed]
    DEVICE [PL0K1000]  ADDRESS [400/27]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20161]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10000901FEmergency Stop on +MP0U103 Pushed]
    DEVICE [PL0K1000]  ADDRESS [400/30]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20162]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10000112W24V Power Supply is Off]
    DEVICE [PL0K1000]  ADDRESS [400/40]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20163]
```

```
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10000111W220/380 V Power Supply is Off]
    DEVICE [PL0K1000]  ADDRESS [400/41]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20164]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10001002FRemote I/O Rack 00 Faulted]
    DEVICE [PL0K1000]  ADDRESS [400/60]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20165]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10001002FRemote I/O Rack 01 Faulted]
    DEVICE [PL0K1000]  ADDRESS [400/61]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20166]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10001002FRemote I/O Rack 02 Faulted]
    DEVICE [PL0K1000]  ADDRESS [400/62]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20167]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10001002FRemote I/O Rack 03 Faulted]
    DEVICE [PL0K1000]  ADDRESS [400/63]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20168]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10001002FRemote I/O Rack 04 Faulted]
    DEVICE [PL0K1000]  ADDRESS [400/64]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20169]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10001002FRemote I/O Rack 05 Faulted]
    DEVICE [PL0K1000]  ADDRESS [400/65]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20170]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10001002FRemote I/O Rack 06 Faulted]
    DEVICE [PL0K1000]  ADDRESS [400/66]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20171]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10001002FRemote I/O Rack 07 Faulted]
    DEVICE [PL0K1000]  ADDRESS [400/67]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20172]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10001002FRemote I/O Rack 10 Faulted]
    DEVICE [PL0K1000]  ADDRESS [400/70]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20173]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10001002FRemote I/O Rack 11 Faulted]
    DEVICE [PL0K1000]  ADDRESS [400/71]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20174]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10001002FRemote I/O Rack 12 Faulted]
    DEVICE [PL0K1000]  ADDRESS [400/72]
    RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20175]
    LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

IO [AutoFault]
    STATE [0] [Normal] NORMAL
    STATE [1] [FAULT]
      ALARM [PL0K10001002FRemote I/O Rack 13 Faulted]
```

```
        DEVICE [PL0K1000]  ADDRESS [400/73]
        RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20176]
        LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

    IO [AutoFault]
        STATE [0] [Normal] NORMAL
        STATE [1] [FAULT]
          ALARM [PL0K10001002FRemote I/O Rack 14 Faulted]
        DEVICE [PL0K1000]  ADDRESS [400/74]
        RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20177]
        LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

    IO [AutoFault]
        STATE [0] [Normal] NORMAL
        STATE [1] [FAULT]
          ALARM [PL0K10001002FRemote I/O Rack 15 Faulted]
        DEVICE [PL0K1000]  ADDRESS [400/75]
        RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20178]
        LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

    IO [AutoFault]
        STATE [0] [Normal] NORMAL
        STATE [1] [FAULT]
          ALARM [PL0K10001002FRemote I/O Rack 16 Faulted]
        DEVICE [PL0K1000]  ADDRESS [400/76]
        RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20179]
        LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

    IO [AutoFault]
        STATE [0] [Normal] NORMAL
        STATE [1] [FAULT]
          ALARM [PL0K10001002FRemote I/O Rack 17 Faulted]
        DEVICE [PL0K1000]  ADDRESS [400/77]
        RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20180]
        LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

EQUIP [PL0K1100]
    STATE [DEFAULT]
    LOGLVL [5]  GINDEX [NONE]

    IO [AutoFault]
        STATE [0] [Normal] NORMAL
        STATE [1] [FAULT]
          ALARM [PL0K11000112W24V Power Supply is Off]
        DEVICE [PL0K1100]  ADDRESS [400/360]
        RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20181]
        LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

    IO [AutoFault]
        STATE [0] [Normal] NORMAL
        STATE [1] [FAULT]
          ALARM [PL0K11000114FMaintenance Group Main Switch is Tripped]
        DEVICE [PL0K1100]  ADDRESS [400/361]
        RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20182]
        LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]

    IO [AutoFault]
        STATE [0] [Normal] NORMAL
        STATE [1] [FAULT]
          ALARM [PL0K11000115FMaintenance Group Circuit Breaker is Tripped]
        DEVICE [PL0K1100]  ADDRESS [400/362]
        RAWLOC [PLC_UREAD]  RAWTYPE [ANA]  RAWINDEX [20183]
        LOGLVL [5]  GINDEX [NONE]  GTYPE [NONE]
```

[more...]

11.4 Sample Source Code (Auto Generation)

The complete set of source code is too large to be printed in here (over 50Mbyte).

The following program is part of the executor that processes the physical I/O.

```
#include "gbl.h"
#include "gbl_err.h"
#include "cfg.h"

ANA     ReadIORec(FILE*);
ANA     AddIORec(char[GBL_MAX_DESC_LEN + 1], ANA, ANA, ANA, ANA,
                 ANA, ANA, ANA, ANA);
ANA     LogIOSqlRec(CFG_STRUCT_IO, ANA);
ANA     ConfigIOTags(ANA);


ANA     ReadIORec(FILE* pDataFile)
{
    CFG_STRUCT_ALARM sAlarm[GBL_MAX_MSG_LEN];
    CFG_STRUCT_STATE sState[GBL_MAX_MSG_LEN];

    char    czToken[GBL_MAX_MSG_LEN];
    char    czMsg[GBL_MAX_MSG_LEN];

    char    czIOName[GBL_MAX_DESC_LEN + 1];
    ANA     iIOStateIndex = GBL_UNDEFINED;
    ANA     iIOStateTotal = 0;
    ANA     iIOAlarmTotal = 0;
    ANA     iIOPreLogLvl = GBL_UNDEFINED;
    ANA     iIOGraphicIndex = GBL_UNDEFINED;
    ANA     iIOGraphicType = GBL_UNDEFINED;
    ANA     iIORawLocation = GBL_UNDEFINED;
    ANA     iIORawIndex = GBL_UNDEFINED;
    ANA     iIORawType = GBL_UNDEFINED;
    char    czIODevName[GBL_MAX_DESC_LEN + 1];
    char    czIODevAddress[GBL_MAX_DESC_LEN + 1];
    ANA     iSmgIndex;
    ANA     iPlcTableIndex;
    ANA     iCount;
    ANA     iScreen;
    ANA     iDrawType;
    ANA     iIOVirtualIndex;
    char    czIOVirtualTag[GBL_MAX_TAG_NAME_LEN + 1];

    char    czNdtlTableName[GBL_MAX_MSG_LEN + 1];
    char    czNdtlWriteTrig[GBL_MAX_MSG_LEN + 1];
    char    czNdtlWriteState[GBL_MAX_MSG_LEN + 1];
    char    czNdtlTag[GBL_MAX_MSG_LEN];

    char    czGraphicType[GBL_MAX_TAG_NAME_LEN + 1];
    char    czGraphicTypeTag[GBL_MAX_TAG_NAME_LEN + 1];

    /*--- init records ---*/

    memset(czIOName, '\0', GBL_MAX_DESC_LEN + 1);

    for ( iCount = 0; iCount < GBL_MAX_DESC_LEN; iCount++ )
    {
        sState[iCount].iStateNextIndex = GBL_UNDEFINED;
        sState[iCount].iStatePrevIndex = GBL_UNDEFINED;
        sState[iCount].iStateValue = GBL_UNDEFINED;
        sState[iCount].iStateAlarm = GBL_UNDEFINED;
        memset(sState[iCount].czStateDesc, '\0', GBL_MAX_DESC_LEN);

        sAlarm[iCount].iAlarmIndex = GBL_UNDEFINED;
        sAlarm[iCount].iAlarmGroup = GBL_UNDEFINED;
        memset(sAlarm[iCount].czAlarmTag, '\0', GBL_MAX_TAG_NAME_LEN);
        memset(sAlarm[iCount].czAlarmCond, '\0', GBL_MAX_DESC_LEN);
        sAlarm[iCount].iAlarmLimit = GBL_UNDEFINED;
        memset(sAlarm[iCount].czAlarmDesc, '\0', GBL_MAX_MSG_LEN);
    } /* end-for */

    /*--- get IO description ---*/

    if (GetNextToken(pDataFile, czToken) == KEYWORD_STRING )
    {
        strcpy(czIOName, czToken);
    }
    else
    {
        ReadError(czToken);
    } /* end-if */

    /*--- get State ---*/

    while ( GetNextToken(pDataFile, czToken) == KEYWORD_ID )
    {
        if ( strcmp(czToken, KEYWORD_STATE) == 0 )
        {
            /*---- process state record ---*/
```

```
    if (GetNextToken(pDataFile, czToken) == KEYWORD_STRING )
    {
        /*--- read state value ---*/

        if ( strcmp(czToken, KEYWORD_DEFAULT) == 0 )
        {
            /*--- setup default state records ---*/

            DefaultStateRec(GBL_IO, czIOName, CFG_iIOPtr,
                            sState,
                            &iIOStateIndex,
                            &iIOStateTotal);

            if ( GetNextToken(pDataFile, czToken) == KEYWORD_ID )
            {
                if ( strcmp(czToken, KEYWORD_STATE) != 0 )
                {
                    break;
                }
                else
                {
                    ReadError(czToken);
                } /* end-if */
            }
            else
            {
                ReadError(czToken);
            } /* end-if */
        }
        else
        {
            /*--- add a new state record ---*/

            sState[iIOStateTotal].iStateValue = atoi(czToken);

            /*--- read state description ---*/

            if ( GetNextToken(pDataFile, czToken) == KEYWORD_STRING )
            {
                strcpy(sState[iIOStateTotal].czStateDesc, czToken);
            }
            else
            {
                ReadError(czToken);
            } /* end-if */

            /*--- read state alarm ---*/

            if ( GetNextToken(pDataFile, czToken) == KEYWORD_ID )
            {
                if ( strcmp(czToken, KEYWORD_ALARM) == 0 )
                {
                    sState[iIOStateTotal].iStateAlarm = GBL_ALARM;

                    /*--- read state alarm description ---*/

                    if ( GetNextToken(pDataFile, czToken) !=
                         KEYWORD_STRING )
                    {
                        ReadError(czToken);
                    } /* end-if */

                    /*--- create alarm record ---*/

                    sAlarm[iIOAlarmTotal].iAlarmLimit =
                        sState[iIOStateTotal].iStateValue;
                    strcpy(sAlarm[iIOAlarmTotal].czAlarmDesc,
                           czToken);
                    iIOAlarmTotal++;
                }
                else
                {
                    if ( strcmp(czToken, KEYWORD_NORMAL) == 0 )
                    {
                        sState[iIOStateTotal].iStateAlarm =
                            GBL_NORMAL;
                    }
                    else
                    {
                        ReadError(czToken);
                    } /* end-if */
                } /* end-if */
            }
            else
            {
                ReadError(czToken);
            } /* end-if */

            iIOStateTotal++;
        } /* end-if */
    }
    else
    {
        ReadError(czToken);
    } /* end-if */
}
else
{
    /*--- no or no more state record ---*/

    break;                  /* break the while loop */
```

```
    } /* end-if */
} /* end-while */

/*--- add state records if any ---*/

if ( iIOStateTotal > 0 )
{
    AddStateRec(sState, iIOStateTotal, &iIOStateIndex);

} /* end-if */

/*--- Read Data Source ---*/

if ( strcmp(czToken, KEYWORD_DEV_NAME) == 0 )
{
    if ( GetNextToken(pDataFile, czToken) == KEYWORD_STRING )
    {
        strcpy(czIODevName, czToken);

        /*--- search device index ---*/

        iCount = 0;
        iSmgIndex = GBL_UNDEFINED;
        while ( iCount < CFG_iSmgPtr && CFG_iSmgPtr > 0)
        {
            if ( strcmp( czIODevName, CFG_sSmg[iCount].czSmgName) == 0 )
            {
                iSmgIndex = iCount;
                iPlcTableIndex = CFG_sSmg[iCount].iPlcIndex; /*RC*/
            } /* end-if */

            iCount ++;
        } /* end-while */

        if ( iSmgIndex == GBL_UNDEFINED )
        {
            ReadError(czToken);
        } /* end-if */
    }
    else
    {
        ReadError(czToken);
    } /* end-if */
}
else
{
    ReadError(czToken);
} /* end-if */

/*--- Read Source Address ---*/

if ( GetNextToken(pDataFile, czToken) == KEYWORD_ID )
{
    if ( strcmp(czToken, KEYWORD_DEV_ADDRESS) == 0 )
    {
        if ( GetNextToken(pDataFile, czToken) == KEYWORD_STRING )
        {
            strcpy(czIODevAddress, czToken);
        }
        else
        {
            ReadError(czToken);
        } /* end-if */
    }
    else
    {
        ReadError(czToken);
    } /* end-if */
}
else
{
    ReadError(czToken);
} /* end-if */

/*--- Read Raw Location ---*/

if ( GetNextToken(pDataFile, czToken) == KEYWORD_ID )
{
    if ( strcmp(czToken, KEYWORD_RAW_LOC) == 0 )
    {
        if ( GetNextToken(pDataFile, czToken) == KEYWORD_STRING )
        {
            if ( strcmp(czToken, KEYWORD_PLC_READ) == 0 )
                iIORawLocation = GBL_PLC_READ;
            else if (strcmp(czToken, KEYWORD_PLC_WRITE) == 0 )
                iIORawLocation = GBL_PLC_WRITE;
            else if (strcmp(czToken, KEYWORD_ADA_READ) == 0 )
                iIORawLocation = GBL_ADA_READ;
            else if (strcmp(czToken, KEYWORD_ADA_WRITE) == 0 )
                iIORawLocation = GBL_ADA_WRITE;
            else if (strcmp(czToken, KEYWORD_PLC_UREAD) == 0 )
                iIORawLocation = GBL_PLC_UREAD;
            else
            {
                ReadError(czToken);
            } /* end-if */
        }
        else
        {
            ReadError(czToken);
        } /* end-if */
    }
```

```
    else
    {
        ReadError(czToken);
    } /* end-if */
}
else
{
    ReadError(czToken);
} /* end-if */

/*--- Read Raw Type ---*/

if ( GetNextToken(pDataFile, czToken) == KEYWORD_ID )
{
    if ( strcmp(czToken, KEYWORD_RAW_TYPE) == 0 )
    {
        if ( GetNextToken(pDataFile, czToken) == KEYWORD_STRING )
        {
            if ( strcmp(czToken, KEYWORD_ANALOG) == 0 )
            {
                iIORawType = FL_ANALOG;
            }
            else
            {
                if ( strcmp(czToken, KEYWORD_MESSAGE) == 0 )
                {
                    iIORawType = FL_MESSAGE;
                }
                else
                {
                    ReadError(czToken);
                } /* end-if */
            } /* end-if */
        }
        else
        {
            ReadError(czToken);
        } /* end-if */
    }
    else
    {
        ReadError(czToken);
    } /* end-if */
}
else
{
    ReadError(czToken);
} /* end-if */

/*--- Read Raw Index ---*/

if ( GetNextToken(pDataFile, czToken) == KEYWORD_ID )
{
    if ( strcmp(czToken, KEYWORD_RAW_INDEX) == 0 )
    {
        if ( GetNextToken(pDataFile, czToken) == KEYWORD_STRING )
        {
            if ( strcmp(czToken, KEYWORD_NONE) == 0 )
            {
                iIORawIndex = GBL_UNDEFINED;
            }
            else
            {
                iIORawIndex = atoi(czToken);

                /*--- keep track of largest raw value ---*/

                switch ( iIORawType )
                {
                    case FL_ANALOG:
                        if ( CFG_iRawAnalogTotal < iIORawIndex )
                        {
                            CFG_iRawAnalogTotal = iIORawIndex;
                        } /* end-if */
                    break;

                    case FL_MESSAGE:
                        if ( CFG_iRawMessageTotal < iIORawIndex )
                        {
                            CFG_iRawMessageTotal = iIORawIndex;
                        } /* end-if */
                    break;
                } /* end-switch */
            } /* end-if */
        }
        else
        {
            ReadError(czToken);
        } /* end-if */
    }
    else
    {
        ReadError(czToken);
    } /* end-if */
}
else
{
    ReadError(czToken);
} /* end-if */

/*--- Read log level ---*/
```

```
if ( GetNextToken(pDataFile, czToken) == KEYWORD_ID )
{
    if ( strcmp(czToken, KEYWORD_LOG) == 0 )
    {
        if ( GetNextToken(pDataFile, czToken) == KEYWORD_STRING )
        {
            iIOPreLogLvl = atoi(czToken);

            if ( iIOPreLogLvl < 0 )
            {
                ReadError(czToken);
            } /* end-if */
        }
        else
        {
            ReadError(czToken);
        } /* end-if */
    }
    else
    {
        ReadError(czToken);
    } /* end-if */
}
else
{
    ReadError(czToken);
} /* end-if */

/*--- Read Graphic index ---*/

if ( GetNextToken(pDataFile, czToken) == KEYWORD_ID )
{
    if ( strcmp(czToken, KEYWORD_GINDEX) == 0 )
    {
        if ( GetNextToken(pDataFile, czToken) == KEYWORD_STRING )
        {
            if ( strcmp(czToken, KEYWORD_NONE) == 0 )
            {
                iIOGraphicIndex = GBL_UNDEFINED;
            }
            else
            {
                iIOGraphicIndex = atoi(czToken);
            } /* end-if */
        }
        else
        {
            ReadError(czToken);
        } /* end-if */
    }
    else
    {
        ReadError(czToken);
    } /* end-if */
}
else
{
    ReadError(czToken);
} /* end-if */

/*--- Read Graphic Type ---*/

if ( GetNextToken(pDataFile, czToken) == KEYWORD_ID )
{
    if ( strcmp(czToken, KEYWORD_GTYPE) == 0 )
    {
        if ( GetNextToken(pDataFile, czToken) == KEYWORD_STRING )
        {
            if ( strcmp(czToken, KEYWORD_NONE) == 0 )
            {
                iIOGraphicType = GBL_UNDEFINED;
            }
            else
            {
                iIOGraphicType = atoi(czToken);

                if ( iIOGraphicType != GBL_UNDEFINED)
                {
                    UpdateGTypeTagDim(iIOGraphicType, iIOGraphicIndex);
                } /* end-if */

            } /* end-if */
        }
        else
        {
            ReadError(czToken);
        } /* end-if */
    }
    else
    {
        ReadError(czToken);
    } /* end-if */
}
else
{
    ReadError(czToken);
} /* end-if */

/*--- Add Alarm Record ---*/

for ( iCount = 0; iCount < iIOAlarmTotal; iCount++ )
{
```

```
    switch (iIORawLocation)
    {
        case GBL_PLC_UREAD:
            AddAlarmRec(GBL_OPS, CFG_iIOPtr,
                        sAlarm[iCount].iAlarmLimit,
                        sAlarm[iCount].czAlarmDesc);
            AddAlarmRec(GBL_MMS, CFG_iIOPtr,
                        sAlarm[iCount].iAlarmLimit,
                        sAlarm[iCount].czAlarmDesc);
        break;

        default:
            AddAlarmRec(GBL_IO, CFG_iIOPtr,
                        sAlarm[iCount].iAlarmLimit,
                        sAlarm[iCount].czAlarmDesc);
    } /* end-switch */
} /* end-for */

/*--- Add current record to local structure ---*/

AddIORec(czIOName, iIOStateIndex, iIOStateTotal,
         iIOPreLogLvl, iIOGraphicIndex, iIOGraphicType,
         iIORawLocation, iIORawType, iIORawIndex);

/*--- Generate FLLAN tables ---*/

if ( iIOGraphicType != GBL_UNDEFINED &&
     iIOGraphicIndex >= 0 )
{
    iDrawType = GBL_UNDEFINED;
    SeekDrawByGType(iIOGraphicType, &iDrawType);

    switch (iDrawType)
    {
        case CFG_DIRECT:    /* static objects */
            /*--- config FLLAN ---*/

            SeekGNameByGType(iIOGraphicType, czGraphicType);
            sprintf(czGraphicTypeTag,
                    "%s[%d]", czGraphicType, iIOGraphicIndex);
            AddFllanTag(CFG_SRV_CLI, czGraphicTypeTag,
                        czGraphicTypeTag, CFG_UNIQUE);
        break;

        case CFG_INDIRECT:    /* movable objects */
            SeekGNameByGType(iIOGraphicType, czGraphicType);

            /*--- config FLLAN ---*/

            for ( iScreen = 0; iScreen < CFG_MAX_MOVABLE_SCREEN; iScreen++ )
            {
                sprintf(czGraphicTypeTag,
                        "%s_X%d[%d]", czGraphicType,
                        iScreen, iIOGraphicIndex);
                AddFllanTag(CFG_SRV_CLI, czGraphicTypeTag,
                            czGraphicTypeTag, CFG_UNIQUE);

                sprintf(czGraphicTypeTag,
                        "%s_Y%d[%d]", czGraphicType,
                        iScreen, iIOGraphicIndex);
                AddFllanTag(CFG_SRV_CLI, czGraphicTypeTag,
                            czGraphicTypeTag, CFG_UNIQUE);
             } /* end-for */

             /*--- config MOVCTL table ---*/

             AddMovRec(CFG_sEquip[CFG_iEquipPtr - 1].czEquipName,
                       iIOGraphicType, iIOGraphicIndex);

        break;

    } /* end-switch */

} /* end-if */

/*--- Generate NDTL tables ---*/

if ( iIORawLocation == GBL_PLC_WRITE )
{

    sprintf(czNdtlTableName, "%s%d",
            NDTL_WRITETABLE_TAG, iIORawIndex);

    sprintf(czIOVirtualTag, "%s", NDTL_WRITETRIG_TAG);
    iIOVirtualIndex = iIORawIndex;
    ConvertVirtualTag(czIOVirtualTag, &iIOVirtualIndex);
    sprintf(czNdtlWriteTrig, "%s[%d]",
            czIOVirtualTag, iIOVirtualIndex);

    sprintf(czIOVirtualTag, "%s", NDTL_WRITESTAT_TAG);
    iIOVirtualIndex = iIORawIndex;
    ConvertVirtualTag(czIOVirtualTag, &iIOVirtualIndex);
    sprintf(czNdtlWriteState, "%s[%d]",
            czIOVirtualTag, iIOVirtualIndex);

    AddNdtl_hdrRec(CFG_iNdtl_hdrPtr, czNdtlTableName, "N",
                   1, "", "", "", "",
                   1, czNdtlWriteTrig, "", "", czNdtlWriteState,
                   iPlcTableIndex, GBL_PLC_WRITE);

    switch ( iIORawType )
    {
```

```
            case FL_ANALOG:
                sprintf(czIOVirtualTag, "%s", NDTL_RAW_ANA_TAG);
                iIOVirtualIndex = iIORawIndex;
                ConvertVirtualTag(czIOVirtualTag, &iIOVirtualIndex);
                sprintf(czNdtlTag, "%s[%d]",
                        czIOVirtualTag, iIOVirtualIndex);

                AddNdtl_ovrRec(CFG_iNdtl_ovrPtr, czNdtlTableName,
                               czNdtlTag, CFG_sSmg[iSmgIndex].iPlcIndex,
                               czIODevAddress, NDTL_DATA_TYPE);
            break;

            case FL_MESSAGE:
                sprintf(czIOVirtualTag, "%s", NDTL_RAW_MSG_TAG);
                iIOVirtualIndex = iIORawIndex;
                ConvertVirtualTag(czIOVirtualTag, &iIOVirtualIndex);
                sprintf(czNdtlTag, "%s[%d]",
                        czIOVirtualTag, iIOVirtualIndex);

                AddNdtl_ovrRec(CFG_iNdtl_ovrPtr, czNdtlTableName,
                               czNdtlTag, CFG_sSmg[iSmgIndex].iPlcIndex,
                               czIODevAddress, NDTL_DATA_TYPE);
            break;

            default:
                GBLLog("ReadIORec: cfg_io.c - internal error", GBL_LOG0);
        } /* end-switch */

    } /* end-if */

    if ( iIORawLocation == GBL_PLC_READ )
    {
        sprintf(czNdtlTableName, "%s%d",
                NDTL_READTABLE_TAG, iPlcTableIndex);

        switch ( iIORawType )
        {
            case FL_ANALOG:
                sprintf(czIOVirtualTag, "%s", NDTL_RAW_ANA_TAG);
                iIOVirtualIndex = iIORawIndex;
                ConvertVirtualTag(czIOVirtualTag, &iIOVirtualIndex);
                sprintf(czNdtlTag, "%s[%d]",
                        czIOVirtualTag, iIOVirtualIndex);

                AddNdtl_ovrRec(CFG_iNdtl_ovrPtr, czNdtlTableName, czNdtlTag,
                               CFG_sSmg[iSmgIndex].iPlcIndex, czIODevAddress,
                               NDTL_DATA_TYPE);
            break;

            case FL_MESSAGE:
                sprintf(czIOVirtualTag, "%s", NDTL_RAW_MSG_TAG);
                iIOVirtualIndex = iIORawIndex;
                ConvertVirtualTag(czIOVirtualTag, &iIOVirtualIndex);
                sprintf(czNdtlTag, "%s[%d]",
                        czIOVirtualTag, iIOVirtualIndex);

                AddNdtl_ovrRec(CFG_iNdtl_ovrPtr, czNdtlTableName, czNdtlTag,
                               CFG_sSmg[iSmgIndex].iPlcIndex, czIODevAddress,
                               NDTL_DATA_TYPE);
            break;

            default:
                GBLLog("ReadIORec: cfg_io.c - internal error", GBL_LOG0);
        } /* end-switch */
    } /* end-if */

    if ( iIORawLocation == GBL_PLC_UREAD )
    {
        sprintf(czNdtlTableName, "%s%d",
                NDTL_UREAD_TABLE_TAG, iPlcTableIndex);

        sprintf(czIOVirtualTag, "%s", NDTL_RAW_DIG_TAG);
        iIOVirtualIndex = iIORawIndex;
        ConvertVirtualTag(czIOVirtualTag, &iIOVirtualIndex);
        sprintf(czNdtlTag, "%s[%d]",
                czIOVirtualTag, iIOVirtualIndex);
        AddNdtluovrRec(CFG_iNdtluovrPtr, czNdtlTableName, czNdtlTag,
                CFG_sSmg[iSmgIndex].iPlcIndex + CFG_DEFAULT_USTATION_SEQ,
                czIODevAddress, NDTL_DATA_TYPE);
    } /* end-if */
    return FAIL;
} /* end (ReadIORec) */


ANA    AddIORec(char czIOName[GBL_MAX_DESC_LEN + 1],
                ANA iIOStateIndex, ANA iIOStateTotal,
                ANA iIOPreLogLvl, ANA iIOGraphicIndex,
                ANA iIOGraphicType, ANA iIORawLocation,
                ANA iIORawType, ANA iIORawIndex)
{
    ANA   i;
    ANA   iCheckCt;
    ANA   iStop;
    ANA   iStop2;
    char  czString[GBL_MAX_MSG_LEN];

    iStop = 1;
    iStop2 = 1;
    /*--- copy io info to local structure ---*/

    CFG_sIO[CFG_iIOPtr].iIOIndex = CFG_iIOPtr;
```

```
/** EW - 8 Oct 98 Check for duplicated GINDEX, GTYPE **/
    if (iChosen == 1)
    {
      for (iCheckCt = 0; (iCheckCt < CFG_iIOPtr) && (iStop == 1); iCheckCt++)
      {
       if ( iIOGraphicIndex == CFG_sIO[iCheckCt].iIOGraphicIndex &&
            iIOGraphicType == CFG_sIO[iCheckCt].iIOGraphicType &&
            iIOGraphicIndex != GBL_UNDEFINED)
       {
         sprintf(czString, "Duplicated GINDEX [%d] GTYPE[%d] RawIndex[%d]\n",
                 iIOGraphicIndex, iIOGraphicType, iIORawIndex);
         GBLLog(czString, GBL_LOG0);
         iStop = 0;
       }
      }

/** EW - 9 Oct 98 Check for duplicated RawIndex **/
     for (iCheckCt =0; (iCheckCt < CFG_iIOPtr) && (iStop2 == 1); iCheckCt++)
     {
      if (iIORawIndex == CFG_sIO[iCheckCt].iIORawIndex &&
          iIORawType == CFG_sIO[iCheckCt].iIORawType )
      {
        sprintf(czString, "Duplicated RawIndex[%d] RawType[%d]\n",
                iIORawIndex, iIORawType);
        GBLLog(czString, GBL_LOG0);
        iStop2 = 0;
      }
     }
    }

/* RC - 27Mar98 - */
  /* strcpy(CFG_sIO[CFG_iIOPtr].czIOName, czIOName); */
  /*-------------------------------------------------------------------------*/
  /* Old code above did not check czIOName length, which imposes error. */
  /* The loop at the following limits czIOName length to 15 characters. */
  /*-------------------------------------------------------------------------*/
    for ( i = 0; i < 15; i++ )
    {
        CFG_sIO[CFG_iIOPtr].czIOName[i] = czIOName[i];
    }
    czIOName[i] = '\0';
/* - RC */

    CFG_sIO[CFG_iIOPtr].iIONextIndex = GBL_UNDEFINED;

    if ( CFG_sEquip[CFG_iEquipPtr - 1].iEquipChildTotal == 0 )
    {
        CFG_sEquip[CFG_iEquipPtr - 1].iEquipChildIndex = CFG_iIOPtr;
        CFG_sIO[CFG_iIOPtr].iIOPrevIndex = GBL_UNDEFINED;
    }
    else
    {
        CFG_sIO[CFG_iIOPtr - 1].iIONextIndex = CFG_iIOPtr;
        CFG_sIO[CFG_iIOPtr].iIOPrevIndex = CFG_iIOPtr - 1;
    } /* end-if */

    CFG_sEquip[CFG_iEquipPtr - 1].iEquipChildTotal++;

    CFG_sIO[CFG_iIOPtr].iIOParentIndex = CFG_iEquipPtr - 1;
    CFG_sIO[CFG_iIOPtr].iIOStateIndex = iIOStateIndex;
    CFG_sIO[CFG_iIOPtr].iIOStateTotal = iIOStateTotal;
    CFG_sIO[CFG_iIOPtr].iIOPreLogLvl = iIOPreLogLvl;
    CFG_sIO[CFG_iIOPtr].iIOGraphicIndex = iIOGraphicIndex;
    CFG_sIO[CFG_iIOPtr].iIOGraphicType = iIOGraphicType;
    CFG_sIO[CFG_iIOPtr].iIORawLocation = iIORawLocation;
    CFG_sIO[CFG_iIOPtr].iIORawIndex = iIORawIndex;
    CFG_sIO[CFG_iIOPtr].iIORawType = iIORawType;


    CFG_iIOPtr++;

} /* end (AddIORec) */


ANA    LogIOSqlRec(CFG_STRUCT_IO sIO, ANA iIOCnt)
{
    static LANA    iSeq = 10000;
           char    czSeq[GBL_MAX_MSG_LEN];

    /*--- insert record ----*/

    FormatSeqNum(iSeq, czSeq);
    GenerateNextSeq(&iSeq);

    fprintf(CFG_hSql, "insert into %s%d(\\\n", TABLE_IO, iIOCnt);
    fprintf(CFG_hSql, "    DOMAIN, TABLE_NBR, IO_INDEX, NAME,\\\n");
    fprintf(CFG_hSql,
       "    NEXTINDEX, PREVINDEX, PARNTINDEX,\\\n");
    fprintf(CFG_hSql,
            "    STATEINDEX, STATETOTAL, PRELOGLVL, GRPINDEX,\\\n");
    fprintf(CFG_hSql,
            "    RAW_LOC, RAW_INDEX, RAW_TYPE, GRP_TYPE) VALUES\\\n");
    fprintf(CFG_hSql, "    ('SHARED', '%s', '%d', '%s',\\\n",
            czSeq, sIO.iIOIndex, sIO.czIOName);
    fprintf(CFG_hSql, "    '%d', '%d', '%d',\\\n",
            sIO.iIONextIndex, sIO.iIOPrevIndex,
            sIO.iIOParentIndex);
    fprintf(CFG_hSql, "    '%d', '%d', '%d', '%d',\\\n",
            sIO.iIOStateIndex, sIO.iIOStateTotal,
            sIO.iIOPreLogLvl, sIO.iIOGraphicIndex);
    fprintf(CFG_hSql, "     '%d', '%d', '%d', '%d');\n\n",
```

```
            sIO.iIORawLocation, sIO.iIORawIndex,
            sIO.iIORawType, sIO.iIOGraphicType);

} /* end (LogIOSqlRec) */


ANA    ConfigIOTags(ANA iTagDim)
{
    UpdateObjectDim("IOName", iTagDim, GBL_SERVER);
    UpdateObjectDim("IONextIndex", iTagDim, GBL_SERVER);
    UpdateObjectDim("IOPrevIndex", iTagDim, GBL_SERVER);
    UpdateObjectDim("IOParentIndex", iTagDim, GBL_SERVER);
    UpdateObjectDim("IOChildIndex", iTagDim, GBL_SERVER);
    UpdateObjectDim("IOChildTotal", iTagDim, GBL_SERVER);
    UpdateObjectDim("IOStateIndex", iTagDim, GBL_SERVER);
    UpdateObjectDim("IOStateTotal", iTagDim, GBL_SERVER);
    UpdateObjectDim("IOStateValue", iTagDim, GBL_SERVER);
    UpdateObjectDim("IOStateName", iTagDim, GBL_SERVER);
    UpdateObjectDim("IORawLocation", iTagDim, GBL_SERVER);
    UpdateObjectDim("IORawIndex", iTagDim, GBL_SERVER);
    UpdateObjectDim("IORawType", iTagDim, GBL_SERVER);
    UpdateObjectDim("IORawValue", iTagDim, GBL_SERVER);
    UpdateObjectDim("IOAlarmState", iTagDim, GBL_SERVER);
    UpdateObjectDim("IOCurLogLevel", iTagDim, GBL_SERVER);
    UpdateObjectDim("IOPreLogLevel", iTagDim, GBL_SERVER);
    UpdateObjectDim("IOGraphicType", iTagDim, GBL_SERVER);
    UpdateObjectDim("IOGraphicIndex", iTagDim, GBL_SERVER);
    UpdateObjectDim("GraphicAnaValue", iTagDim, GBL_SERVER);
    UpdateObjectDim("GraphicMsgValue", iTagDim, GBL_SERVER);
    UpdateObjectDim("RawDigital", CFG_iRawAnalogTotal, GBL_SERVER);
    UpdateObjectDim("RawAnalog", CFG_iRawAnalogTotal, GBL_SERVER);
    UpdateObjectDim("RawAnalogLink", CFG_iRawAnalogTotal, GBL_SERVER);
    UpdateObjectDim("RawMessage", CFG_iRawMessageTotal, GBL_SERVER);
    UpdateObjectDim("RawMessageLink", CFG_iRawMessageTotal, GBL_SERVER);
    UpdateObjectDim("OPS_ACK", CFG_iUReadTotal, GBL_SERVER);
    UpdateObjectDim("OPS_ACK", CFG_iUReadTotal, GBL_CLIENT);

} /* end (ConfigIOTags) */
```