RC 20652 (12/06/96)
Computer Science/Mathematics

# IBM Research Report

## PRODUCTION RESOURCE MANAGER: BUILDING AND APPLYING FRAMEWORKS FOR MANUFACTURING DECISION SUPPORT

Walter C. Dietrich, Jr., Brenda L. Dietrich,
Tom Ervolina, J.P. Fasano, Robin Lougee-Heimer,
Elizabeth J. Poole, Jung-Mu Tang, Robert H. Wang,
Robert J. Wittrock, and Danny C. Wong

IBM Research Division
T. J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Goodwin R. Chin

System Software Associates
4000 Executive Parkway, Suite 275
San Ramon, CA 94583



IBM Research Division
Almaden • T.J. Watson • Tokyo • Zurich

# Production Resource Manager: Building and Applying Frameworks for Manufacturing Decision Support

**Walter C. Dietrich, Jr., Brenda L. Dietrich, Tom Ervolina, J.P. Fasano, Robin Lougee-Heimer, Elizabeth J. Poole, Jung-Mu Tang, Robert H. Wang, Robert J. Wittrock, and Danny C. Wong**

IBM Corporation
T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

**Goodwin R. Chin**

System Software Associates, Inc.
Suite 275
4000 Executive Parkway
San Ramon, CA 94583

## Abstract

Decision-support software is often used to solve manufacturing logistics problems. The data may come from many different sources; the techniques for synthesizing the data and communicating the solution depend on the problems being solved and the users' business processes, resulting in a need for unique applications. This paper describes our experience using a framework that provides reusable design and implementation components for decision support applications.

## Introduction

Developing manufacturing logistics decision support software is a labor-intensive and costly endeavor. Except for the underlying solver software (such as linear-programming libraries and constraint-based solvers), most applications are developed as one-off solutions. This paper describes the Production Resource Manager (PRM) Framework, which encapsulates the systemic similarities among manufacturers, allowing software developers to readily extend framework components into applications that address specific manufacturing requirements. The PRM Framework contains frameworks which provide 1) rapid GUI development, 2) access to a variety of manufacturing information system data, 3) business objects for representing manufacturing logistics problems, and 4) transformation objects for mapping manufacturing logistics problems to mathematical solution methods.

This paper describes the overall framework architecture, with an emphasis on important use cases [Jacobsen92], responsibilities [Wirfs-Brock90], design patterns [Gamma94], and object class hierarchies. The PRM Framework objects represent the manufacturing logistics domain and provide a reusable architecture for application development, allowing developers to quickly write new applications with a proven base design.

## Background and Design Objectives

In the late 1980's IBM used a three-phase manufacturing planning approach. Business planning was done at the corporate level, Material Requirements Planning (MRP) was done monthly in a single corporate-wide run, and detailed planning was done at each manufacturing plant. The IBM printed circuit card plant experienced shortages of components used in several different PS/2 personal computer cards. IBM's planning process required the card plant to commit an availability schedule for the cards within a few days of receiving card requirements and component availability projections. The card planners had only spreadsheets to help them allocate the scarce components. The Enterprise Shortfall Implosion Tool (ESIT) was implemented in 1990 to provide allocation capability [Dietrich90]. The term implosion conveyed the idea of bottom-up planning, in contrast with the traditional top-down explosion-based planning. The ESIT tool was intended to run once in each monthly planning cycle. ESIT's output, a feasible committed schedule for finished PS/2 cards, was used to finalize the PS/2 box plan. Planners quickly began exploring other uses for ESIT. It was used frequently for "what if" analysis of changes in supply or demand. More "implosion-like" allocation problems surfaced, so ESIT was made available to planners throughout IBM; however, differences between site MRP systems and manufacturing policies made roll-out difficult. Adapting ESIT to address multi-level build processes required significant code modifications.

With the increased availability of RISC workstations, specialized optimization libraries, and GUIs, it became clear that a change in tool development philosophy was needed. In order to fully realize the benefits of implosion, two issues had to be addressed: our software must be easy to integrate with a wide variety of MRP systems, and developing new applications must be easy and fast. The entire software system was redesigned using object technology. This new generation of implosion technology is architected to permit rapid development of customized applications and rapid interface with data sources. Implosion applications, ranging from final assembly scheduling to world-wide allocation of scarce components, are being deployed throughout IBM manufacturing.

## Framework Architecture

We used a very general model for solving real-world problems mathematically. Using a two-step refinement process, real-world problems are transformed into mathematical models which can be solved to find solutions for the real-world problems. In the first step, the real world is modeled using domain-specific concepts. In the second step, this domain-specific model is transformed into a mathematical model. Once the mathematical model is solved, its result is translated into a solution in the domain model, which is then transformed into a solution in the real world. Corresponding data transformations must also be made, and they can be quite complicated.

The PRM Framework supports this modeling paradigm by providing frameworks for representing the manufacturing domain and for transforming data as it enters and leaves the framework. The PRM Framework contains two top-level frameworks: the Data Interface Framework (DIF Framework) and the Scenario Framework.

## Scenario Framework

As shown in the figure on the next page, the Scenario Framework is a layered system that consists of three frameworks: the Core Framework, the Application Framework (with facilities for combining, sorting, and filtering data), and the User Interface Framework. The User Interface Framework supports the development of both interactive, graphical user interfaces and batch-, file-, or database-oriented user interfaces, but it is not discussed further in this paper.

The most important class in the PRM Framework is the `Scenario` class. It provides the glue that makes everything else work together. For any tractable real-world problem there is a domain model that represents the problem and its solution. The `Scenario` classes are responsible for (1) representing the domain model, (2) populating the domain model with data from the real world, (3) transforming the domain model to and from the mathematical model, and (4) providing a mapping back to the real world. These responsibilities are fulfilled with the help of other classes. Generally, each problem has its own `Scenario` subclass. Each `Scenario` instance represents one instance of a problem and its data.

### Core Framework

The Core Framework contains fundamental classes in the manufacturing domain, such as capacities, materials, and demands, and classes that wrap the solver and cache data that will be sent to the solver. Capacities and materials are represented by objects called `Parts`. Customer orders and forecasts are represented by objects called Demands. Every instance of a Demand or Part, whether it is used by the solver or not, is in the UniversalSetOfDemands (USOD) or UniversalSetOfParts (USOP), respectively.

The most interesting classes in the Core Framework are the LastProblemSolved (LPS) and Working–Problem (WIP) classes. The LPS class provides a wrapper [Dietrich89] for the Solver subsystem. In other words, it provides an object-oriented interface to the Solver subsystem. It represents the last set of inputs that was given to the solver (the "problem") and the last solution generated by the solver. Because it represents both the problem and the solution, it allows clients to see a consistent picture of the model. The WIP class allows new problems to be created (generally by editing old ones). It represents the original input (and all changes made to that input) to the solver . The LPS and WIP classes work together. When a client of the WIP class asks for problem data that hasn't been changed, the WIP class gets it from the LPS class. When a new solution is needed, the LPS class gets all of the changes from the WIP class and puts them into the Solver subsystem. Because the WIP class is closely related to the LPS class, the WIP provides the same level of abstraction that the LPS class provides.

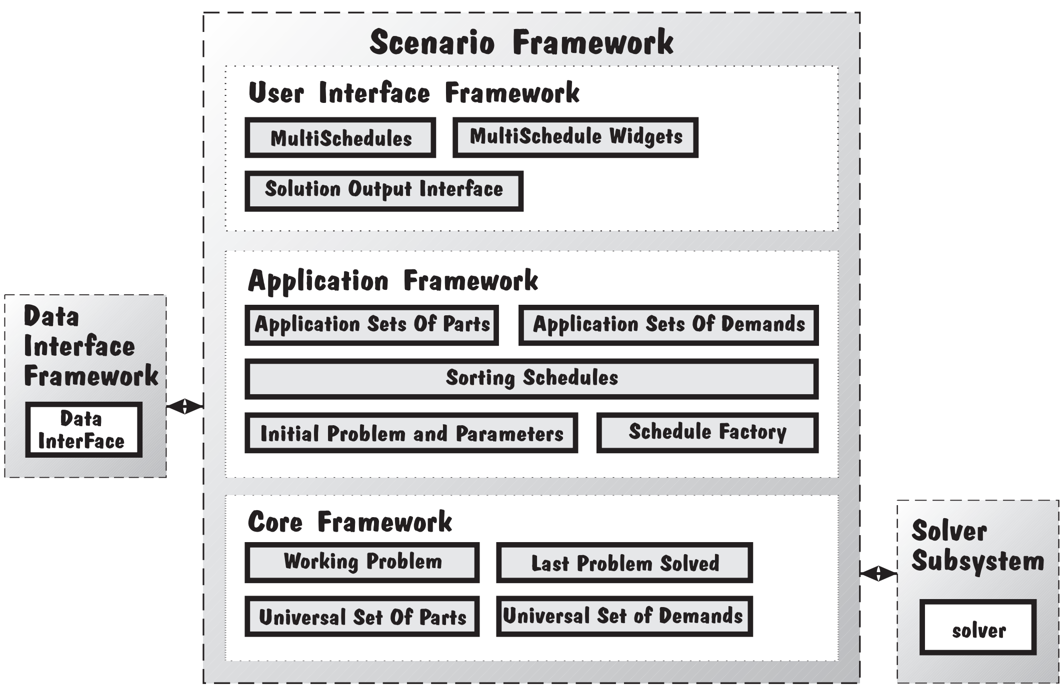
## Application Framework

The classes in the Application Framework provide a higher level of abstraction. These classes help the Scenario class represent the domain model of the manufacturing logistics problem.

When a Scenario instance is created, the domain model and the solver model must be populated. This is the responsibility of the InitialProblemAnd–Parameters (IPP) class. The IPP uses the DIF Framework to get data and put it into the model. IPP is designed as a base class. When writing an application to solve a manufacturing problem, an application developer derives from the IPP class an application-specific IPP class.

### *Schedule Framework*

In the Manufacturing Logistics domain, a spreadsheet view of part and demand data is a very powerful mechanism. In this view, each column corresponds to one time period and each row corresponds to a part or demand. Because this view of the data is powerful for users (such as planners, materials managers, and customer representatives), manufacturing



information systems and decision support algorithms, the PRM Framework provides a flexible Schedule Framework for manipulating and viewing part or demand attributes.

The base class in the Schedule Framework is the `Schedule` class, which consists of `ItemTimeVec` instances. (A `TimeVec` represents a vector of floating-point numbers with one number for each time period in the problem.) Operations on schedules include insertion, array subscripting, and value retrieval based on `Item`. To support application development, the Schedule Framework must provide more functionality. For instance, a PRM application might need a schedule called *SortedPartInventorySchedule* which is sorted by part name, and contains the supply quantity for each part that has non-zero inventories. To provide this functionality, the framework has a `SortingSchedule` (`SortSchedule`) class which contains a representation of the attribute definitions, the sorting criteria, and the filtering criteria for the schedule. To make this representation flexible with respect to these definitions and criteria, the Schedule Framework makes heavy use of the Strategy pattern [Gamma94]. The Schedule Framework has an `AttributeStrategy` class hierarchy, which implements attribute definition strategies for accessing and updating a variety of attributes such as demand volume, shipment volume, backlog volume, supply volume, and shortage volume. The `AttributeStrategy` class hierarchy also contains classes that allow new schedules to be constructed from other ones. The framework also has a `PairCompareStrategy` class hierarchy, which provides sort criteria such as alphabetically by part name, and a `FilterStrategy` class hierarchy, which provides a variety of selection criteria such as “non-zero” `TimeVec` objects and regular expression matching.

A wide variety of schedules can be created by combining instances of `AttributeStrategy`, `PairCompareStrategy`, and `FilterStrategy`. To encapsulate the knowledge needed to make schedules (and simplify the use of the Schedule Framework), there is a `ScheduleFactory` class that has methods that take a `ScheduleDescriptor` and return a `SortSchedule` that uses the appropriate strategies. Automated updating of schedules is enabled by using the subscription pattern. By extending the subscription pattern, objects can be both subscribers and observers.

### Class Hierarchy for Scenario Framework

Each application is built by using some parts of the framework as components and by creating subclasses of other parts of the framework. The `SortSchedule` class is an example of a component, while the `ScheduleFactory`, `InitialProblemAndParameters`, and `Scenario` classes are examples of base classes. Because it is difficult to create and initialize Scenario Framework objects correctly using constructors alone, the Builder and Abstract Factory patterns are used to instantiate all of the objects in the Core Framework and the Application Framework, and to establish the relationships among these objects.

## Data Interface Framework

As mentioned earlier, many different manufacturing applications can be developed using the PRM framework. Most manufacturing companies use either inhouse or commercial MRP/CRP/ERP systems (hereafter referred to as manufacturing information systems) regularly to plan and track their production. These companies have made large investments in these systems and in keeping the data in them up-to-date. PRM must access data in these systems.

The Data Interface Framework is responsible for providing a consistent interface to manufacturing information systems, isolating the rest of the PRM Framework from the specific manufacturing information system being used.

### Design Criteria

Because many different manufacturing information systems are in use, the Data Interface Framework should allow the rest of PRM to access data the same way from any manufacturing information system. In addition, it should be easy to extend the DIF Framework to support new manufacturing information systems. Since a wide variety of applications can be written using PRM, and since it is impossible to anticipate the information that will be required by a new application, it should be easy to extend the DIF Framework to support new attributes.

In addition to these primary requirements, there are two secondary requirements. It should be possible to use the DIF Framework without using the rest of PRM (to maximize code reuse) and it should be possible to implement interfaces to manufacturing information systems without requiring a specific programming language. Permitting the use of different languages provides flexibility in the implementations and maximizes the pool of developers available.

### DIF Framework Design

The basic design, which uses the Bridge pattern, is shown in the figure on the next page. Two main classes are defined in this pattern, `DataInterface` (`DIF`) and `DataInterfaceImplementation` (`DifImp`). `DIF` is the abstract base class which provides the interface between manufacturing information systems and applications. The detailed implementation is defined in the `DifImp` class, which is hidden from application developers. The `DifImp` class contains a dispatch table (`Dictionary`) which maps attribute names to function pointers, allowing the functions to be implemented in legacy languages. `DIF` clients use the `Attribute` method to access data in the manufacturing information system.

Our design provides significant flexibility. (1) New attributes can be added easily. (2) Clients of `DIF` can find out what attributes are supported by a given `DifImp`. (3) The `DifImp` programmers can write all of the access functions in any language that can be called from "C," a very important feature when accessing legacy systems. This design also provides reusability: when writing PRM applications for a manufacturing information system that is only slightly different from a previously-used system, most of the same functions can be reused.

In general, application developers only need to instantiate the `DIF` and `DifImp` base classes. The dispatch table mechanism provided in the `DifImp` base class is flexible and powerful enough for most manufacturing information systems and most applications. When more flexibility is needed, subclasses can be derived for `DIF`, `DifImp`, or both.

Overall, the DIF Framework is a flexible, modifiable, extendable, and reusable framework with following features:

- Gives the user complete control of the data access methodology and allows easy integration with other parts of PRM;
- Allows the Data Interface design and implementation to be reused with both PRM and non-PRM applications;
- Allows each application to access a variety of manufacturing information systems without changing or recompiling the application code and allows run-time substitution of data access functions (using the `Dictionary`).

## Developing a PRM Application

Once the initial requirements are known, one or more use cases are developed. A model of the problem is defined mathematically and is implemented by a new subclass of `Scenario`. A new subclass of `InitialProblemAndParameters` implements the mapping to the solver and uses the `LastProblemSolved` class to populate the solver's internal model.

A new subclass of `ScheduleFactory` implements the mapping back to the domain model, reusing many existing attribute, sort, and filter strategies (such as supply volume, part name pair-compare, and zero-time-vector filter). The vast majority of the application logic is put into the Application Framework and is simply triggered by the GUI.

## Conclusions

A few key design concepts made the PRM Framework successful. The data interface classes separate the I/O from the problem being solved so that any application can be used at different sites by replacing only the data interface implementation classes. This also allows a site to use many applications communicating with a single set of databases through a single data interface implementation. The scenario architecture handles the transformation to and from the manufacturing logistics domain to the mathematical one.

We made heavy use of Use Cases for analysis, Responsibility-Driven Design for high-level design and Design Patterns for detailed design. We extended known patterns for additional functionality, which aided both our thought processes and our intra-team communication. We found that using spiral development with heavy user involvement was essential.

A variety of applications have been written using the PRM Framework. The Framework's performance overhead is low. The DIF Framework has allowed us to use data from a variety of sources. The Scenario structure supports the reuse of a fundamental design by many applications. The PRM Framework has successfully met its objectives.

## Bibliography


Dietrich89 - W. Dietrich, L. Nackman, and F. Gracer, "Saving a Legacy with Objects", *Proc. OOPSLA'89,* New Orleans: ACM Press 1989, 77-83

Dietrich96 - B. Dietrich, IBM Technical Research Report RC 20389, "Applications of Implosion in Manufacturing", February, 1996.

Gamma94 - E. Gamma, R. Helm, R. Johnson, and J. Vlissides., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, October, 1994.

Jacobsen92 - I. Jacobson, M. Christerson, P. Jonsson, and G. Oevergaard, *Object-Oriented Software Engineering, A Use Case Driven Approach*, Addison-Wesley, 1992.

Wirfs-Brock90 - R. Wirfs-Brock, B. Wilkerson, and L. Wiener, *Designing Object-Oriented Software*, Prentice Hall, Englewood Cliffs, NJ, 07632, 1990