6042977	Supply Volume	10	10	10	15	15	20	20
	Usage Volume	5	10	15	15	10	10	10
32G3917	Supply Volume	5	15	10	15	10	0	0
	Usage Volume	5	15	10	15	10	0	0

FIGURE 12. Tabular Representation of Sample `MultiSchedule`

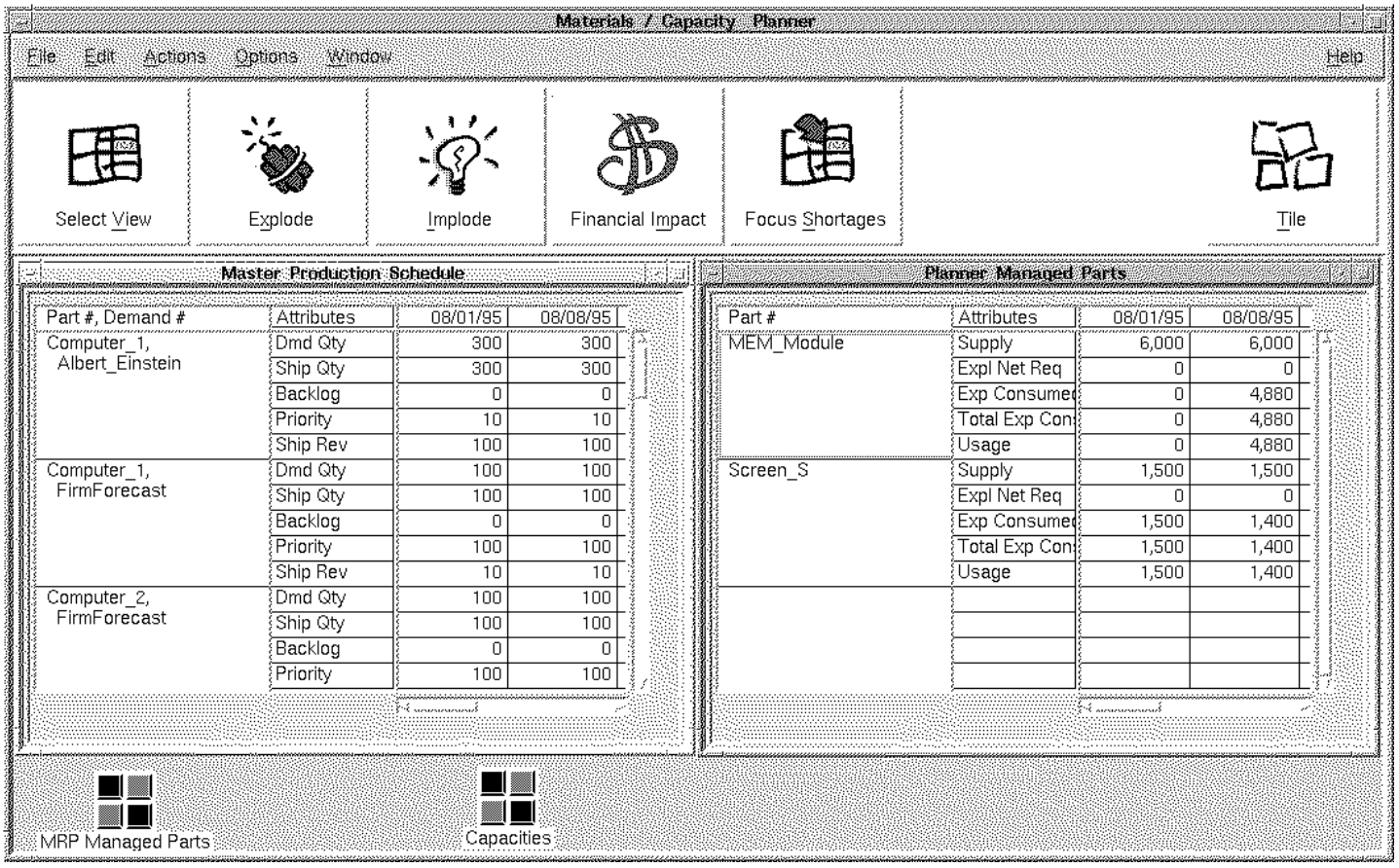

FIGURE 13. CMRP Screen
Two of the Multi Schedule Widgets are iconified (minimized).

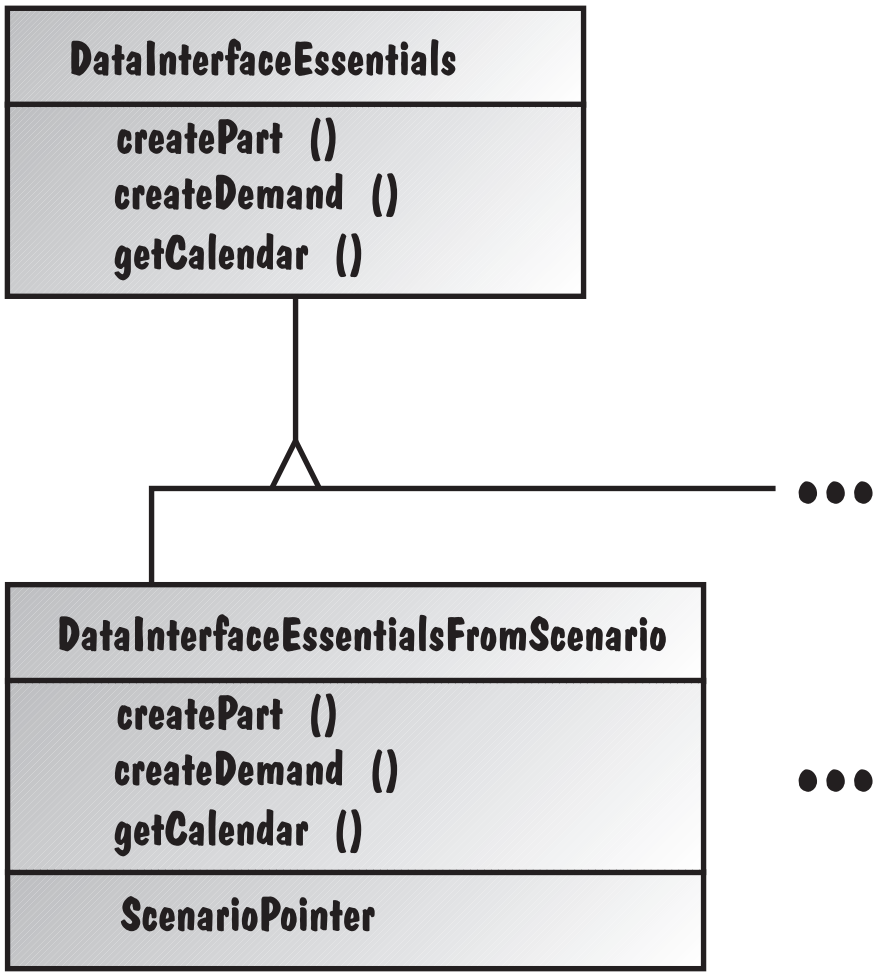

FIGURE 10. DIF Essentials

Attribute	Pointer to Function
Demand Quantity	getDemandQuantity()
Priority	getDemandPrioity()
Revenue	getDemandRevenue()

FIGURE 11. `DataInterfaceImplementation` **Table**
This shows part of the `DifImp` table for demand information.

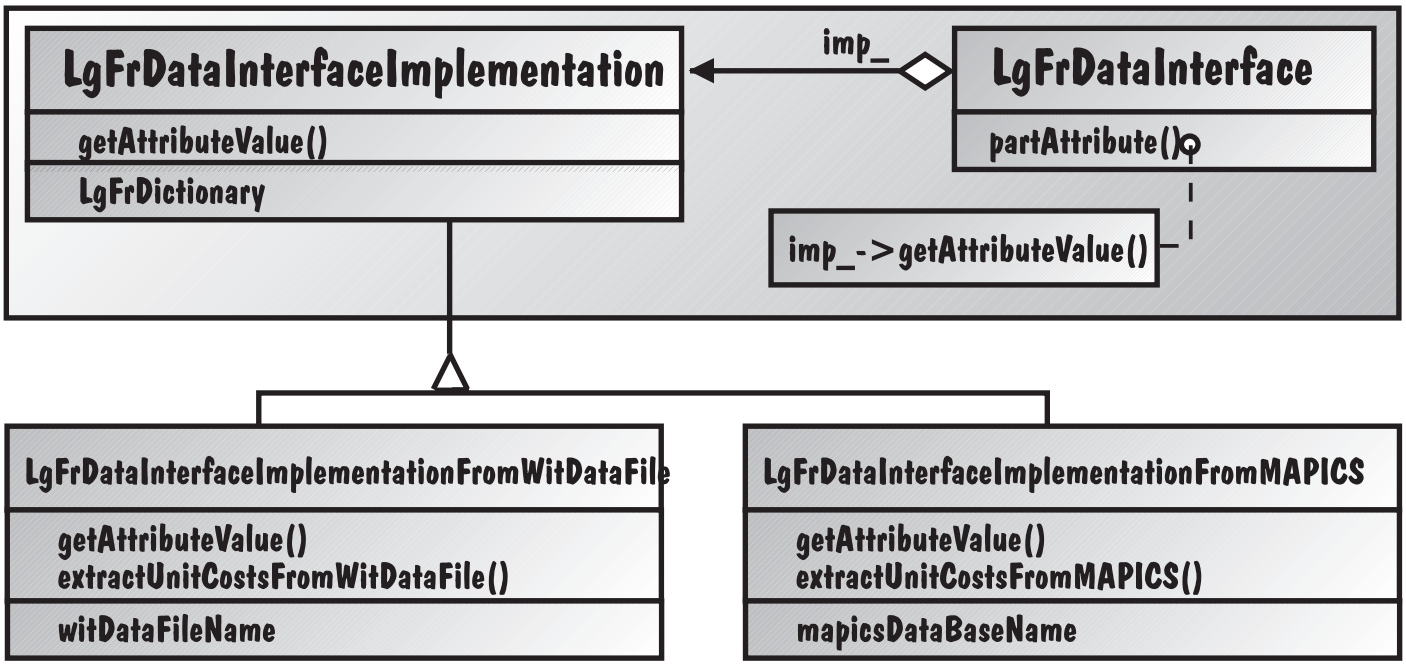

FIGURE 9. Interaction Diagram for Data Interface Classes

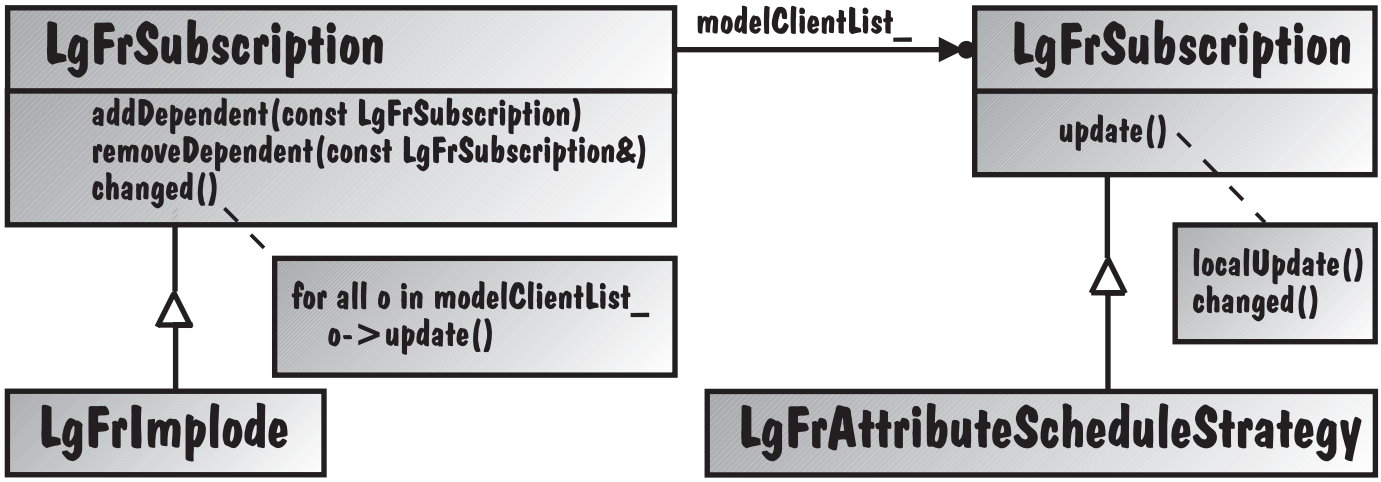

FIGURE 7. Observer Pattern

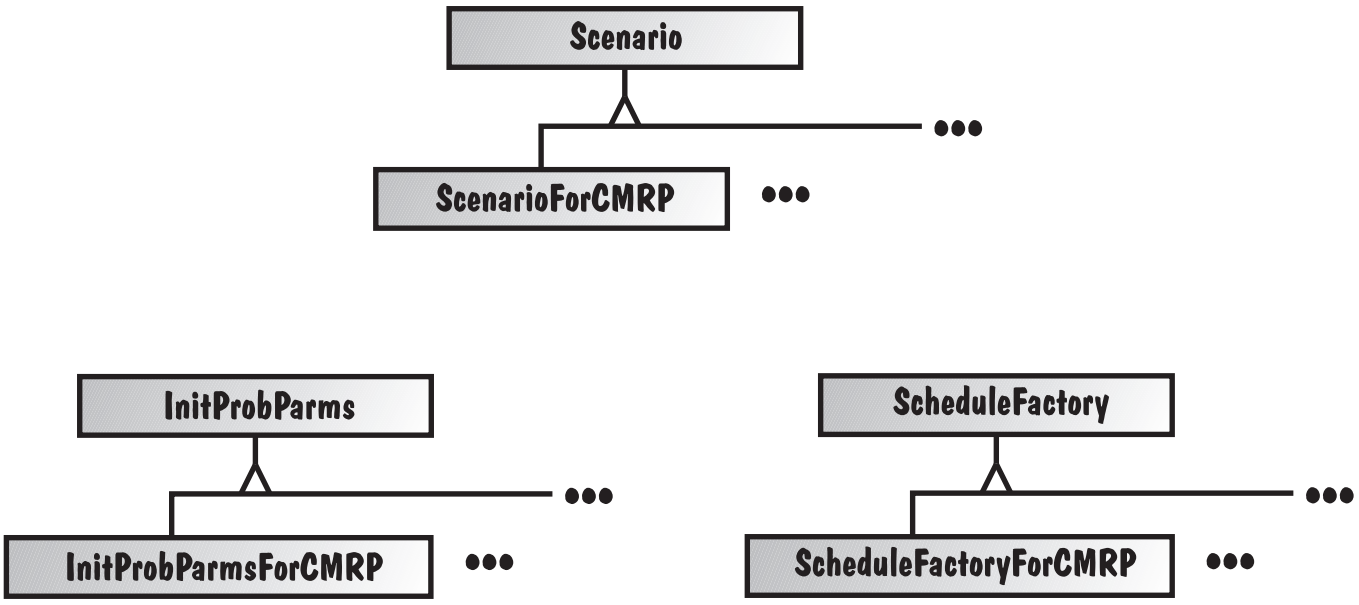

FIGURE 8. Class Heirarchies that Vary by Application
Except for the TIA application, each application has its own subclass of `Scenario`, `InitialProblemAndParameters`, `ScheduleFactory`. TIA uses `Scenario`, Ini`InitialProblemAndParameters`, and `ScheduleFactory` without subclassing them.

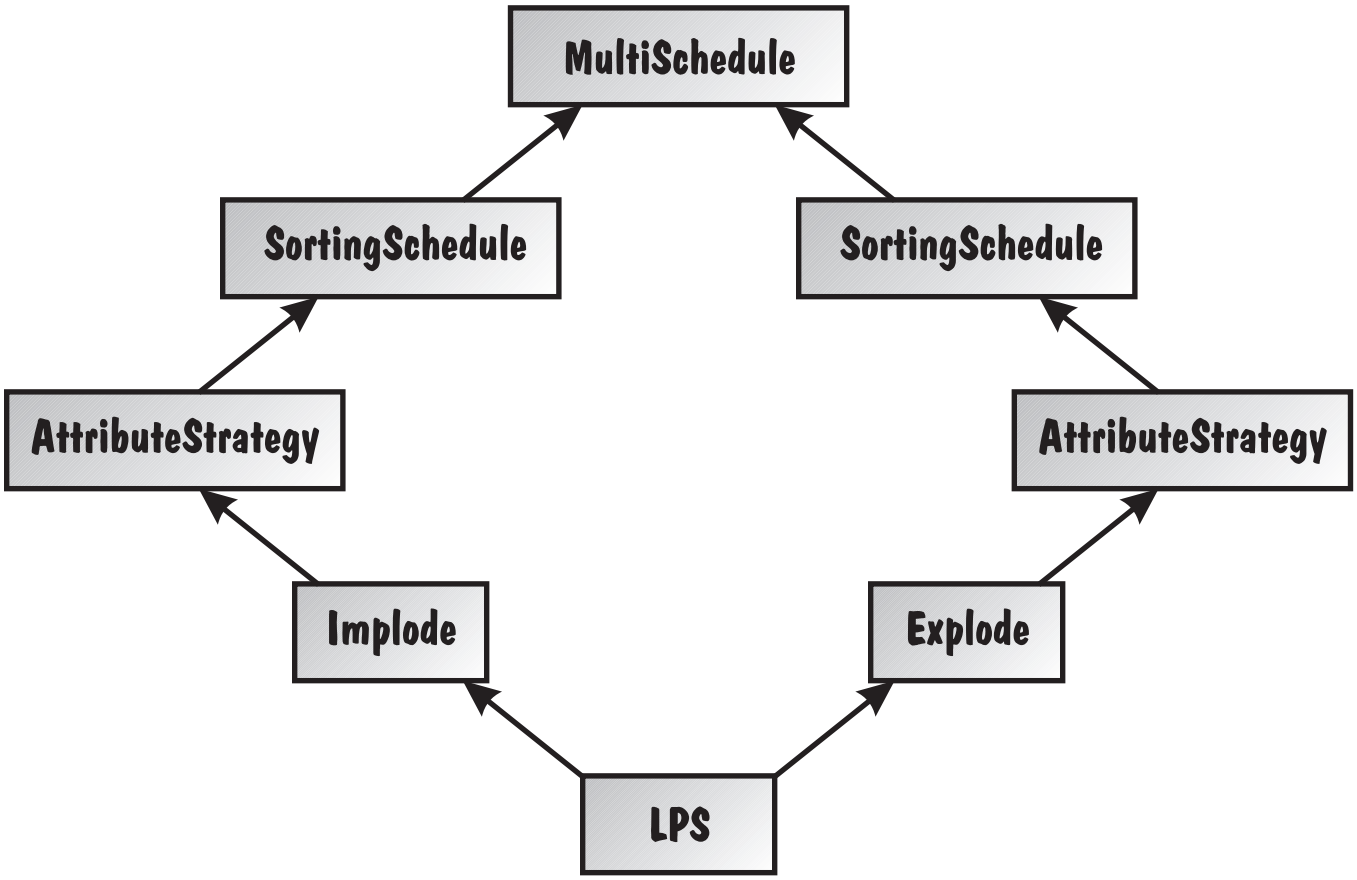

FIGURE 6. Subscription Chain
The object at the head of each arrow subscribes to the object at the tail of the arrow. After the LPS calls the Solver's implode function, a series of notifications occur. These cause the changes in the LPS to propagate through the objects on the left side of the figure all the way to the Multi Schedule object. A similar sequence occurs after the LPS calls the Solver's explode function.

6042977	10	10	10	15	15	20	20
32G3917	5	15	10	15	10	0	0
92G7235	0	0	20	20	15	10	6
04H6922	50	75	40	50	80	95	100

FIGURE 4. Tabular Representation of Sample Schedule
The data in this schedule represents supply volumes.

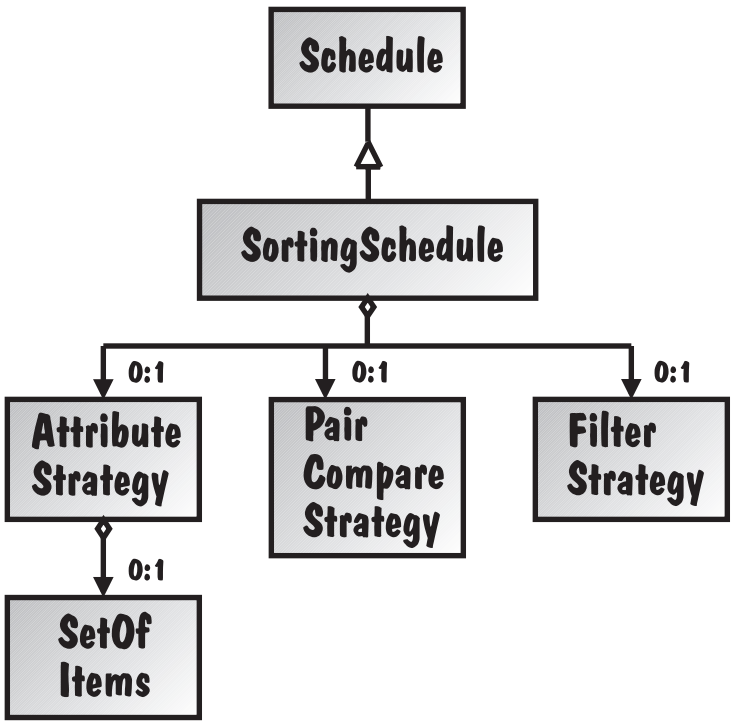

FIGURE 5. Schedule Framework Class Diagram
The Schedule framework consists of the schedule class irarchy (`Schedule`, and `SortingSchedule`) and the `AttributeStrategy`, `PairCompareStrategy`, and `FilterStrategy` class hierarchies.

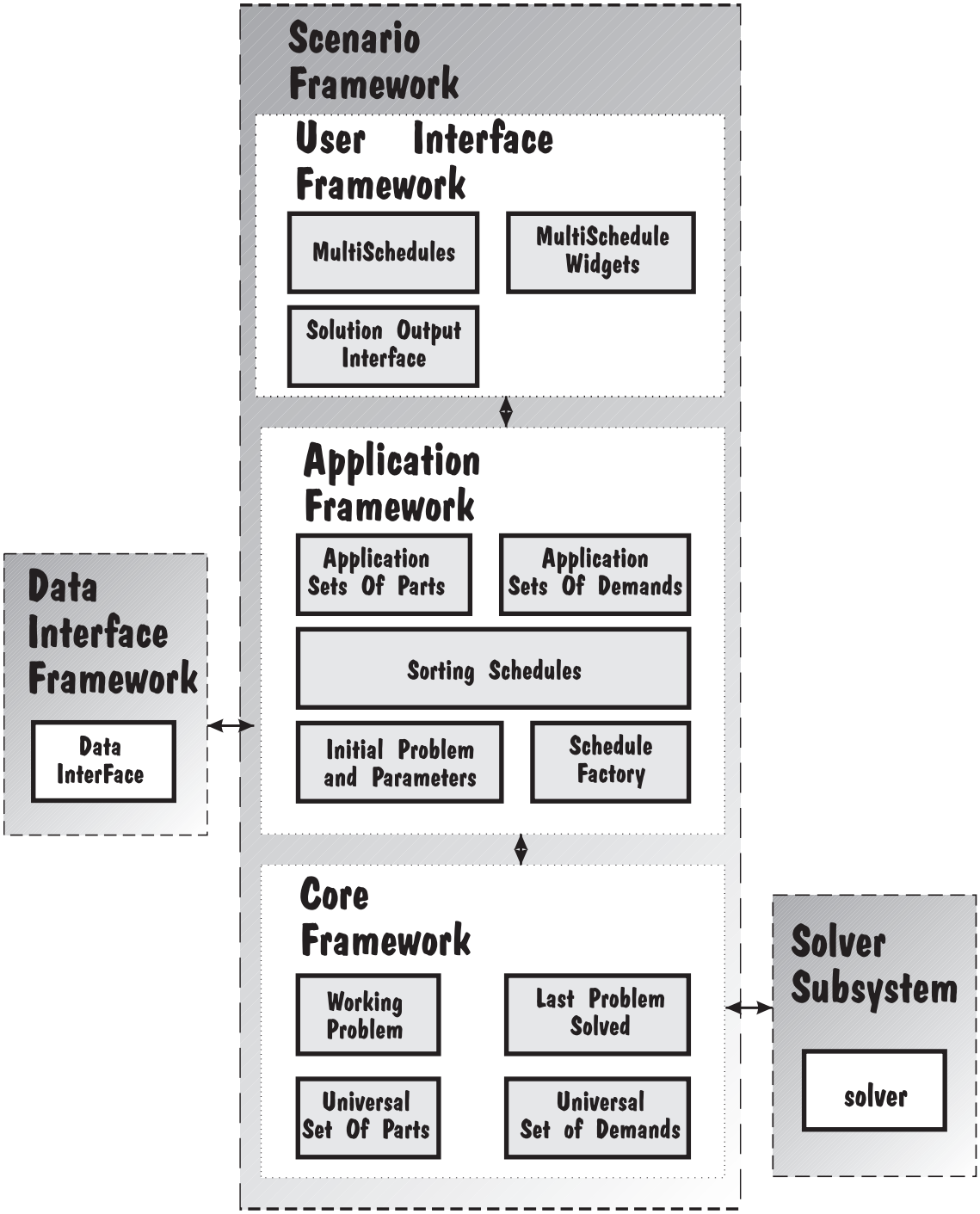

FIGURE 3. Framework Overview
The PRM framework consists of several framework, some of which are nested. It uses a solver subsystem.

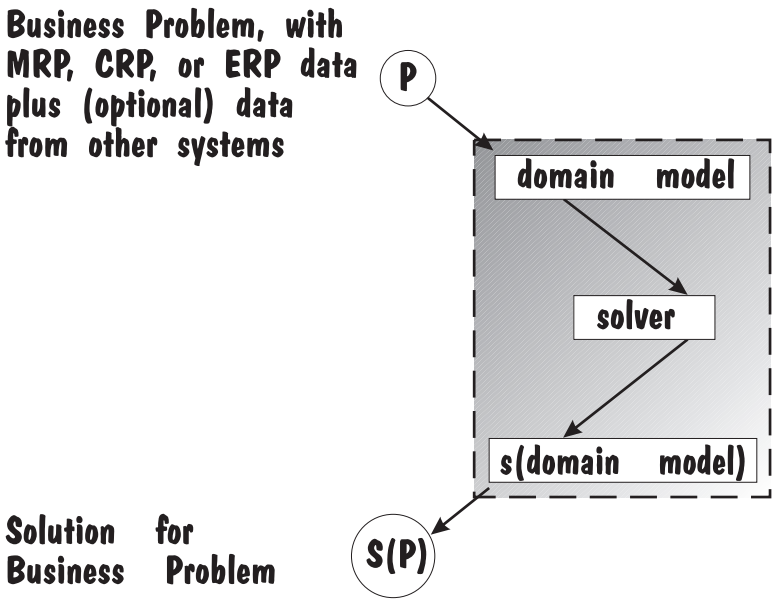

FIGURE 2. Manufacturing Logistics Modeling Process
This figure shows how manufacturing logistics problems can be changed into mathematical models via a two-step process. The data in the MRP, CRP, or EPS is the real-world data and the problem is a business problem that uses the MRP/CRP/ERP data and possibly other data.

Figures and Tables

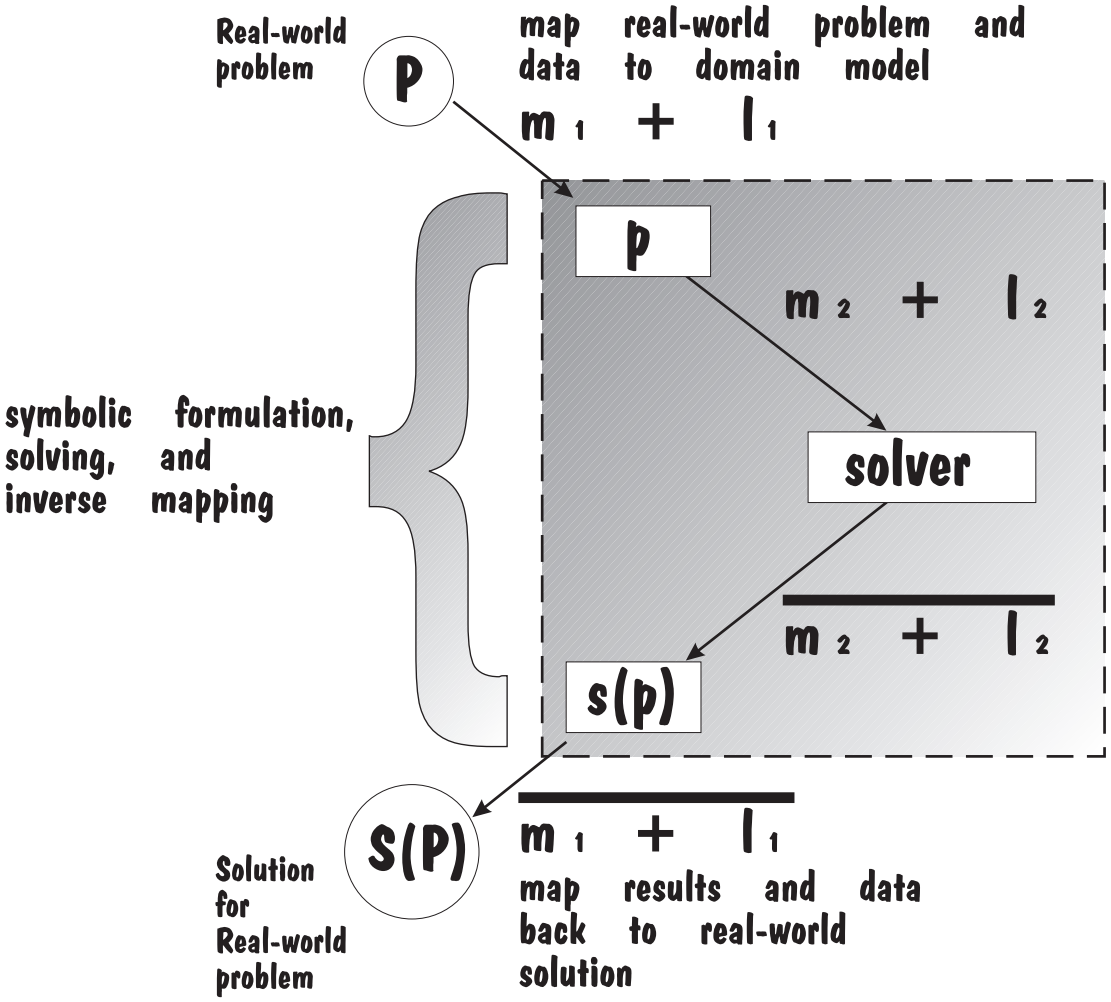

FIGURE 1. Modeling Process
This figure shows how real-world problems can be changed into mathematical models via a two-step process

Bibliography

Alpern89 - B. Alpern, R. Hoover, B. Rosen, P. Sweeny, and K. Zadeck, “Incremental Evaluation of Computational Circuits”, Research Report RC 14755, IBM Research Center, Yorktown Heights, NY, 1989

Apics95- J. Cox III, J. Blackstone Jr., and M. Spencer, *The APICS Dictionary*, 8th edition, 1995, American Production and Inventory Control Society, Inc.

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.

Mapics93 - *AS/400 Manufacturing Accounting and Production Information Control System Extended Advantage (MAPICS XA) Master Production Schedule Planning User’s Guide,* MAPICS, Inc., Atlanta, GA, 1993.

Sap94 - *SAP R/3 System Architecture (BC010),* SAP AG, Walldorf, Germany, 1994.

Vollmann92 - T. Vollmann, W. Berry, and D. Whybark, *Manufacturing Planning and Control Systems*, 3rd edition, Richard D. Irwin, Inc.,Homewood, IL,1992.

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

8.0 Conclusions

The design concepts that made the PRM Framework successful are the following. The data interface classes separate the I/O from the problem being solved so that any application can be used at different sites by only replacing the data interface implementation classes. It also allows a site to use many applications via only one set of data bases and one data interface implementation. The scenario architecture handles the transformation to and from the manufacturing logistics domain to the mathematical one.The spreadsheet/tabular GUI has proven to be flexible for developers and comfortable and intuitive for end users.

We made heavy use of Use Cases for analysis, Responsibility-Driven Design for high-level design and Design Patterns for detailed design. They 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.

The model of the problem was defined mathematically and was implemented by a new subclass of `Scenario` called `ScenarioForCMRP`. This subclass contained `ApplicationSetOfParts` (`ASOP`) instances for capacities, constrained parts, and unconstrained parts. A new subclass of `InitialProblemAndParameters` implemented the mapping to the solver. This subclass got data from the DIF Framework. It used the `LastProblemSolved` class to populate the solver's internal model. It also initialized the `ASOP` instances.

A new subclass of `ScheduleFactory` implemented the mapping back to the domain model. `ScheduleFactoryForCMRP` reused many existing attribute, sort, and filter strategies (such as supply volume, part name pair-compare, and zero-time-vector filter). The reason so many strategies were reused was that `ScheduleFactoryForCMRP` associated the right instance of `ASOP` with each schedule. For example, the schedule for *Constrained Part Supply Volume* was constructed by reusing `SupplyVolumeAttributeStrategy` in a schedule that used the constrained `ASOP`.

The GUI used `MSWidget` instances for the MPS, Capacity, Constrained Parts, and Unconstrained Parts windows. The GUI used the `ScheduleFactoryForCMRP` class to insert CMRP-specific schedules into the appropriate `MultiSchedule` instances. For example, the GUI inserted the *Constrained Part Supply Volume* schedule into the Constrained Parts `MSWidget`.

The vast majority of the application logic was put into the Application Framework and was simply triggered by the GUI. For example, when the user clicked on the *Implode* button (the light bulb button), the GUI called the `ScenarioForCMRP` class's `implode` method. The `implode` method caused the solver to compute a solution that took all constraints into account. Then the Subscription mechanism caused the output schedules and the multi-schedules to update themselves automatically. When the `implode` method returns, the GUI updated the data on the screen.

Except for new subclasses for the Builder and Abstract Factory patterns that control Scenario initialization [Gamma94], most of the new code in TIA was GUI code. The `MSWidget` classes were reused without change. Most of the development effort was spent with users, deciding what Implosion Engine options to expose through the GUI, what schedules to display, how many `MSWidgets` to display, and how to subdivide the data. TIA development required less than two person-weeks.

7.2 The CMRP Application

The CMRP application is used by procurement planners when the availability of either manufacturing capacity or some purchased materials is constrained. Standard planning methods can generate purchase orders that exceed known availability limits. Furthermore, if the materials that are readily available are purchased without regard to the availability of the short materials, then large inventories of unusable parts can be created. By producing an MPS that respects capacity and material availability limits, the CMRP application generates purchase orders for unconstrained materials so that those materials arrive just in time for use.

Using either demand priorities or economic factors to drive the resource allocation decisions, CMRP quickly calculates a feasible, optimal MPS, which accounts for availability of constrained materials, procurement lead-times of unconstrained materials, manufacturing capacity, and demand. It determines a procurement plan for each unconstrained material, and identifies the most critical resources. This schedule can be viewed with CMRP's GUI, (Figure 13, "CMRP Screen") and the user can perform "what-if" analysis.

7.2.1 Developing the Application

Once the initial requirements were known, a model of the problem was defined. A transformation was developed to map this model to and from the solver. The data elements that were not in traditional manufacturing information systems were defined. The general appearance of the GUI was described and the multi-schedules that would be used were specified.

able per period and quantity consumed per period. See Figure 12, “Tabular Representation of Sample MultiSchedule.” The `MultiSchedule` class makes it easier to use the information stored in schedules by combining schedules in different ways. The `MultiSchedule` class contains a set of schedules. Due to the large volumes of data that planners must deal with, the `MultiSchedule` class supports sorted and filtered views of the data. Schedules in a `MultiSchedule` can be chosent to be “selection schedules” (used for filtering out uninteresting data), and “order schedules” (used for sorting by part name, demand name, quantity, shortage, etc.). `MultiSchedule` provides iterators that can traverse by item (for each item, iterate over all the attributes), and by attribute (for each attribute, iterate over all the items).

The `MultiScheduleWidget`(`MSWidget`) collaborates with the `MultiSchedule` to provide GUI support for PRM applications. In this collaboration, the `MSWidget` is responsible for managing and updating the display of the data, and the `MultiSchedule` is responsible for sorting, filtering, and generally managing the schedules that are displayed.

7.0 Two PRM Applications

Several applications have been written using the PRM Framework and they interface to a variety of manufacturing information systems. TIA (The Implosion Application) is the simplest application and is used to demonstrate the Implosion Engine and prototype new models. CMRP (Capacitated MRP) is used by master production schedulers and materials managers to plan production and purchasing. Other applications have been written for scheduling daily production, reducing inventories, and giving customers optimal ship-date quotes.

7.1 The TIA Application

TIA was written after two other applications. TIA was written because analysts needed an easy way to experiment with Implosion Engine models and data. The domain model (see Figure 2, “Manufacturing Logistics Modeling Process”) in TIA is exactly the same as the solver model. Although most applications require subclasses of `Scenario`, `IPP`, and `ScheduleFactory`, TIA uses them directly.

Access functions from multiple `DifImp`'s can be mixed and matched to create `DifImp`'s for hybrid manufacturing information systems. (Such hybrid information systems occur when manufactures want to use best-of-breed systems or when they are moving from legacy information systems to new ones.)

5.4 Data Interface Framework Summary

In general, application developers only need to instantiate the `DIF` and `DifImp` base classes. In addition, the general attribute table mechanism provided in the generic `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 either or both of these classes.

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

- Grant the customer total control to manage the data access methodology and allow easy integration with other parts of PRM;
- Allow the DIF Framework design and implementation to be reused with both PRM and non-PRM applications;
- Allow one application to access a variety of manufacturing information systems without changing or recompiling the application code and even allow run-time substitution of data access functions using the flexible attribute table managing mechanism.

6.0 User Interface Framework

The User Interface Framework supports the development and implementation of user interfaces for batch and interactive applications. Because the domain model and solver interface logic is implemented in the Scenario Framework, user interfaces developers can concentrate on usability and human factors. The User Interface Framework contains a class for saving data in flat files and a hierarchy of classes for displaying or saving attributes grouped by part or demand.

A `MultiSchedule` contains one or more sets of related schedules. For example, a `PartMultiSchedule` might contain all parts needed by a specified assembly, and for each part might include the attributes for quantity avail-

it does not provide an obvious way to add new attributes and allow implementations to be written in any language. Satisfying those requirements requires invention.

5.3 DIF Framework Design

Figure 9, “Interaction Diagram for Data Interface Classes,” shows the basic design. Two main classes are defined in this structure, `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. `DIF` clients use the `Attribute` method to access data in the manufacturing information systems.

Although the DIF Framework is separated from its clients (for example, the Scenario Framework), the DIF Framework calls client methods to create parts and demands and access calendar information. If the DIF Framework called the Scenario Framework methods directly, it would be impossible to use the DIF Framework with non-PRM applications, limiting reusability. This problem can be solved by using the Bridge pattern a second time, but an important aspect of good pattern usage is to only use them when needed. This problem can be solved by simply using inheritance. See Figure 10, “DIF Essentials.”

As shown in Figure 9, “Interaction Diagram for Data Interface Classes,” the `DIF` instance uses a `DifImp` instance to access a manufacturing information system. The attribute the client is accessing is passed to the `DIF` instance and the `DifImp` instance uses a table (see Figure 11, “DataInterfaceImplementation Table”) to look up a function that will access that attribute in the manufacturing information system.

This 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.” This last feature is important when accessing legacy systems. This design also provides a lot of reusability. (1) 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. (2)

large investments in building or acquiring these systems and keeping the data in them up to date. PRM must access data in these systems.

5.1 Design Criteria

Because many different manufacturing information systems are in use, the Data Interface (DIF) Framework should allow the rest of PRM to access data the same way from any manufacturing information system. For example, the Application Framework should be able to use exactly the same method and parameters to read in supply volumes whether the data is coming from R3 or Trident (two ERP systems). 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 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 the primary requirements described above, 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.

5.2 Design Alternatives

There are many alternative approaches that could be used to implement the DIF Framework. One straightforward approach is to use simple inheritance. With this approach, an abstract class is used to define the interface and a different concrete subclass interfaces to each manufacturing information system. This would be difficult to modify and extend. Another approach is to use double dispatching, but this is not extensible enough either.

Satisfying the design criteria requires a design which decouples interface from implementation, provides extensibility, and hides the details of the manufacturing information systems. The Bridge pattern [Gamma94] does this but

trigger an update in the AttributeStrategy object, 3) changes in the AttributeStrategy object trigger an update in the SortSchedule object, 4) changes in the SortSchedule object trigger an update in the Multi-Schedule object.

The Subscription base class is implemented using a modified Observer pattern [Gamma94] as shown in Figure 7, “Observer Pattern.” The Subscription mechanism is different from the Observer pattern because Subscription instances can be both subject and observer. This is necessary because many times an object can act both as observer and subject (e.g. a SortSchedule object can be an observer of an Implode object and the subject of a Sched-uleSensitiveStrategy object).

When a subject’s state changes, the observers must be informed. Doing this efficiently is a nontrivial problem because classes in the Schedule Framework allow many-to-many relationships between subjects and observers. See [Alpern89] for a discussion of this problem and good solutions.

4.4 Class Hierarchy for Scenario and related classes

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 class is an example of a base case. See Figure 8, “Class Heirarchies that Vary by Application.” 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 relationship between these objects.

5.0 Data Interface Framework

As mentioned earlier, many different manufacturing applications can be developed using the PRM framework. Most manufacturing companies use either in-house or commercial MRP/CRP/ERP systems (herein after referred to as manufacturing information systems) regularly to plan and track their production. These companies have made

chy also contains classes that allow new schedules to be constructed from other ones. For example, there are `AttributeStrategy` classes that allow one to subtract two schedules or to sum each row of a schedule. The framework also has a `PairCompareStrategy` class hierarchy, which provides sort criteria such as part name or `TimeVec` quantity. The framework also has a `FilterStrategy` class hierarchy, which provides a variety of selection criteria such as "non-zero" `TimeVec` objects and regular expression matching. The `SortSchedule` class contains optional references to instances of the above strategy objects. To make a *SortedPartInventorySchedule*, one instantiates a `SortSchedule` object and associates the object with the `SupplyVolumeAttributeStrategy`, the `NamePairCompareStrategy`, and the `NonZeroTimeVecFilterStrategy`.

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. To make the above *SortedPartInventorySchedule*, one only has to call the `ScheduleFactory`'s `newPartSchedule` method with the *SortedPartInventorySchedule* instance of `ScheduleDescriptor`. When the *SortedPartInventorySchedule* is made, it is populated with supply volumes that are taken from the `WIP` object. If a supply volume in the `WIP` object changes, the data in the *SortedPartInventorySchedule* should change too. The Subscription Mechanism is used to do this.

4.3 The Subscription Mechanism

The Subscription mechanism provides the dynamic update mechanisms necessary to keep schedule objects up to date when problem-related data changes (e.g. either the user changes an input quantity, or the solver creates a new result). Any class in the framework that has values that may change dynamically (based on the values in other objects) inherits from the `Subscription` class. Figure 6, "Subscription Chain," shows an example of a simulation triggered update. After `LPS` calls the solver to calculate a new implosion result, the following events occur: 1) the presence of new data in the `LPS` object triggers an update in the `Implode` object, 2) changes in the `Implode` object

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. Application-specific IPP classes interact with the DIF Framework to populate LPS, USOD and USOP objects with manufacturing data through a feedInto method. Besides populating objects in the Core Framework, the IPP class is also responsible for initializing any other objects the Scenario object needs.

4.2.1 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. For example, the planned inventory in a whole factory is a table containing the volume of each part in each time period. See Figure 4, “Tabular Representation of Sample Schedule.” 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. 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 Sorting-Schedule (SortSchedule) class which contains a representation of the attribute definitions, the sorting criteria, and the filtering criteria for the schedule. See Figure 5, “Schedule Framework Class Diagram.” 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 hierar-

UniversalSetOfDemands (USOD) or UniversalSetOfParts (USOP), respectively. The most interesting classes in the Core Framework are the LastProblemSolved (LPS) and WorkingProblem (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 to the solver and all changes made to that input. 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.

The Core Framework also has Calendar and TimeVec classes. Calendar represents the time period being considered and the number of subperiods in the problem. TimeVec represents a vector of floating-point numbers with one number for each subperiod in the problem

4.2 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.

Many Scenario subclasses need proper subsets of the parts and demands contained in the UniversalSetOfParts and UniversalSetOfDemands instances, so there are ApplicationSetOfParts (ASOP) and ApplicationSetOfDemands (ASOD) classes that may be contained in each Scenario instance. For example, an ASOD instance might contain all of the demands from preferred customers.

When a Scenario instance is created, the domain model and the solver model must be populated. This is the responsibility of the InitialProblemAndParameters (IPP) class. The IPP uses the DIF Framework to get

tion-level algorithms. The User Interface Framework supports the development of both interactive, graphical user interfaces and batch, file- or data-base-oriented user interfaces. It has facilities for combining, sorting, and filtering data in the Application Framework so that the development of user interfaces is simpler. The User Interface Framework is designed to be used in conjunction with GUI-builders for interactive applications.

The PRM Framework depends on a solver subsystem. The current PRM solver subsystem, the PRM Implosion Engine, is a layered object-oriented system [Dietrich96]. Because the Implosion Engine started out as a C program and continues to be used in non-object-oriented-systems, it is accessed through a procedural Application Programming Interface (API).

4.0 Scenario Framework

The most important class in the PRM Framework is the `Scenario` class. It provides the glue that makes everything else work together. As shown in Figure 2, “Manufacturing Logistics Modeling Process,” 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 it’s own `Scenario` subclass. Each `Scenario` instance represents one instance of a problem and its data. The Scenario Framework contains a low-level framework called the Core Framework and higher-level framework called the Application Framework and User Interface Framework.

4.1 Core Framework

The Core Framework contains fundamental classes in the manufacturing domain, such as capacities, materials, and demands (sales orders and forecasts), 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

model. Once the mathematical model is solved, the inverse of step two transforms the result into a solution in the domain model and the inverse of step one transforms it into a solution in the real world. Corresponding to steps one and two are two steps in which information in the real world is transformed into data in the domain-specific model and then transformed into data in the mathematical model. The solver's result must also be transformed into information that can be used in the real world. These transformations can be quite complicated, depending on both the overall problem being solved and the state of the subproblem the users are interested in.

Figure 2, "Manufacturing Logistics Modeling Process," shows this meta-model applied to manufacturing logistics decision support. MRP/CRP/ERP[1] systems have replaced the real world[2] and the domain has been changed to manufacturing logistics. The PRM Framework supports this by providing frameworks for representing the manufacturing domain and for transforming data as it enters and leaves the framework. As Figure 3, "Framework Overview" shows, the PRM Framework contains two top-level frameworks: the Data Interface Framework and the Scenario Framework. The Data Interface Framework is responsible for providing an interface to the manufacturing information system that is the same regardless of which manufacturing information system is being used. It isolates the rest of the PRM Framework from the manufacturing information system.

The Scenario Framework is a layered system that consists of three frameworks: the Core Framework, the Application Framework, and the User Interface Framework. The Core Framework provides the lowest-level domain-specific classes in the PRM Framework. These consist of the fundamental objects in the domain, specialized collections of fundamental objects, objects that wrap [Dietrich89] the solver, and objects that cache data that will be sent to the solver. The Application Framework is responsible for representing the domain-specific model of the problem being solved. It contains classes that represent attributes of the capacities, materials, and demands that are relevant to the problem being solved, classes that transform these attributes to and from data in the solver, classes that can be used to combine these attributes in various ways, and classes that automate the construction of applica-

1. ERP (Enterprise Resource Planning) systems generally provide MRP and CRP functionality and also support many other business processes, such as payroll and accounting.

2. Although the fidelity between the data in an manufacturing information system and the real world varies, our goal is to help manufacturers make the best decisions possible using the data they already maintain.

committed schedule for finished PS/2 cards, would be used at a corporate planning meeting to finalize the PS/2 box plan.

Planners in Austin quickly began exploring other uses for ESIT. The explosion module was used frequently for "what if" analysis of changes in supply or demand. New output reports were created to provide additional decision support capability (such as reports on short components that showed the affected card volumes). More "implosion-like" problems surfaced in areas such as outsourcing of production and use of substitute parts.

ESIT was presented to material planners at other IBM manufacturing sites. Differences between the site MRP systems and the site manufacturing policies made corporate-wide roll-out difficult. Adapting ESIT to address plants with multi-level build processes (e.g., components to cards to computers) required significant code modifications.

With the increased availability of RISC workstations and optimization libraries specifically developed for them, it became clear to the research team that a change in development philosophy was needed. In order to fully realize the benefits of implosion, we have found that two issues must be addressed. Since there are no dominant MRP systems, our software must be easy to integrate with a wide variety of MRPs. In addition, many manufacturing companies have unique environments and philosophies, so developing different implosion applications must be easy.

The entire software system has been redesigned using object technology. This new generation of implosion technology is architected to permit rapid development of customized applications. Implosion applications, ranging from final assembly scheduling to world-wide allocation of scarce components, are being deployed throughout IBM manufacturing.

3.0 Framework Architecture

Figure 1, "Modeling Process," shows a very general way of solving real-world problems using mathematical solvers. 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

ysis tools are available to aid in this process. These tools identify each plan inconsistency (e.g., shortage), trace the requirements for a scarce resource to the higher-level assemblies that use it, and through the use of MRP/CRP emulation, allow the planner to evaluate the quality of alternative plans. These tools are not truly schedulers because they are only capable of evaluating a proposed schedule. They cannot generate a feasible MPS, let alone determine a schedule that is both feasible and optimal with respect to economic factors.

The third, or execution, phase includes both detailed scheduling of manufacturing processes and management of the intricacies of shipping and purchasing. Fundamental here is the necessity of re-planning as the unpredictable occurs. As orders are cancelled, material shipments arrive late, equipment fails, and product development is delayed, manufactures must re-plan quickly to remain competitive.

Experience within IBM

In the late 1980's IBM used a variation of the three phase approach in which the business planning phase was done at the corporate level, the MRP was done monthly in a single corporate wide run, and the detailed planning was done locally at each manufacturing plant. The corporate MRP system did not consider the availability of materials, or the limited capacity at some of IBM's semiconductor plants. The IBM printed circuit card plant in Austin was experiencing shortages of components for PS/2 personal computer cards. Each scarce component was used in several different cards and each card was used in one or more PS/2 models. IBM's world-wide material planning process required the card plant to commit an availability schedule for PS/2 cards within a few days of receiving card requirements and component availability projections. The card volume planners had no tools, other than simple spreadsheets, to aid them in allocating the scarce components to cards. The Enterprise Shortfall Implosion Tool (ESIT), which provided allocation capability was implemented for Austin in 1990 [Dietrich90]. The term implosion was used to convey the idea that this tool provided bottom-up planning, rather than the traditional top-down explosion-based planning used in MRP. ESIT was implemented in PL/I on MVS. IBM's corporate logistics group designed and wrote the data interfaces to be compatible with IBM's corporate planning systems. IBM's research division implemented the implosion module. An explosion module was also included in ESIT. The ESIT tool was intended to run once, or perhaps twice, in each monthly planning cycle. The output of ESIT's implosion, a feasible

new models and software tools for manufacturing resource allocation. For a discussion of the underlying mathematics and the solution methods see [Dietrich96].

Current Practice

There are three distinct phases in most firm's manufacturing planning processes: (1) overall manufacturing planning, (2) detailed capacity and material planning to support the overall plan, and (3) execution planning to realize the overall plan [Vollmann92].

In the first phase, the company determines quarterly or monthly production plans for a horizon of two to seven years using aggregated capacity, supply and demand information. From this process, a master production schedule (MPS) is generated which incorporates the known customer demand.

The second phase, detailed planning, uses the MPS and inventory data to determine the time-phased requirements for material and capacity. The material requirements planning process (MRP) uses product bill of material information, including "quantity-per" and "offset", together with "net and explode" iterative logic, to determine the amount of raw material that must be acquired in each period. Capacity requirements planning (CRP) is quite similar to MRP, except that the netting operation does not permit the carry-over of unused capacity from one period to the next. Although MRP and CRP logic is fairly simple, the number of calculations required can be enormous. For this reason, until the 1990's MRP was typically a batch process, run on a regular "re-generation" schedule. With the increased computing power available in today's desktop machines, MRP and CRP can be run much more frequently.

One commonly cited shortcoming of both MRP and CRP is that neither process considers the availability of resources (material, capacity) to be constrained. They simply subtract the quantity that is available from the quantity that is required, and generate a list of net requirements. The consideration of constraints is relegated to the master production scheduling process which must take into account "backlog, availability of material, availability of capacity, management policy, and goals" [Apics95], and resolve competing demands for limited resources. The MPS has often been identified as a critical but poorly addressed issue in manufacturing planning. Several data anal-

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 new application development. This allows developers to quickly write new applications with a proven base design. As a demonstration of the framework, this paper also briefly describes the construction of two PRM applications.

The organization of this paper is as follows. Section 2.0 describes the manufacturing logistics domain, and summarizes the design objectives of the PRM Framework. Section 3.0 presents the PRM Framework architecture in terms of its frameworks and their relationships. Section 4.0 discusses the Scenario and Solver subsystems. The Scenario Framework contains objects which can be combined and configured to solve different manufacturing logistics problems. The Factory and Strategy patterns [Gamma94] are used to create objects that take data from a variety of sources. The Core Framework contains a set of objects for capturing inputs, specifying the working problem, and maintaining previously obtained solutions. Section 5.0 describes the Data Interface (DIF) Framework, which uses the Bridge pattern [Gamma94] to implement a reusable data interface to a variety of manufacturing information systems, such as MAPICS XA [Mapics93] and SAP R3 [Sap94]. Section 6.0 describes the interaction between the User Interface Framework and the Scenario Framework. Section 7.0 describes the construction of two PRM applications: TIA (The Implosion Application) and CMRP (Capacitated MRP).

2.0 Domain Background and Framework Design Objectives

Manufacturing firms compete on the basis of cost, quality, and delivery. They are under constant pressure to deliver better quality products more quickly at lower cost. Manufacturing companies use a variety of software tools to improve the quality and speed with which decisions are made. In this section we briefly discuss MRP/MRP II environments and the limitations of some current practices. We describe a situation within IBM which led us to develop

Objectifying Manufacturing Logistics Decision Support

**Walter C. Dietrich, Jr., Goodwin R. Chin*, 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 Corp.
T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

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

Abstract

When planners solve difficult manufacturing logistics problems, they frequently employ decision support software. The data this software uses may come from many different sources, depending on both the environment and the decisions being made. The techniques for synthesizing the input information and communicating the solution depend on the problems being solved and the users' business processes. This leads to a need for many diverse and unique applications. This paper describes a framework that provides reusable design and implementation components for decision support applications.

1.0 Introduction

Developing manufacturing logistics decision support software has been a labor-intensive and costly endeavor. Except for the underlying solver software (such as linear-programming libraries and constraint-based solvers) many applications have been developed as one-off solutions.

This paper describes the Production Resource Manager (PRM) Framework, which encapsulates the systemic similarities among manufacturers, and allows software developers to readily extend framework components into applications that address specific manufacturing requirements. The PRM Framework contains frameworks which provide 1) rapid graphical user interface (GUI) development, 2) access to a variety of manufacturing information

RC 20637 (11/21/96)
Computer Science/Mathematics

IBM Research Report

OBJECTIFYING MANUFACTURING LOGISTICS 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

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

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