Integration of Rules and Optimization in Plant PowerOps

Thomas Bousonville, Filippo Focacci, Claude Le Pape, Wim Nuijten, Frederic Paulin, Jean-Francois Puget, Anna Robert, and Alireza Sadeghin

ILOG S.A, 9 rue de Verdun, 94253 Gentilly, France
{tbousonville, ffocacci, clepape, wnuijten, fpaulin, jfpuget, anrobert, asadeghin}@ilog.fr

Abstract. Plant PowerOps (PPO) [9] is a new ILOG product, based on business rules and optimization technology, dedicated to production planning and detailed scheduling for manufacturing. This paper describes how PPO integrates a rule based system with the optimization engines and the graphical user interface. The integration proposed is motivated by the need to allow business users to manage unexpected changes in their environment. It provides a flexible interface for configuring, maintaining and tuning the system and for managing optimization scenarios. The proposed approach is discussed via several use cases we encountered in practice in supply chain management. Nevertheless, we believe that most of the ideas described in this paper apply in almost any area of optimization application.

1 Introduction

Most manufacturing companies are organized today around integrated programs called Enterprise Resource Planning (ERP) systems. ERP systems provide the information backbone needed to manage the day-to-day execution handling the many transactions that document the activity of a company. Since the beginning of the new century, Advanced Planning and Scheduling (APS) systems have been increasingly adopted to plan the production taking into account capacity and material flow constraints in order to meet customer demand. APS systems embed algorithms for planning and scheduling spanning from the application of very simple priority rules to complex optimization algorithms depending on the needs of each customer. Although rule-based scheduling and simulation-based scheduling are still widely used, today the best APS systems offer scheduling algorithms based on Meta-heuristics, Constraint Programming and Mathematical Programming.

The highly competitive marketplace on the one hand pushes to improve the production efficiency; on the other hand it pushes to increase the flexibility necessary to adapt to the continuous variations of customer demand. Today manufacturing companies need to produce a higher variety of products and customized products. The increasing needs for flexibility are pushing today's APS systems to their limits. Many companies are struggling with the limitation of the first generation of APS systems and are looking for new solutions.

A company that needs to implement advanced supply chain optimization tools has two possible choices: either it will implement an APS package or it will build it using optimization components and technology via often long and costly custom development. The drawback of buying an existing APS package is that it provides a generic optimization model which will not take into consideration all production constraints and policies characteristic of the company. Often the company is forced to fit into the predefined model. The bottom line can be a very high total cost of ownership combined with unhappy end users who, in some cases, replace the system with their previously developed Excel spreadsheet. The alternative of developing a custom solution is only viable for few companies (often with large OR departments). And even in this case the usability of custom development is not guaranteed. In both cases, changing the supply chain optimization system to follow the rapidly evolving business conditions is an issue.

The challenge for APS packages vendors is therefore to provide enough generality to avoid developing an optimization engine for each and every customer and to build a flexible and configurable enough system to meet the real needs of the customer. Ideally, such package should be configurable by its business users. These are the people that actually solve a business problem, such as producing the production plan for a plant. These business users usually do not possess the IT skills that are needed for adapting an APS package to the peculiarities of their plant.

There are many ways in which an IT package could be made more flexible. One could add a scripting language to it for instance. Unfortunately, even scripting languages are deemed to be too complex to be learned and used by business users. Another possibility could be to use tools that business users use, such as a spreadsheet. However, a spreadsheet interface is not powerful enough to express complex use cases such as business policies. A third approach to flexibility is emerging nowadays. It is called business rules. This approach let business users make statements about their business in a friendly way. These rules are then used to preprocess (or postprocess) the input (or output) of an optimization application. This use of rules is quite different than the so called rule inference systems that were used in expert systems in the 80's. Indeed, rules aren't used here to solve a problem. Rather, they are used to state what the problem is really about. This difference is at the root of the current success of business rules in the market place.

We are convinced that advances and the increased popularity of Business Rules create the opportunity to provide the flexibility the lack of which was limiting the applicability of APS systems. In this paper we describe how Plant PowerOps [9] takes advantage of this technology and we claim that the proposed rules interface can be generalized to the vast majority of optimization applications. Indeed, although this paper does not propose any advance in either rule based systems or operations research, it presents a new, extremely pragmatic, way of applying and integrating rule based systems to optimization models and algorithms.

The structure of the paper is as follows: in section 2 we present an overview of Plant PowerOps briefly describing the types of problems solved by PPO and its architecture. Section 3 is devoted to present different use cases where the integration of rules and optimization is demonstrated to be a powerful combination to overcome the limits of today's optimization software. Section 4 explains the reasons behind some of the design decisions taken in the development of the proposed integration, discusses open questions and future work. Section 5 presents some related approaches. Section 6 concludes the paper.

2 Plant PowerOps Overview

Plant PowerOps (PPO) is a new *Advanced Planning and Scheduling (APS)* system dedicated to production planning and detailed scheduling for manufacturing. It enables users to plan the production taking into account capacity and material flow constraints to meet customer demand.

Although Plant PowerOps provides production planning, lot sizing, and detailed scheduling engines, due to space limitations, we will concentrate the examples to the detailed scheduling features of PPO. The scheduling engine is used to schedule production activities (such as chemical reactions, mixing, forming, assembling or separating), and setup activities (such as cleaning or preparing) on different machines or production lines in order to efficiently produce quality finished products in a timely manner, while satisfying customer demand for the finished product.

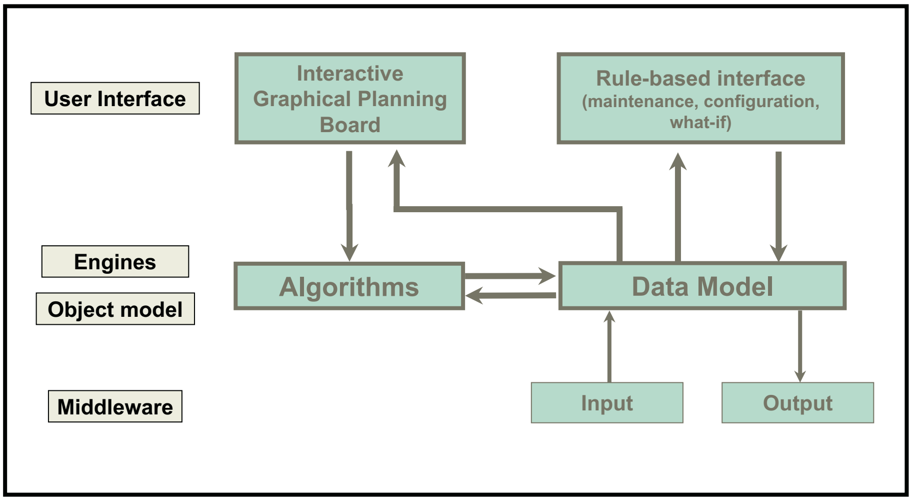
User Interface
Interactive Graphical Planning Board
Rule-based interface (maintenance, configuration, what-if)
Engines
Object model
Algorithms
Data Model
Middleware
Input
Output

Fig. 1. Architecture

As shown in figure 1, Plant PowerOps provides

– a pre-defined data model to capture intricacies of the manufacturing operations. The pre-defined data model represents, for example, production recipes, production orders, calendars (e.g. breaks, shifts, productivity profiles), resources, customer demands.
– effective optimization models and algorithms based on Mathematical Programming, Constraint Programming and Local Search. These algorithms automatically generate feasible, cost-effective detailed schedules minimizing a combination of objective functions such as total tardiness, total earliness, total setup cost, makespan, processing cost, etc.
– a graphical planning board to visualize, analyze, manually adjust and update production schedules.
– a rule-based customization interface to configure and maintain over time the graphical user interface and the parameters of the model and of the algorithm. The rule-based interface provides also the ability to define optimization scenarios and to modify problem data and solutions.
– an integration framework to connect PPO to a database, to an existing ERP (Enterprise Resource Planning), or to an existing MES (Manufacturing Execution System).

2.1 Rule-Based Interface

The integration between the business rules and the optimization model and between the business rules and the graphical user interface is loose. Rules apply either in a pre-processing step (i.e. before the execution of the engine) or in a post-processing step (i.e. after the execution of the engine).

The rule interface of PPO is based on *ILOG JRules* [8] which parses and interprets *production rules* and executes them in forward-chaining using the Rete algorithm [5].

The typical syntax of the production rules interpreted by PPO is the following:

```
when
    conditions
then
    actions
```

Conditions (or left hand side of the rule) are methods returning booleans on objects of the PPO data model. For example, a condition looking for all activities which are due on Jan 1st is translated as *evaluate(theActivity.getDueDate() equals "Jan 1, 2005"). Actions* (or right hand side of the rule) can be (i) any method (returning void) typically modifying the state of the objects, (ii) insertion in the working memory, (iii) retraction from working memory. We use actions to produce side effects on the model or on the solution. On top of the rule language, JRules provides a syntax in natural language like flavor that is used in all the examples shown in this paper.

The rule engine and the optimization engine are fully independent and communicate only through modification of the model and of the solutions. The Plant PowerOps user selects the rules she/he wants to apply in a given scenario before or after the optimization per se. Moreover, the optimization model is exposed via a high level *closed* interface. The interface is *closed* in the sense that we do not provide direct access to decision variables so that only predefined constraints are possible.

The left hand side of a rule checks conditions on the model and its right hand side produces side effects on the model if the conditions are met. Rules apply either in a pre-processing step (i.e. before the execution of the engine) or in a post-processing step (i.e. after the execution of the engine). Typically, a pre-processing rule can be seen as a way to transform a model (coming from the legacy system) into a new model upon which optimization is performed. The optimization engine optimizes the transformed model and has no knowledge of the rules that applied to generate it. Post-processing rules check the state of the model and the solutions after optimization has occurred and possibly modify the solutions. The advantage of the loose integration proposed relies on its simplicity and modularity.

Note that the way rules are used in PPO is very different from the way rules are used in expert systems. In expert systems rules are used to solve the problem at hand. This usually requires complex sequences of rules firing, and maintenance was a real concern. In PPO the number of rules we expect to be active is very limited; the rules are often independent from each other and rarely chained together. This simplifies the maintenance issue while still allowing business users to understand and manage them.

3 Use Cases

3.1 The Chocolate Factory

In order to demonstrate the interest of integrating business rules and optimization algorithms we will describe some use cases that could be faced by supply chain managers and production planners of an imaginary chocolate factory.

This imaginary factory produces chocolate confections. Many production steps and machines are required to complete the manufacturing process. The manufacturing process is driven by customer demands and production orders, these processes being driven by recipes and the materials they produce. Costly setup times are required, and multiple process modes (e.g. an activity may be performed in alterative machines) are possible and may be associated with different process costs. Also, activities have precedence constraints.

The factory produces chocolate eggs, rabbits and squirrels. These can be made of either dark chocolate or milk chocolate. Each shape of product –egg (E), rabbit (R), or squirrel (S)– made of either dark (D) or milk (M) chocolate, can be filled with coconut cream (C), hazelnut cream (H), or filled with nothing (N). The possible combinations are identified using three-letter acronym such as *DCE* for dark coconut egg, *MHR* for milk hazelnut rabbit, *DNS* for dark no filling squirrel, etc.

The chocolate factory is composed of two production lines located in two different cities in Switzerland. Each production line has the following production equipment: a cocoa grinding machine, a chocolate mixing machine, a nut grinding machine, a cream mixing machine, and a molding machine.

Recipes for the 12 products with cream filling consist of 5 activities (*Cocoa grinding*, *Chocolate mixing*, *Nut grinding*, *Cream mixing* and*Molding*). Recipes for the 6 products with no cream filling consist of only 3 activities (*Cocoa grinding*, *Chocolate mixing*, *Molding*). For each recipe there are precedence constraints such as *Cocoa grinding must precede chocolate mixing.*

There are 5 customer demands:

– One for 3 batches of MNE due Jan. 2, 2005 at 6:00 am
– One for 2 batches of MHR due Jan. 3, 2005 at 6:00 am
– One for 3 batches of DCE due Jan. 4, 2005 at 6:00 am
– One for 4 batches of MHR due Jan. 5, 2005 at 6:00 am
– One for 7 batches of DNE due Jan. 6, 2005 at 6:00 am

These demands result in a total of 19 production orders, one for each batch.

In December, 2004, you are trying to solve a scheduling problem to commence on January 1, 2005, 06:00 am. This example also includes setup times and setup costs, and processing costs. For example, the chocolate mixer requires cleaning when changing from mixing milk chocolate to mixing dark chocolate. The objective is that manufacturing activities should finish as close as possible to their ideal due dates. There are costs associated with late completion. There are also costs associated with the choice of alternative resources that activities are performed in, and with setup times required by those activities. These should also be kept to a minimum.

Typically, the production planner runs Plant PowerOps once a day in order to meet customer expectations and keep internal costs to a minimum. In addition, the production planner runs Plant PowerOps whenever unexpected events (such as a machine breakdown) occur, in order to repair the schedule adapting it to the new situation of the factory. The supply chain manager uses Plant PowerOps to run several simulation scenarios in order to adapt the supply chain business policies to modifications of the market.

We first describe the activities of the supply chain manager; we will successively move to the description of the activities of the production planner.

3.2 What-If Analysis

One of the most important tasks of the supply chain manager is to design and control the production system. A way to achieve supply chain efficiency is to simulate and study the impact of external events and production policies. A very first step is to modify the production data. This can clearly be done by hand on a local copy of the legacy system. A much more effective way to perform massive and complex data modifications is to describe the modification using a rule-based language. The examples of this section demonstrate how business rules can be used to simulate events; in the examples of section 3.3 we demonstrate how business rules can be used to define policies to be applied upon these events. In particular, example 3 represents a business policy applicable to example 1 and example 4 represents a business policy applicable to example 2.

Example 1. Resource Shutdown. Although resource breakdowns are quite infrequent, they may have very important consequences to the efficiency of the production system. Also the supply chain manager may consider the possibility to close part of the factory on a specific day (e.g. Jan 1st 2005). The simulation of a resource shutdown is a necessary first step to try several action plans (business policies) that will be executed upon such an event (see example 3).

```
Declarations
    for the resource, instance of resource
      where the name of the resource is Cream mixer 1
           or the name of the resource is Chocolate mixer 1,
    for the bucket, instance of bucket
      where the bucket is between Jan 1, 2005 6:00am and Jan 2, 2005 6:00am
Then
    the resource is unavailable in the bucket
```

Note that the left hand side of this rule is expressed by the Declarations section which designates the matching objects. Note also that in this first example the resource shutdown can easily be coded using a graphical user interface instead of the rules interface. Section 4.3 discusses the relation between GUI and rules interface.

This rule is automatically translated into a production rule that has a side effect on the model:

```
    when {
        the_resource:IloMSResource((getName() equals "Cream mixer 1")
            or (getName() equals "Chocolate mixer 1"));
      the_bucket:IloMSBucket(isBetween("Jan 1, 2005 6:00am",
            "Jan 2, 2005 6:00am"));
    } then
      modify the_resource.setCapacity(0,the_bucket);
```

Example 2. Important Sales Agreement with a Customer. The conditions of a big sales agreement are going to be negotiated with an important customer of the company, which could result in doubling the business made with this company. During the Sales and Operations Planning meeting, the sales representative asks the supply chain manager to study the impact on the production that would be caused by the deal (e.g. on production capacity).

```
Declarations
    for the customer order, instance of demand
        where this demand is a customer order
If
    the name of the customer for the customer order is "Hane"
Then
    set the quantity of material requested by the customer order to
        the quantity of material requested by the customer order × 2
```

3.3 Business Policies

As mentioned before, simulation in terms of massive and complex data modification is only the first step for an effective management of the production system. Once we are able to simulate events, we want to define those business policies that enable us to best deal with the events.

Example 3. Reduce Safety Stock During a Resource Shutdown. Safety stocks are necessary to face unexpected events and stochastic data. A resource breakdown is one such event and it justifies the usage of safety stock. Combining the optimization algorithm and the business rules capability of Plant PowerOps the supply chain manager is able to find the following business policy:

```
Declarations
    for the down bucket, instance of bucket,
    for any resource, instance of resource,
    for any material, instance of material,
    for the impacted bucket, instance of bucket
    where the start time of this bucket is greater
      than the start time of the down bucket
      and the start time of this bucket is less than
      the start time of the down bucket + 15
If
    the capacity of any resource in the down bucket is 0
Then
    set the safety stock of any material in the down bucket to 0
    and set the safety stock of any material in the impacted bucket to 0
```

This rule accounts for the fact that not only during the shutdown time, but also during a given time that follows the resource unavailability it is appropriate to use the safety stock in order to fulfill the demand. The rule is telling the optimizer to accept a lower stock by reducing the desired safety stock level to 0. Note that by using the rules in example 1 and 3 the supply chain manager is able to define an appropriate policy to apply in case of resource breakdowns or decided shutdown.

Example 4. Gold Customer Production Policy. In order to obtain a pivotal selling agreement (see example 2), the CEO of the company has promised to never deliver late large orders coming from the gold customer. The supply chain manager has implemented the following business policy into the system.

```
Declarations
    for the demand , instance of demand
        where the demand is a customer order
If
    the category of the customer for the demand is gold
Then
    set the tardiness variable cost for the due date of the demand to "high"
```

3.4 Model Preprocessing

Adding an APS on top of an ERP means stepping from pure transactional data processing to the more complex optimization tasks. It often turns out that the existing data in the ERP data base is not sufficient (i) to express all constraints that hold for the production problem, (ii) to incorporate implicit preferences of the planner, (iii) to balance between conflicting objectives in the evaluation of a solution.

While adding appropriate fields for static data to the legacy system is not a big issue, there are numerous cases where this data has to be calculated dynamically (optimization weights, load dependent preferences, etc.). Maintenance of these data and procedures can become a nightmare.

Preprocessing rules help to express explicitly the necessary transformation logic and avoid out-of-date and inconsistent data by creating it dynamically. They also provide an easy way to build a set of preferences the user wants to apply in a given context.

Example 5. Products for the Same Customer Demand are to be Produced on the Same Production Line. For some products a high degree in regularity is important. Let's assume a nearly identical molding quality can only be guaranteed when the chocolate is processed on the same line. To dispatch this constraint we can use the following rule:

```
Declarations
    for the demand, instance of demand,
    for order A, instance of production order
      where the demand is satisfied by order A,
    for order B, instance of production order
      where the demand is satisfied by order B,
    for activity 1, instance of the activities generated from order A,
    for activity 2, instance of the activities generated from order B
If
    the name of activity 1 contains Molding
    and the name of activity 2 contains Molding
Then
    insert in the working memory a new activity compatibility constraint
      so that activity 1 and activity 2 are processed on the same line
```

Note that activity compatibility constraints are part of the object model of Plant PowerOps [9]. This constraint forces two given activities to be executed in resources belonging to the same production line.

3.5 Tune the Engine

An effective plan is always a trade-off between conflicting objectives. For example, in order to minimize the setup and production costs we should produce long campaigns of similar products. Such a production policy will probably lead to poor customer satisfaction because there is a continuous demand for a mix of different products. After having classified its possible customers in three categories (*normal*, *silver*, *gold*), the supply chain manager decided to adapt the objectives of the optimization to the configuration of the customer demands to be satisfied. In case of large amounts of demands from gold customers, customer satisfaction should be privileged. Otherwise production efficiency should be more important.

Example 6. Emphasize Customer Satisfaction.

```
If
    the percentage of gold customers is less than 20
Then
    set the total setup cost weight to ”high”
    and set the total tardiness weight to ”low”
Else
    set the total setup cost weight to ”low”
    and set the total tardiness weight to ”high”
```

The last three following scenarios concern the use of the business rules interface during the activity of the production planner. The production planner uses Plant PowerOps for generating the day to day schedule of the factory and is not allowed to change the business policies defined by the supply chain manager. He/she is nevertheless able to use the rule based interface to configure the system for his/her daily activities and to run validation tests.

3.6 Data Validation

Example 7. Minimal Order Quantity. For technical reasons (or by mistake) the sales department may enter into the system customer orders with low quantity of finished products. To prevent these orders from being considered in the planning, the following rule enables the production planner to make sure that only orders with more than two batch units are scheduled.

```
Declarations
    for the demand, instance of demand
        where the demand is a customer order
If
    the quantity of material requested by it the demand is less than 3
Then
    display the name of the demand
    and display ”requests less than 3 product units”
```

3.7 Solution Checking

In addition to built-in solution checking, PPO allows defining factory specific checking rules applied to solutions. It does not matter if the solution has been generated by the optimizer or by hand. This technique is well suited to check soft constraints or desired properties that are not directly expressed in the constraint model.

Example 8. Temporal Dispersion of Related Activities. The following rule keeps the planner informed when two activities belonging to the same production order have been scheduled far away from each other.

```
Declarations
    for order A, instance of production order,
    for activity 1, instance of the activities generated from order A,
    for activity 2, instance of the activities generated from order A,
    for the solution is the best scheduling solution
If
    the start time of activity 2 in the solution is greater than
        the end time of activity 1 in the solution + 15
Then
    add in the checker of the solution a violation "Dispersed activities"
```

3.8 Graphical Rules

Graphical actions include coloring, filtering and selection. While most of them are predefined (filtering types of resources, color late activities), others have more complex parameters.

Example 9. Select Late Activities That Belong to a Gold Customer. As we have seen above, some orders may have a higher priority than others. Therefore we would like to refine the information presented by selecting only the late orders that are produced for a gold customer. Using the rule interface this can be expressed as follows:

```
Declarations
    for the customer order, instance of demand
        where this demand is a customer order,
    for the order, instance of production order
        where the customer order is satisfied by the order,
    for activity 1, instance of the activities generated from the order
If
    the category of the customer for the customer order is gold
    and the tardiness cost of activity 1 in the solution is greater than 0
Then
    add activity 1 to selection
Else
    remove activity 1 from selection
```

For *coloring* different types of color schemas make sense: customer type, order value, material properties, etc. Using a rule based configuration interface the user can establish a series of commonly used coloring schemas without coding. The gain against a call for application extension can be measured in money, time and autonomy [1].

All the scenarios presented demonstrate the flexibility of a rule-based interface on top of optimization algorithms. Note that these scenarios could not have been done on the any of the most popular APS in the market without a major development effort. In fact either they do not provide any form of scripting language (e.g. Oracle/APS), or the scripting language does not allow modifications of the optimization model (e.g. the ABAP language of SAP/APO). In some cases it is possible to write special purpose optimization algorithms replacing the ones available in the APS (for example, this is true for both Oracle and SAP). However this implies a large project, including writing transformation from and to business model and a brand new optimizer. It would be overkill to achieve one of the scenarios described by such custom development.

4 Open Questions and Future Work

4.1 Loose Integration or Tight Integration

Although conceptually interesting, we are convinced that, in general, a tighter integration where the rules and the optimization engines directly communicate, would be much more complex without bringing a sensible added value. A tighter integration would end up being yet another high level optimization language (based on business rules). Such an imaginary *rule-based optimization language* would be far from being practical as supply chain optimization tool dedicated to people with little optimization experience. Moreover, the interaction of optimization and rules engines would generate difficult robustness issues. On the contrary, in the proposed approach, the robustness of embedded heuristics is enforced by the closed model. The end user may enrich the model using predefined constraints and influence the search procedure, but not interact with it. Therefore, once we are able to deal with infeasible input data, we do not have to deal with issues such as rules that could make the optimization problem impossible to handle as this translates into infeasible input data. Somewhere in between the loose integration proposed in this paper and a tight integration is applied in [4] for optimization systems used in the airline and railway industries and is described in section 5.

[1] Note that the manufacturing object model of Plant PowerOps does not provide the concept of a customer category (*normal*, *silver*, *gold*). Plant PowerOps enables users to dynamically attach properties to objects. These properties can be used in the left hand side (the *if* statement) of business rules thus providing a powerful mechanism to extend the object model and to write constraints (rules) based on these extensions as shown in the examples 4 and 9.

4.2 Use Rules to Guide the Search Heuristics

Although we are convinced that a loose integration of rules and optimization is better than a tight cooperative integration, nevertheless we are aware that business rules may play an important role in a more sophisticated method to guide the engine towards desired solutions. The design of methods to guide the search based on business rules is subject of future work. We believe that the following types of interactions could be highly interesting. Interaction of rules and optimization in constructive search methods; definition of local moves via business rules; use business rules to describe how to repair an infeasible schedule, and finally definition of soft constraints (preferences) via business rules. The challenge of the design of rule-based methods to guide the search will be to keep the clear separation between the rule based interface and the optimization engines.

4.3 Rules and GUI

Nowadays business rules systems such as *ILOG JRules* [8] provide a user friendly interface to write rules in natural language (see all provided examples) or technical language (see example 1). Rules can be stored in rule repositories and saved. Moreover, *parametric rules* or *rule templates* can be defined to enable users to generate new rules by modifying (specializing) a given rule template. Despite all that, writing a rule is always a complex task compared to a sequence of clicks in a graphical user interface. Consider example 1 of section 3.2 where a rule defines a machine breakdown or shutdown. This is a typical case where a small graphical item could provide the very same functionality with a much simpler user interaction. In our experience, it is not always easy to decide which functionality should be provided as GUI items and which should be provided via a rules interface. Our current approach is to provide a set of pre-defined rules first, which may become part of the graphical user interface later upon request.

5 Related Work

The integration of rule-based systems and optimization has been widely investigated in the literature. For example, one of the first constraint-based scheduling system, SOJA [10], used rules both to select the activities to schedule over the next day and to heuristically guide the constraint-based search. A more systematic approach proposing integration of constraints and rules can be found in the programming language LAURE [3] [1]. Caseau and Koppstein propose a multi-paradigm object-oriented language integrating rule-based and constraint-based technology. LAURE supports forward chaining production rules and backward chaining. In LAURE rule-based programming provides deductive capabilities that is merged with constraint satisfaction for improving the efficiency of constraint satisfaction. The integration of rules and optimization in LAURE is tight, and the rule technology is part of the optimization language used to solve the problems. The programming language LAURE evolved in a new programming language called CLAIRE [2] which packages the features proved useful in LAURE in a much simpler language. The backward chaining functionality of LAURE was removed, and the forward chaining functionality was basically used to build propagation algorithms.

A different integration of rule technology and problem solving can be found in the vast literature on *Constraint Handling Rules* (see e.g. [6]). Constraint Handling Rules is a high-level rule-based language for writing constraint solvers and reasoning systems. Again, the spirit of the integration of rules and optimization is very different from the loose integration proposed in this paper.

A rule-based front end to optimization is available in the crew pairing optimization system of *Carmen Systems* ([4], [7]) where the rule language *Rave* is used to define feasible pairings. In the airline and railway industries, legal pairings must satisfy a large number of governmental and collective agreements which vary from an airline to another. Such rules are not hardwired, but rather specified by the user using the specific rule language *Rave.* The interaction between the optimization engine and the rule engine is tighter than the one proposed in this paper as the rule engine is called to validate possible pairings during column generation. It is still a loose integration in the sense that the rule engine behaves as a black box for the optimization engine and provides simply a yes/no answer on the feasibility of possible pairings. The advantages of the integration of rules and optimization of *Carmen System* are that the rules can be easily changed and maintained by users and it is easy to perform what-it analysis.

6 Conclusions

We have proposed a new, pragmatic, approach for the integration of business rules and optimization engines. The proposed integration provides the flexibility, adaptability and extensibility that was missing in today's supply chain optimization systems. Besides the presentation of the integration framework, one goal was to present a categorization of pertinent rules for optimization applications. This classification was done based on the rule purpose in the application context: what-if analysis, business policies, model preprocessing, engine tuning, data validation, graphical actions and solution checking. Although the proposed approach is described on supply chain optimization, we believe it can be applied to most optimization applications. For example, similar investigation is conducted at ILOG in the area of transportation. We hope that the flexibility provided by the interaction of rules and optimization removes many obstacles in the adoption of Advanced Planning and Scheduling systems.

References

1. Y. Caseau and P. Koppstein. A Rule-based approach to a Time-Constrainted Traveling Salesman Problem. In *Proceedings of Symposium of Artificial Intelligence and Mathematics*, 1992.
2. Y. Caseau and F. Laburthe. CLAIRE: Combining objects and rules for problem solving. In T. Ida M.T. Chakravarty, Y. Guo, editor, *Proceedings of the JICSLP'96 workshop on multi-paradigm logic programming*, 1996.

3. Yves Caseau and Peter Koppstein. A cooperative-architecture expert system for solving large time/travel assignment problems. In *Database and Expert Systems Applications*, pages 197–202, 1992.
4. N. Kohl E. Andersson, E. Housos and D. Wedelin. *Crew pairing optimization*, pages 228–258. Kluwer Academic Publishers, 1990. G. Yu, editor.
5. C.L. Forgy. Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, pages 17–37, 1982.
6. Thom Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming, Special Issue on Constraint Logic Programming*, 37(1-3):95–138, October 1998.
7. Curt A. Hjorring and Jesper Hansen. Column generation with a rule modelling language for airline crew pairing. In *Proceedings of the 34th Annual Conference of the Operational Research Society of New Zealand*, 1999.
8. ILOG. *ILOG JRules 5.0 User's Manual and Reference Manual.*
9. ILOG. *ILOG Plant PowerOps 1.0 User's Manual and Reference Manual.*
10. C. Le Pape. Soja: A daily workshop scheduling system. soja's system and inference engine. In *Proceedings of the Fifth Technical Conference of the British Computer Society Specialist Group on Expert Systems, Warwick, United Kingdom*, 1985.