SUPPORT SYSTEM FOR PROCESS FLOW SCHEDULING

Juan Lerch, Enrique Salomone and Omar Chiotti

GIDSATD – UTN FRSF, Lavaisse 610, 3000 Santa Fe - Argentina
INGAR – CONICET, Avellaneda 3657, 3000 Santa Fe –Argentina
salomone@ceride.gov.ar, chiotti@ceride.com.ar

Abstract

Process flow scheduling is a concept that refers to the scheduling of flow shop process plants, whose scheduling calculations are guided by the process structure. In a wide variety of high-volume process industries, the process flow scheduling concept implies an integrated structure for planning and scheduling.

This integrated vision of the planning function and the very particular characteristics of the process industry production environment challenge the application of the most traditional approaches to support planning/scheduling activities. We are working with the aim of developing a conceptual system foundation to support process flow scheduling.

We started from a grass roots study of the current process flow planning/scheduling practices and we applied an object oriented analysis and design methodology to design a system to support process flow scheduling. Essentially, the system supports a process flow scheduling model that can be used to instantiate production resources of a particular industry, the plant structure and the products with their respective production processes. In this way, the particular scheduling problem of a production order set will be instantiated.

In this work we describe the process flow scheduling problem, a process flow scheduling support system architecture, a general ontology of the process flow scheduling, an object oriented design of the system and some implementation details.

Key Words: Scheduling, system, framework, process flow

1. Introduction

Process flow scheduling (PFS) is a concept that refers to the scheduling of flow shop process plants, whose scheduling calculations are guided by the process structure (Taylor and Bolander, 1994). In a wide variety of high-volume process industries, the process flow scheduling concept implies an integrated structure for planning and scheduling.

This integrated vision of the planning function and the very particular characteristics of the process industry production environment challenge the application of the most traditional approaches to support planning/scheduling activities. Moreover, most of the currently available systems have been initially developed for short-term scheduling of discrete manufacturing industries, and then extended to attempt a universal application to other manufacturing environments (Allweyer et al., 1996).

The strategy of accommodating new requirements into the same conceptual foundation has produced limited progress in trying to address needs of process industries. We are working in this direction with the aim of developing such a conceptual system foundation to support process flow scheduling.

We started from a grass roots study of the current planning/scheduling practices of the process industry and we applied a system analysis methodology to design a system to support PFS. Essentially, the system supports a PFS meta-model that can be used to instantiate production resources of a particular process industry, the plant structure and the products with their respective production processes. In this way, the particular scheduling problem of a production order set will be instantiated.

Therefore, our objective has been to develop a conceptual system foundation to support PFS. In this work we describe the PFS problem, the PFS support system architecture, a general ontology of the PFS, an object oriented design of the system and some implementation details that we consider of interest.

2. The Process Flow Scheduling

Taylor and Bolander (1994) define a strategy to guide the PFS that is based on three principles: the scheduling calculations are guided by the process structure; stages and clusters are scheduled using processor-dominated scheduling or material dominated scheduling approaches; and process trains are scheduled using reverse flow scheduling, forward flow scheduling or mixed-flow scheduling.

2.1 The scheduling calculations are guided by the process structure

In accordance to this principle, a classification procedure and terminology are needed to define process structures. The terminology and classification used in this work are represented in Figure 1. The process structure consists of process units, clusters, stages, and trains. A process unit performs a basic manufacturing step. Process units are combined in clusters at each stage and stages are separated by storages. Separating clusters and stages by storage allows for scheduling them as an independent entity. Stages are then organized in trains. Usually, materials are not transferred from a train to another one, although there are exceptions that should be taken into consideration.

The definitions used in our work are the following:

Storage: It is some kind of warehouse where products can be kept. These products may be either intermediate (products that still require to be processed) or final. Storage may have a finite capacity or it con be assumed infinite, in case that the storage capacity is not taken into account for the Scheduling.

Process Unit: It is any equipment of the plant that is capable of carrying out a product transformation. A process unit have a finite capacity, and it generally requires an storage origin (raw material) and an storage destiny. Process units that belong to *stage N* receive materials from s*tage N-1* and provide materials to *stage N+1*
Clusters: They are sets of process units that share the same scheduling. Once the Cluster scheduling is defined, it is possible to schedule each process unit inside the Cluster. That is, units inside the Cluster may be scheduled as if they were a unique process unit. The smallest Cluster is the one that contains only one process unit.
Connections: A connection is the material link between a process unit and a storage, and its object is to transport products from one to another.

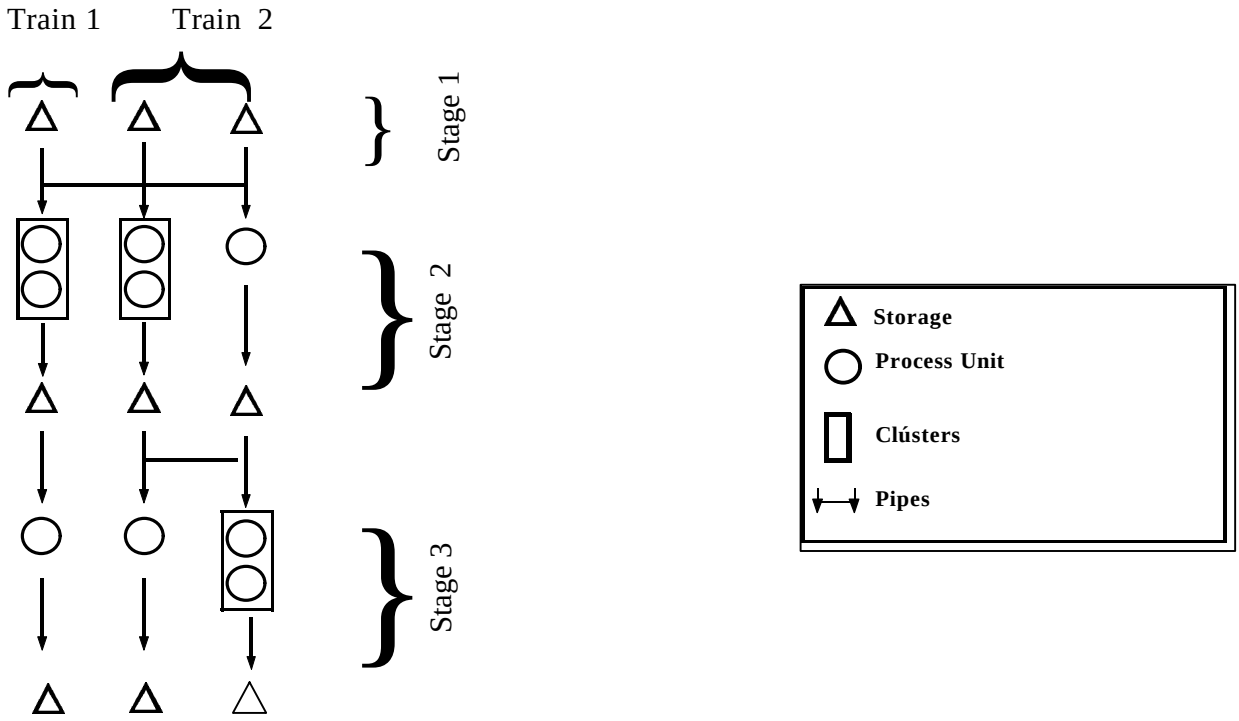

Figure 1: the process structure of an industrial plant

2.2 Stages and clusters are scheduled using process-dominated or material dominated approaches

This principle defines two alternatives to schedule stages and clusters: Stages and clusters can be scheduled using process-dominated scheduling. This occurs when process units are scheduled before material scheduling; Material dominated scheduling. This occurs when process units are scheduled after material scheduling.

Choosing the strategy to be used depends on the specific environment. Process-dominated scheduling is convenient when production capacity is quite expensive, when the cluster is the bottleneck of production, or when setup costs are high. On the other hand, material dominated scheduling is convenient in the case of expensive materials, limited capacity, high setup costs, or when the cluster consists of a set of process units that operate as a Job Shop**.**

2.3 Process trains are scheduled using reverse flow, forward flow or mixed-flow approaches

This principle defines three alternatives to schedule the process trains. Process trains may be scheduled using **reverse flow scheduling, forward flow scheduling**, or **mixed-flow scheduling.** The first one consists of starting from the last stage of the process up to the first ones after obtaining a Scheduling. The second one

consists of starting by the first stage up to the last one following the material flow. And the last one involves mixed strategies since it consists of starting by an intermediate stage, scheduling a bottleneck process unit or cluster or an expensive material and then performing a reverse flow scheduling for the previous stages and a forward flow scheduling for the following stages.

In addition, another aspect that must be taken into account in PFS is that connections between storages and process units are to be involved in the process. In some industries, this is an important characteristic, since sometimes connections are shared by different process units and each of them needs the connection to be exclusive so that it can provide the material produced in its storage. Therefore, connections must be represented and dealt with as another resource to be scheduled.

3. The Support System Architecture

We have defined a system architecture to support the PFS. The aim has been to reuse modules of typical scheduling systems. In this way, the architecture will have to allow us both to focus on the process industries scheduling and to enable a support scheduling application development reusing components.

In Figure 2 we present a conceptual representation of an industrial production process scheduling system. The system receives the customer orders and the information about both actual product stocks and materials availability. Furthermore, the system has information about providers, products and production policies. Based on this information, the production process scheduling system defines the required production orders. Then, the system must obtain a good feasible schedule of the production order set.

The actual production state of the plant must be fed back to the scheduling system. This task is delegated to the supervisory control module, whose function is to monitor the production process with the purpose of detecting deviations between the proposed schedule and the real operation. These deviations are analyzed to decide if a new schedule is required.

To execute the scheduling task, the system has a structure that supports the representation of all components involved in a productive process, i.e.: plant topology, equipment items, actual orders set, products, production processes, resources, and the lot concept, which is an important aspect in the process industry. This representation structure defines the PFS module, which has three interfaces:

A configuration interface. By means of this interface a plant can be instantiated, i.e. its processing units, storages, clusters, stages and trains, products and their processes.

A data interface. By means of this interface, the production orders can be input and information on the current schedule can be obtained.

A control interface. This interface makes it possible to include parameters for the scheduling algorithm.

Figure 3 shows the architecture proposed for the scheduling system. It can be observed that the PFS module, which is in charge of maintaining the problem representation logic, is just a module of the system.

A scheduling system must interact with other systems to obtain external data**,** for example customer orders. The manager module must input data and inform the PFS module about them. This module is also responsible for providing information to the users' services. The users' service is in charge of providing the manager information in the format required by the end user (For example, html, xml, xsl, graphic, etc.), and reflecting the user's commands in the PFS module.

The persistency module is in charge of keeping the PFS information in a persistent storage, since the PFS module has been designed using Objects. The object oriented is the best technique for representing the scheduling problem logic, since it implies quite complex relationships among items.

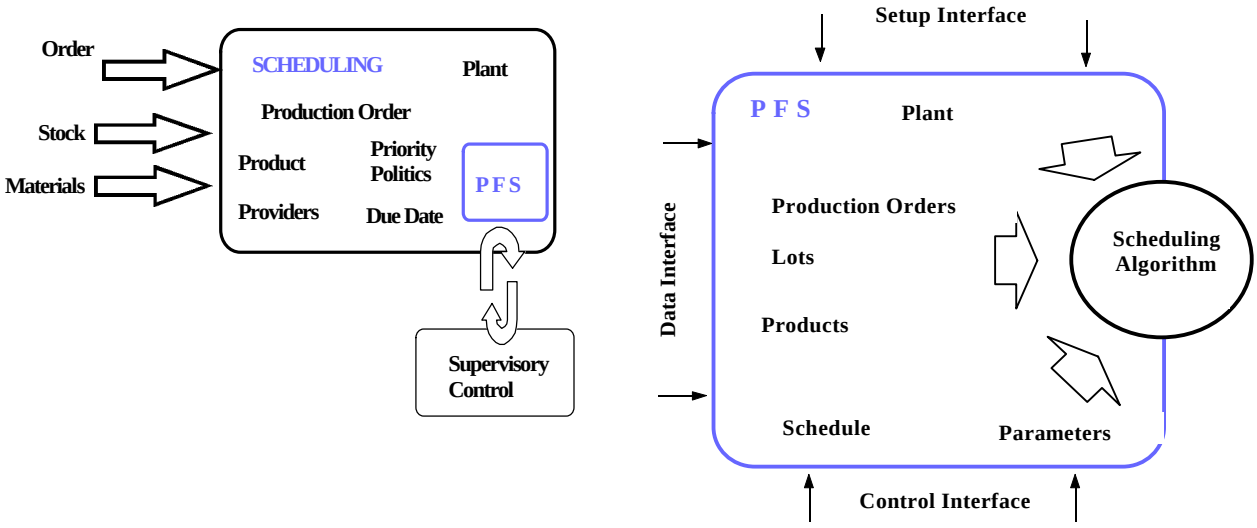

Figure 2: conceptual representation of an industrial production process scheduling system

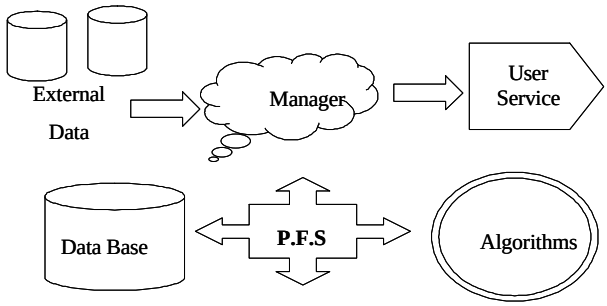

Figure 3: Architecture of the scheduling system

Scheduling resolution algorithms are designed to find the solution to the Scheduling problem. The needed information can be found in the PFS module. Algorithms take this information, find partial or final schedule and then use the PFS module as a solution repository.

This architecture makes it possible to isolate the problem representation from its solution. It also enables the items belonging to a scheduling system to be developed in parallel. Persistence of data is also isolated from the technology to be used. The main advantage of this strategy is that it allows us to concentrate on the aspects related to the process industry, such as representing the problem by identifying the involved items and then designing and test different solution algorithms for each case.

4. Process Flow Scheduling Ontology

Figure 4 shows a representation of the general ontology we have taken as a basis to design the PFS module. The main items in this ontology (Smith, 1997) are:

Activities. They represent an action or a step that is necessary to produce a product. They need *resources* to be executed. *Resources* are entities that allow or support the execution of activities.

On the other hand, we have *products*, whose function is to fulfil a *demand*. *Products*, *demand* and *resources* impose constraints on the *activities*. These constraints set boundary marks on the activities' start,

end and length domains. In this framework, solving the Scheduling consists of finding values for the start, end and length variables of an activity so as to meet the imposed constraints.

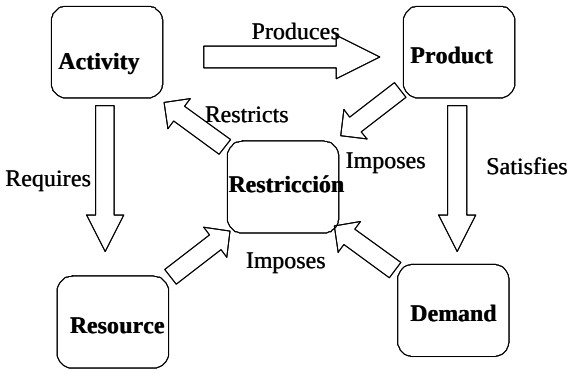

Figure 4: general scheduling ontology

It should be noted that we have expanded the constraint concept so that the “activity requires a resource” relationship can be also modeled by means of constraints. The same happens with the “activity produces a product” and “product satisfies demand” relationships.

The proposed ontology suggests using a model of entities that relate with one another by means of constraints, which makes it possible to expand the model so as to use optimization techniques.

We have also expanded the general ontology to include items that are significant for the scheduling problem. These items are the plant and its structure, the production process, the production order and production lots concepts that affect both demand and the activities required to satisfy it.

5. PFS Support System Design

We have made a logic separation in the system design process. Since we have used an object-oriented methodology (Booch, 1999), the used mechanism has been **packages**. Figure 5 shows the diagram of the packages structure that has guided our design.

The five packages we have used are **Constraints, Product**, **Plant**, **Scheduling** and **Iterators.**

Plant package includes the items related to the production plant, i.e., storage, cluster, stages, trains, connections, process units, and the plant itself.

Product package contains everything related to products, i.e., production process, production orders, production lots and the product itself.

Scheduling package contains the activities and the scheduler, who is in charge of supplying an interface for all the classes that participate of the solution-searching process.

Constraints package involves constraints among activities, activities and resources, and constraints among processes, since we have expanded the constraint concept so as to be also able to model processes by means of constraints.

Last, we have included iterators as a method to access to objects inside the PFS. Iterators are found in the **Iterators** package and have been implemented through the iterator pattern (Gamma, 1995).

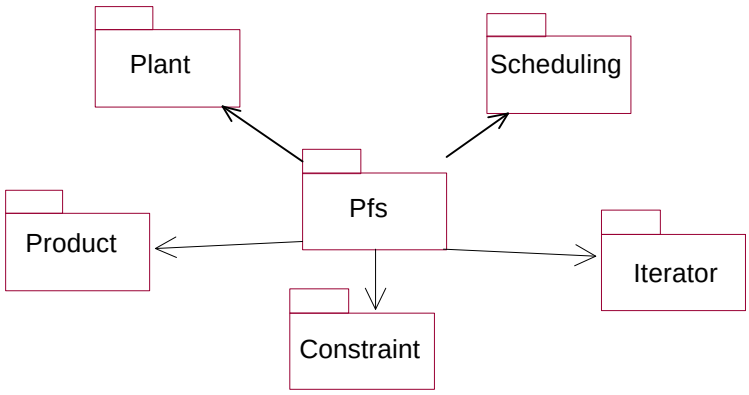

Figure 5: diagram of the packages structure

To sum up, we may say that this way of approaching design, starting from a general architecture, taking the scheduling ontology as a basis, and again separating the design into packages, has enabled us to concentrate our efforts on designing a particular aspect of the problem. In the following subsections, we will present a detailed design and the main design decisions made of each component of the PFS module.

5.1 Production Plant

Figure 6 shows the classes structure corresponding to the main items of a processing plant. It presents the **Plant** class, which is made up of a set of trains in **Train** class. These trains are constituted by a set of stages in **Stage** class, each of which has n equipment items in **EquipmentItem** class.

Plant: It is the main container. It contains a list of process types in **ProcessType**, trains in **Train** and equipment items in **EquipmentItem** and the methods to add these items.

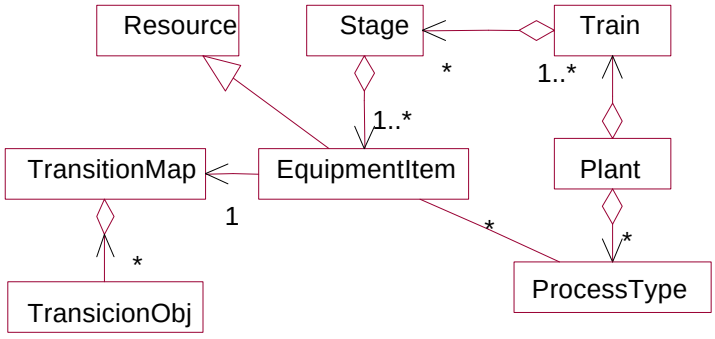

Figure 6: class structure of a process plant

EquipmentItem: It is a piece of equipment of the plant, for example mixers, tanks, etc. Their main characteristics are: (1) They have a *production rate* that states the processing speed of the equipment, and (2) a *holding capacity* that refers to the amount of material that a piece of equipment may contain. The latter is a typical characteristic of batch equipment and tanks or warehouses.

TransitionMap: Transitions occur when a piece of equipment finishes an activity and begins a different one. These transitions may be of different types (cleaning, dead times, setup). The function of the **TransitionMap** class together with the **TransitionObj** class is to behave as a matrix, whose files and columns are the type of activity that was being carried out and the type of activity to be performed respectively. It brings back the cost or the associated time.

5.2 Production Resources

A resource in **Resource** class has a main characteristic or responsibility. It consists of keeping a record of its state in each time interval. To achieve this goal, a **TimeTable** class has been defined, whose function is to bring back and store, for a certain time interval, a resource state. The diagram in Figure 7 shows how this operation is performed.

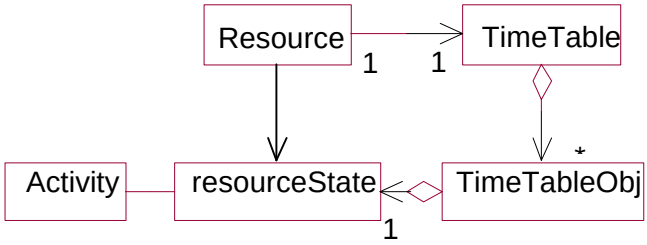

Figure 7: TimeTable class

It can be observed that a resource in **Resource** class has an associated time table in **TimeTable** class, which is an ordered list of time table objects in **TimeTableObj** class. The latter is constituted by a resource state in **ResourceState** class, which represents the state of a resource. The association to the **Activity** class refers to the activity that incorporates the state to the resource.

The diagram in Figure 8 shows the classes hierarchy for resources in **Resource** class. Two types of resources are mainly distinguished: **EquipmentItem** class and **Material** class. At the same time, an equipment item may be: a process unit in **ProcessUnit** class, a storage in **Storage** class or connections in **Pipes** class or in **Filter** class.

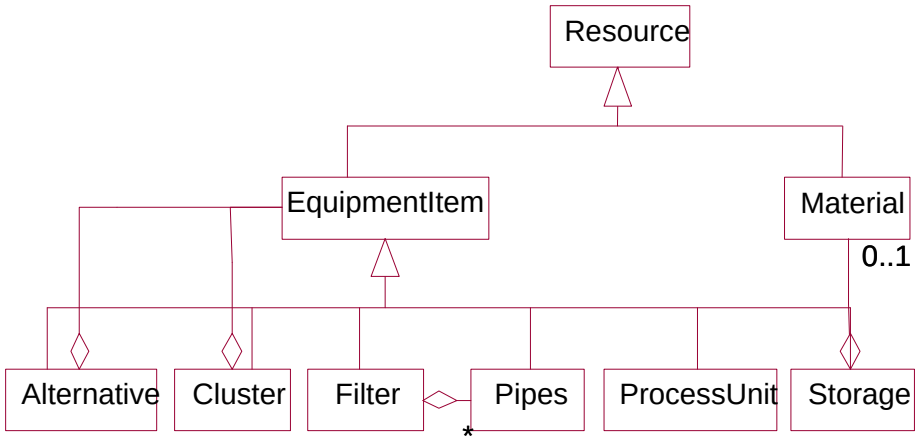

Figure 8: Class hierarchy for resources

The **Material** class makes it possible to record the inventory levels of a material or product. When the **Material** class is used as a resource, it is suggested that the changes of a given material quantity as time goes by is wanted to know. On the contrary, when an equipment item is used, the equipment availability along time is wanted to observe.

A process unit in **ProcessUnit** class is a piece of equipment that is capable of performing some process over a product. Generally, process units take material from a storage, they process it and keep it in another storage. Storage in **Storage** class is any kind of warehouse for either intermediate or final products.

Pipe and **Filter** classes collaborate between them to model connections among equipment item and storages. A pipe in **Pipe** class represents a connection between an equipment item and another one. A filter in **Filter** class is a node that allows linking connections and represents a shared part between connections.

A cluster in **Cluster** class is a set of equipment items, which may be considered as a sole equipment for the scheduling. There are activities that can be alternatively executed by two or more equipment items. The equipment item assignation is realized during the scheduling. In this case, an alternative object can be instantiated in **Alternative** class, which is composed of these two or more equipment items; and then, to specify that the activity requires one of these alternatives (equipment items) in **Alternative** class.

5.3 Production Process

A process in **Process** class is a set of stages required to obtain a product. A process is related to other processes by means of constraints between processes in **ProcessConstraint** class. These constraints belong to following types: "process *P1* must start after process *P2* ends", "process *P1* involves processes *P2* and *P3*". In this case, a processes hierarchy is achieved

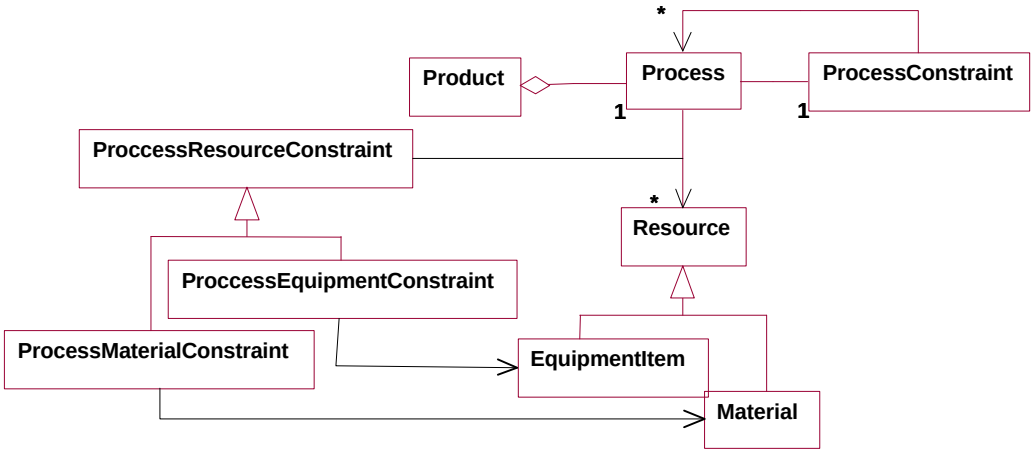

Figure 9: Production process class diagram

The diagram in Figure 9 shows the classes for production processes. Processes in **Process** class require resources in **Resource** class, which implies constraints in **ProcessResourceConstraint** class. These constraints belong to types: "process *P1* requires equipment *E1*", "process *P1* consumes material *M1*", "process *P1* produces material *M2*".

Consume and produce type constraints have two parameters: the *base*, which indicates in which units (volume, mass) it is measured, and the *factor***,** which indicates the proportion of a resource produced or consumed by the process.

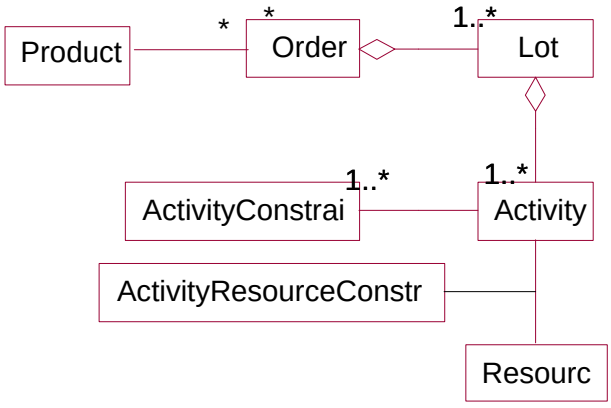

Figure 10: production orders class diagram

5.4 Production Orders

More closely related to the Scheduling, there are production orders. The diagram in Figure 10 shows the classes for production orders. A production order in **Order** class is divided into n lots in **lot** class. Each lot consists of a set of activities in **Activity** class that are associated to resources in **Resource** class by means of constraints in **ActivityResourceConstraint** class. As in the process definition, activities are associated to other activities by means of the constraints among activities in **ActivityConstraint** class.

The association between lots in **Lot** class and stages of a plant train in **Stage** class (Figure 6) should be highlighted. Although it is not shown in the diagram, this association occurs because a resource is a equipment item of the plant and these equipment items belong to a particular stage of a train in the plant.

5.5 Order Instantiation Process

The figure 11 shows the process of instantiating orders based on the production process associated to a product. This is a complex process that requires particular techniques to define size and amount of production lots in **Lot** class for an ordered quantity in **Order** class of the product in **Product** class.

The idea is to instantiate the activities of a lot in **Activity** class according to the process information, as well as constraints among activities in **ActivityConstraint** class and constraints between activities and resources in **ActivityResourceConstraint** class.

In other words, the information required for instantiating activities is contained in the process definition, the resource characteristics and the plant distribution (stages, trains). The way in which this process should be performed is defined by the lot algorithms. Although it is easy to carry out a heuristic that creates an activity for each process, it is also desirable to define different algorithms that will depend on the particular problem.

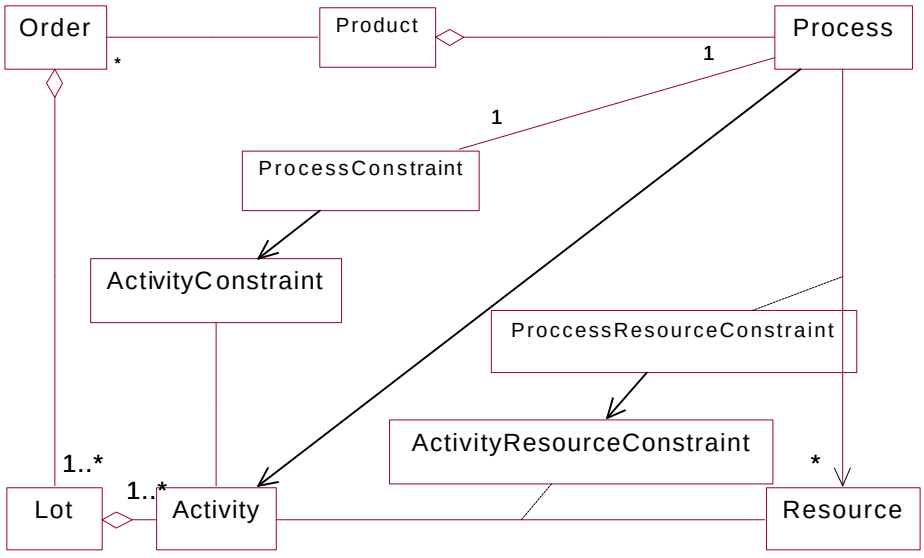

Figure 11: order instantiation process

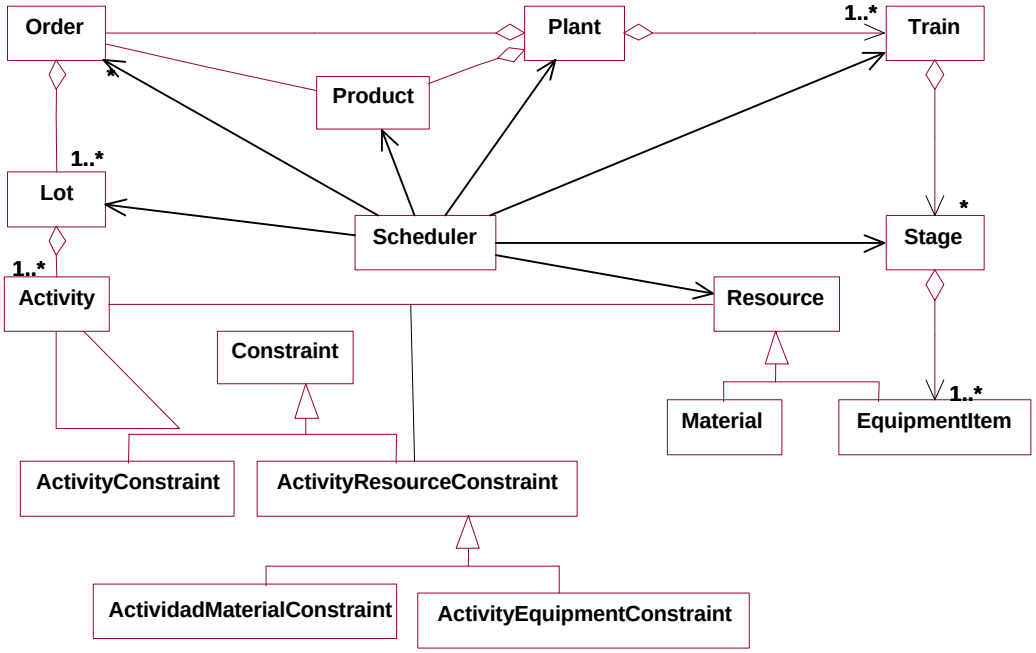

Figure 12: scheduling process class diagram

One of the characteristics of our design is that the relationships among the items involved in the scheduling problem make it possible to solve not only the scheduling itself, but also the *lot sizing* problem, which is also a typical and important problem in the process industry.

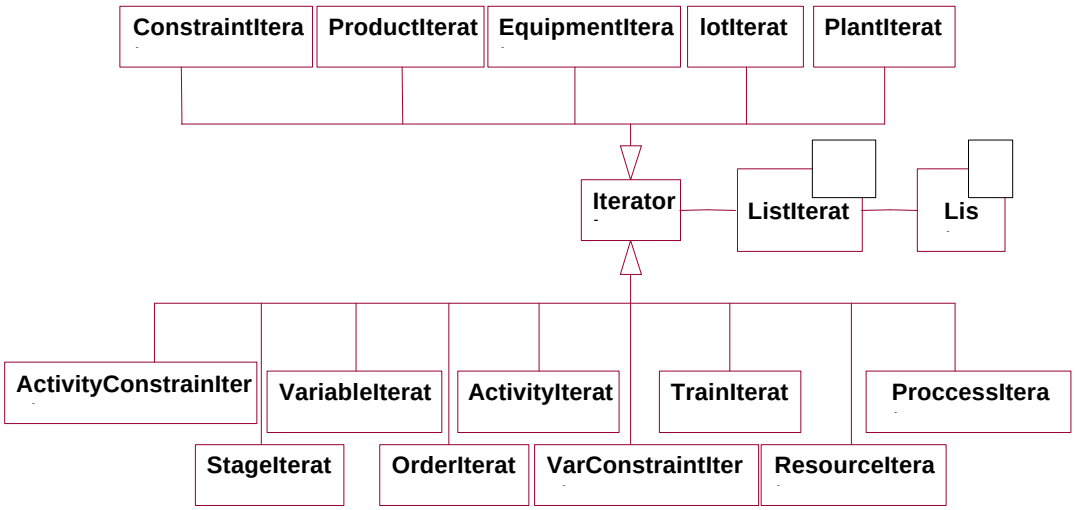

Figure 13: PFS iterators

5.6 Scheduling

The scheduling problem implies a complex relationship among all classes previously defined. To reduce complexity, we use the design pattern *Farcade* (Gamma, 1995). The resulting class diagram is shown in Figure 12. We have defined the **Scheduler** class, which provides a unified interface to a set of interfaces of the subsystem. In this way, the **Scheduler** class is a higher-level interface that makes the subsystem easier to use, since it minimizes communication and dependence among subsystems. That is, the **Scheduler** class can

act as an interface so that external objects can access to instances of the classes that compose a scheduler in **Scheduler** class. This infers the convenience of using the iterator technique because when an iterator is to be created, it may require the scheduler's first component and successively the following ones. In this way, it makes it possible to easily expand the access points to the PFS objects. Figure 13 shows the different iterators we have defined following the design pattern *Iterator* (Gamma, 1995).

6. Implementation details

The PFS system can be implemented by using the class libraries provided by Ilog Solver and Ilog Scheduler(Le Pape, 1994), which allow to use optimization techniques and code reuse. This strategy consists of assigning the responsibility of instantiating Ilog objects to each PFS class, with the aim of using the functions already defined in these objects. In this way, the Ilog class functions are increased with the aim of including the PFS characteristics. The elemental attributes of the PFS classes can be defined as instances of the class type provided by Ilog. For example, The capacity of a storage can be defined as an instance of the IlcIntVar class. Storage capacity is assigned a domain [Min, Max] and the ability of returning to a previous state, which is appropriate for the optimization algorithms. This implementation strategy must be used with the remaining PFS classes, specially with the constraints, since it is a highly proved technology of Ilog.

Under this strategy, the PFS system can be considered as a three-layer system, as it is schematically represented in Figure 14. The first layer is Ilog, which is the less abstract level. The second layer is defined by PFS, which is based on the Ilog layer to provide a more abstraction level oriented to support the representation of flow process scheduling problems. The third layer is composed by algorithms, which use the PFS functions to obtain a schedule. Finally, the PFS's clients, for example, the manager defined in the architecture presented in Section 3, uses the algorithms and PFS's functions.

Finally, we can say that by combining the knowledge about the domain provided by PFS with the optimization capabilities of a proved solver, an appropriate framework for process flow scheduling can be obtained.

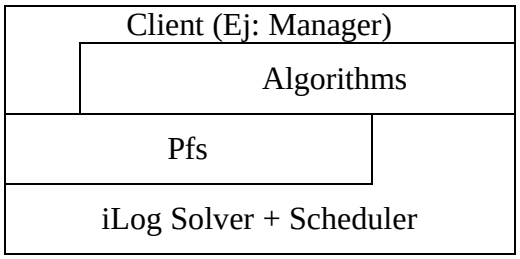

Figure 14: schematic representation a PFS framework

7. Conclusions

The complexity of the flow process scheduling problem has been met using the design pattern *Farcade*. It provides a simple default view of the scheduler system that is good enough for most clients. It shields clients from scheduler components, thereby reducing the number of objects that clients deal with and making the scheduler system easier to use.

The system architecture defined to support the flow process scheduling problem, separates the problem representation from the problem solution representation, and also allows to develop the elements common to the scheduling system. Data persistency is also freed from the technology to be used. The main advantage of this strategy is that it allows to focus on the flow process aspects, so as to identify the participating elements to represent the problem, to design solution models and to test them.

What has been previously described allows us to consider the designed PFS system as an object oriented framework for process flow scheduling. In this framework the problem representation remains notably separated from the solution techniques. Using PFS system a particular industry can be instantiated; and also, PFS system can be used to test solving algorithms that combines both optimization and heuristics techniques.

Reference

Allweyer, Th.; P. Loos; A. W. Scheer, A.-W., *Requirements and New Concepts for Production Planning and Scheduling in the Process Industries*. In Fransoo, J. C.; Rutten, W. G. M. (eds.): Proceedings of the Second International Conference on Computer Integrated Manufacturing in the Process Industries, pp. 4-17, Eindhoven, (1996)

Booch, G., J. Rumbauch and I Jacobson, *The Unified Modeling Language User Guide,* A. Wesley, (1999)

Gamma E., R.Helm, R.Johnson, and J.Vlissides, *Design Patterns: Elements of Reusabel Object-Oriented Software,* Addison-Wesley, 1995.

Le Pape, C., *Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems*. Intelligent Systems Engineering, 3(2):55-66, (1994)

Taylor, S.G. and S. Bolander, *Process Flow scheduling, A scheduling systems framework for flow manufacturing*, Library of Congress, (1994)

Smith, S.F. and M.A. Becker, *An ontology for Constructing Scheduling Systems*, AAAI Symposium on Ontological Engineering, Standfor, CA, March (1997).