Frameworks Derived from Business Process Patterns

Oscar Barros [a,*], Samuel Varas [a]

[a] *Industrial Engineering Department*
University of Chile
República 701, Santiago, Chile

Abstract

A novel approach for the design of Business Objects Frameworks that encapsulates high level business knowledge and logic is presented. These frameworks are derived from formal and explicit Business Process Patterns that include best practices for businesses in a given application domain. A pattern and a framework derived from it can be applied to improve a process for a given business in the domain and to develop an application to support such process. This provides a very flexible way, based on reusable components, to develop solutions and software for complex business decisions, which is an alternative to packaged products. The approach is exemplified by using a specific application domain and applied to a real case in the domain.

Key words: Business Patterns, Business Objects, Framework, Software Development

1 Introduction

Several authors (Bohrer et al., 1998; Cline and Girou, 2000; D'Sousa and Nills, 1999; Fan et al., 2000) have established the need of Business Objects that represent things and behavior in a business domain and provide a solution to generalized, recurring problems in it. Such Business Objects (BO) would be organized in a framework (Cline and Girou, 2000; D'Sousa and Nills, 1999; Fowler, 1996), which is not necessarily executable, that can be adapted and

* Corresponding author, Phone 5626784403, fax 5626897895.
Email addresses: obarros@dii.uchile.cl (Oscar Barros), svaras@dii.uchile.cl (Samuel Varas).

specialized to solve particular business problems. The value of a Business Objects Framework (BOF) depends on the relevance -in terms of impact on business results- of the business situation its represents, the quality of the support it gives to such situation and the effort needed to make it work.

Examples of specific well known attempts to implement ideas above are as follows:

i) The San Francisco Project (Bohrer et al., 1998) that, based on requirements derived for a vertical domain defined by several IBM's business partners, developed an extendable component-based development platform. This includes basic business logic for common business functions -e.g. financial management, order management and the like- to be enhanced and extended by developers; Common Business Objects (CBO) that perform processing functions used in many applications domains; and a Foundation, which provides an infrastructure that is used to build the business logic and the CBO. These components were commercially available for a few years and are no longer marketed by IBM.

ii) Fowler's patterns (Fowler, 1996), that are published frameworks in domains such as accounting, billing and payroll. They identify object structures and associated logic that synthesize generalized solution in such domains. The logic considered is mostly information processing logic and not true decision oriented business logic.

iii) The Catalysis approach (D'Sousa and Nills, 1999), which proposes frameworks similar to Fowler's, but for a wider range of domains. It attempts to cover some business decision logic, but at a basic naive level.

All above approaches share a common weakness, which is that they do not start with an explicit business process domain model that defines with precision the high level decision logic needed to run a business according to best practices.

In trying to overcome above limitation, we have developed a new approach to design and produce BOF, which is novel in that:

i) It is based on formal models of generalized business processes for a given domain -called Business Process Patterns (BPP)- which include high level logic derived from best practices that assure a well run business (Barros, 2000).

ii) It is systemic since business logic for each activity of a process -e.g. marketing, selling, order processing, producing and distributing - is consistent and integrated with the whole.

iii) It can be naturally connected with UML modelling of BO.

iv) BO include business logic that offers alternatives and incremental levels of complexity and sophistication for supporting a business activity.

v) It is open in that BPP and BOF are published for wide use in a web site (www.obarros.cl, 2003).

In summary, the most distinctive characteristic of our approach is that it is closer to the most important decisions of a business than any previously proposed framework and provides a very flexible, reusable component-based approach for supporting such decisions.

It has been experimentally tested in real-life situations in Chile.

2 Business Process Patterns

Business Process Patterns (BPP) are models of how a business in a given domain should be run, according to the best practices known (Barros, 2000). Hence they are based on empirical knowledge of how activities of a process in the best companies of a given domain are performed. Such knowledge can be obtained from books (Hieleber et al., 1998), web sites (www.bwpccoe.org, 2003; www.ebusinessforum.com, 2003; www.siebel.com/bestpractices, 2003) and direct observation of firms. Our patterns have benefited from the knowledge derived by hundred of cases in which processes of many different companies have been modelled, analyzed and redesigned [1].

We have found that beyond specific best practices for a given domain -usually expressed in the form of an specific business logic-, BPP share a common structure of activities and flows. Thus, products or services provision processes -such as manufactured goods, health services, justice services, financial services, etc.- share such a common structure. A first level of detail of such a process structure for a very large domain is shown in Figure 1, where an activity-based modelling scheme that uses IDEF0 is presented (Barros, 2000). This pattern is a more precise version of the value chain of a firm (Porter, 1986). Such BPP establishes what activities and relationships, by means of information flows, in the model should exist in practice in order that the business it realizes is well run. One activity in the model is of particular interest, since if represents the centralized IT-based storage of data needed to support the process, which is called *State Status*. Thus the BPP assumes that every transaction that occurs in the activities other that *State Status* is informed to this, and state of relevant entities is updated and fed back to former activities, so that they can act upon such knowledge.

Detail of flows -by means of attributes definition- and actions of activities, described by business logic, is given in the BPP dictionary(www.obarros.cl, 2003).

[1] Representative cases are published in the web site www.obarros.cl (in Spanish)

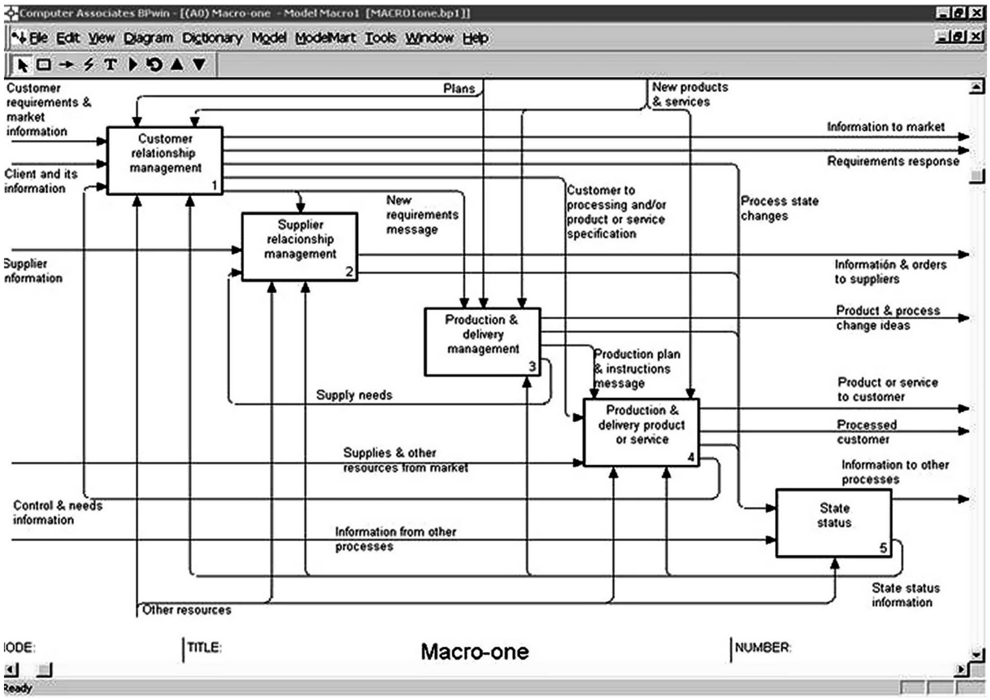

Fig. 1. Business Process Pattern for value chain

Further detail of any activity can be given by decomposition of it, following the IDEF0 scheme. For example, Figure 2 shows the detail of activity 3. At this level of detail our domain is still as general as the first one.

If we want to give further detail, we have to be more specific about the domain, so that we can define the business logic and flows with precision. In order to show how to do this and use the same case for the rest of the document, we synthesize our experience of many real cases in the following domain definition for the activity *Production planning and control* of Figure 2 (Barros, 1995).

We assume we use physical installations to produce a product or give a service, where there is a physical entity which is the final product or the one that receives the service. This domain represents situations such as manufacturing, health services, justice services, and telecommunication services.

Under theses assumptions, we decompose *Production planning and control* as shown in Figure 3.

Finally, to give more details of activity *Scheduling* of Figure 3, we reduce the domain to situations in which tasks are processed on physical facilities in lots -predefined by *Planning* and known by means of *Production plan*. We also assume that when changing from one lot to another a set-up cost, which depends on the pair involved, is incurred. Such set-up costs are assumed to be known for all pairs. This case is representative of some situations in manufacturing and other business, such as paper mill machine processing of lots, where color

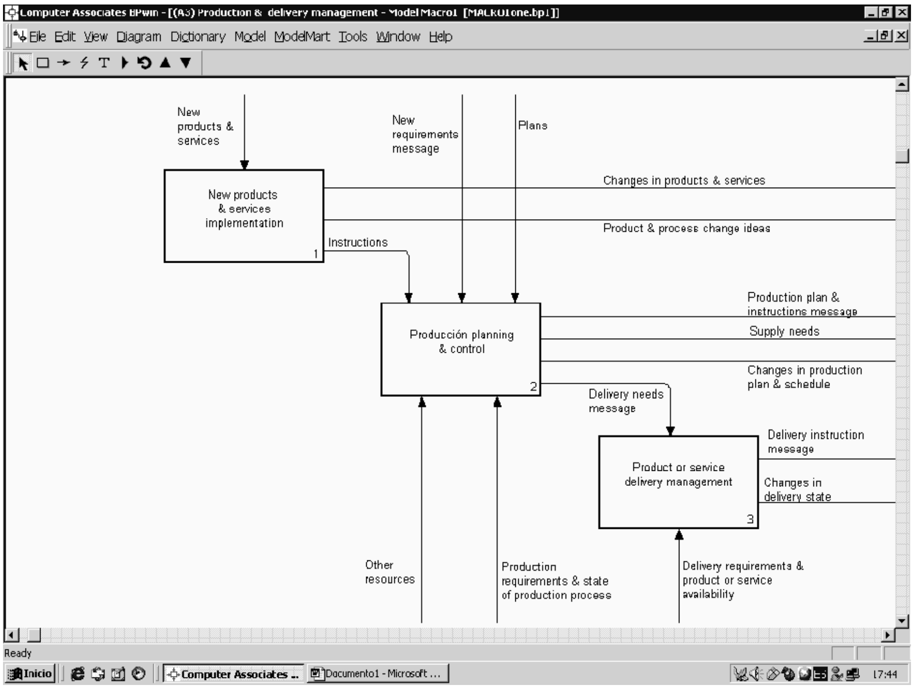

Fig. 2. Detail of *Production & delivery management*

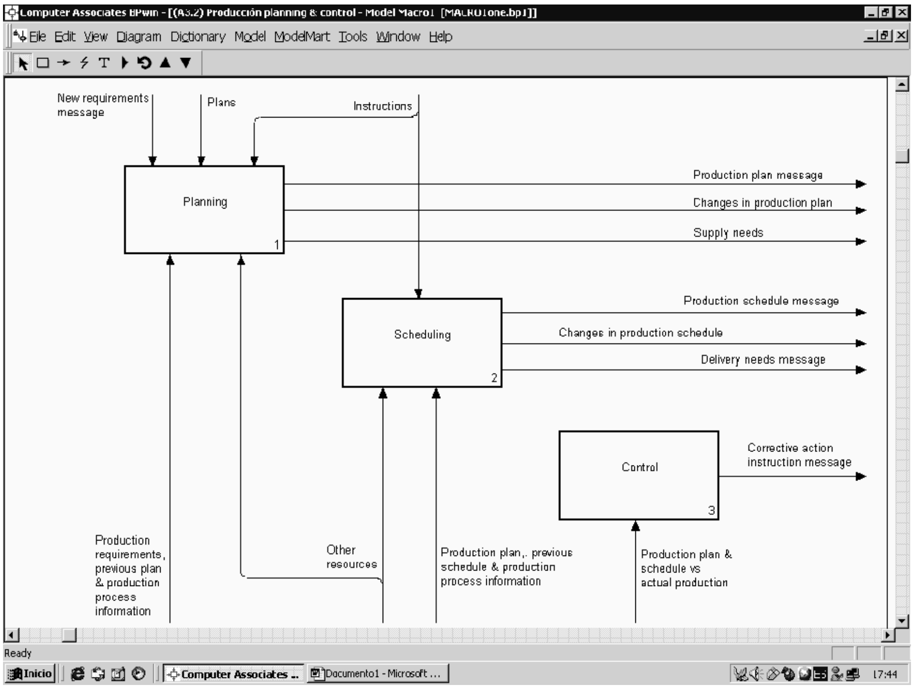

Fig. 3. Detail of *Production planning and control*

of paper of a preceding lot affect the set-up of the following; printing shops; processing of patients in surgical installations -because of cleaning and equipment set-up between operations-; processing of batches in food industry lines; and assignment to technicians and routing of telephone repair calls.

At this specific domain we can be very precise about the business logic that produces an optimal or near optimal solution - in terms of cost minimization - which means a best practice. Business logic, which guides the action of an activity, determines the exact information flows that are supplied and that are produced. We will show how such logic is specified in the next section.

We have given a third level of detail of just one activity of a given domain. In a real-life situation, where a BPP is to be used to redesign a whole process, all the lowest level activities of it should be detailed, which we do not do here, because we are just presenting the way our approach works. Also all the logic for the different activities should be consistent, since they generate the flows that allow the interaction among themselves, as shown in Figures 1, 2 and 3. Thus, for example, the logic for producing *Production plan* in Figure 3 should the right one in terms of the definition of lots needed by *Scheduling* in the same figure, which is known by means of *State Status*.

Of course, BPP can be developed for any business domain of interest, which, besides the cases presented, may include new product development, business planning, human resource management, financial resource management, etc.

3 Business Logic Specification

Our aim is to give generalized business logic for an specific domain. In the case we are presenting, we have defined our domain, as outlined in previous section, as a situation -representative of many real-life experiences - which can be formalized as follows:

Consider the case where n tasks must be processed on at most m facilities following a defined route r. Each task is characterized by its type, which is a group of similar tasks having the same lead, processing and setup times at each facility. Each facility is characterized by its capacity, given by the number of similar and parallel facilities and their technical characteristics. Finally, a route is defined by a sequence of facilities which participate in the processing of a given task or group of them. Figure 4 shows the general setting of our characterization, where there are 3 tasks (t_1, t_2, t_3), 6 facilities $(F_1, F_2, F_3, F_4, F_5, F_6)$, 3 routes, where for example route for task t_1 is (F_1, F_2, F_3, F_6), t_2 is (F_1, F_4, F_5, F_6), and t_3 is (F_4, F_3), and facility 2 has two parallel facilities.

Then, the goal is to schedule the n tasks, processing them in the order required by their routes, such that all or at least most of the tasks are completed before of their lead times, minimizing time or cost, or maximizing facilities utilization. It is well known that the standard tasks scheduling problem is NP-hard (Garey

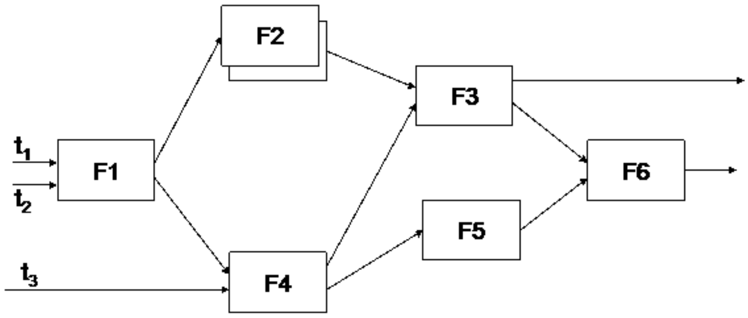

Fig. 4. Scheduling Problem characterization

and Johnson, 1979; Garey et al., 1976), because it is a strong combinatorial problem. However, there are some cases where it is possible to have good solutions in polynomial time, which we specify below.

To further formalize our scheduling problem, we consider the following notation:

- m represents the number of different types of facilities or machines, where we consider that there are m_j equivalent parallel facilities of j^{th} type.
- n is the number of distinct tasks types, where n_{ij} is the number of tasks type i^{th} waiting for being processed at facility type j. Then, $n_j = n_{1j} + \ldots + n_{Nj}$ is the number of tasks to be scheduled at facility type j.
- P_{ik} is the processing time of task i at facility k.
- S_{ijk} is the setup time if task type i is processed immediately before task type j at facility k. Therefore, matrix S_j represents the setup time matrix for the j_{th} facility.
- T_{W1i} is earliest time to begin processing of task i.
- T_{W2i} is the latest time to finish of processing task i. This is also known as due or lead time.

The scheduling problem of n tasks at m facilities is an order $\vec{\pi} = (\vec{\pi}_1, \vec{\pi}_2, \ldots, \vec{\pi}_m)$ of tasks to be processed at each facility. In particular, at any specific facility k, the processing order $\vec{\pi}_k$ is a n_k-tuple $(\pi_{1k}, \pi_{2k}, \ldots, \pi_{n_k k})$, where π_{ik} is the task that will be processed in order i^{th} at facility k.

Then, for initial waiting tasks distribution $\vec{n}^0$ and m facilities, the following heuristic provide a scheduling logic for our problem.

Heuristic Schedule$(\vec{n}^0, m)$

$\pi_j = 0,\ j = 1, \ldots, m$

$\vec{n} = \vec{n}^0$

$SelectSet(m, \vec{n}, \Phi)$

$GetTask(\Phi, \vec{\pi})$

$if\ (m > 1)\ then$

$\quad ImproveSchedule(\vec{\pi})$

where

- Routine $SelectSet()$ selects a subset of tasks for every facility, where the result is given in the m-tuple $(\Phi_1, \Phi_2, \ldots, \Phi_m)$ of subsets of Φ.
- Routine $GetTask()$ selects a sequence π_j for the set of tasks Φ_j of each facility $j, 1 \leq j \leq m$; its implementation will depend on the specific case.
- Routine $ImproveSchedule()$ improves the current schedule π for all facilities.

This heuristic synthesizes many proposals of algorithms and heuristics for solution of problems in the domain (Beck et al., June 9-13, 2003; Johnson, 1954; Thangiah et al., 1996).

In what follows we provide solutions (business logic) for both $SelectSet()$ and $GetTask()$ routines. Logic for $ImproveSchedule()$ is provides when relevant. These solutions will depend on the characteristics of facilities, set-up times and lead times; some of them are proved optimum and the others are heuristics. We will concentrate on simpler cases in order to avoid very complex logic. However, these cases are useful for solving relevant real life cases, as we will show in Section 5.

First, we structure the domain by defining a hierarchy of cases, going from simple (at the top) to more complex (at bottom), as shown in Figure 5. This is also an inheritance tree, since an algorithm or heuristics on a given case(node) of the tree is applicable to cases on lower nodes of such branch. This can also be thought as an specification of generalized business logic with alternative and incremental options, as proposed in (Barros, 2002). Thus a node with two branches represents alternative solutions (business logic specified as an algorithm or a heuristic) to the scheduling activity for different situations. A node in a branch that follows another one represents a more complex case that uses an algorithm or heuristic from the simpler one as part of the solution for that node. This characteristic will be exploited in the next section to define frameworks with specialization hierarchies and incremental methods. The hierarchy is simplified, for presentation purposes, since it does not consider all the possible cases for the situation at hand.

Next, we give real-life examples of situations for each node in the tree in Figure

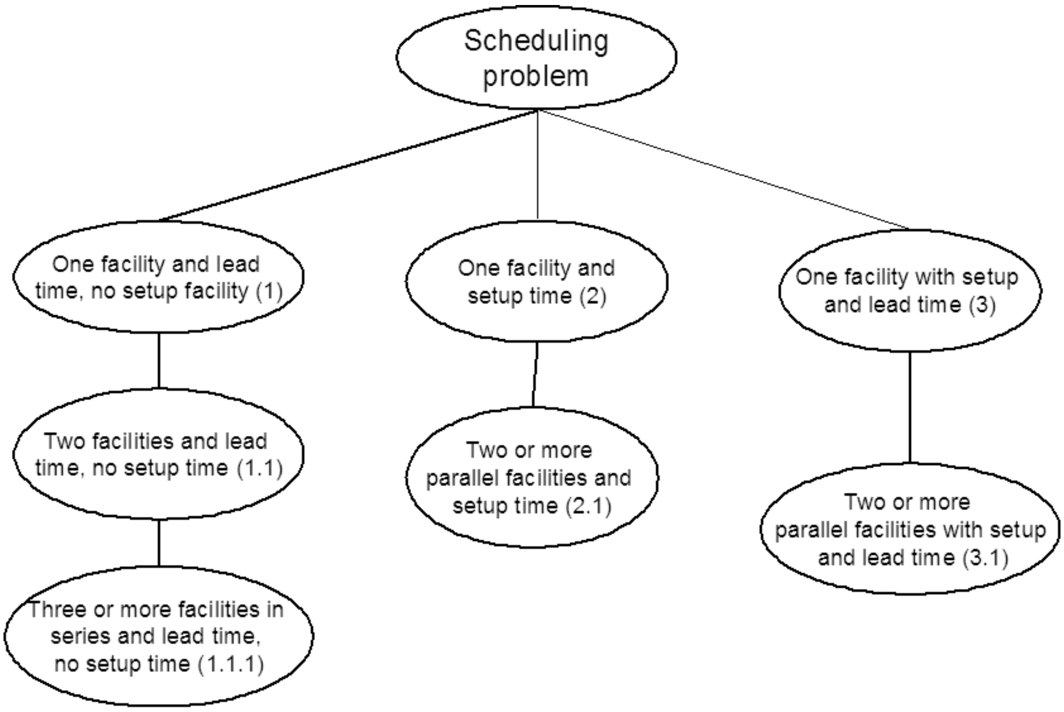

Fig. 5. Scheduling Problem Structure

5 -to show the practical relevance of the cases- and detail the business logic for each of them.

1. One facility; tasks with lead times, but no set-up times. This case corresponds to situations with simple machines that require little or no set-up time, or complex installations where set-up times have been eliminated to allow for just-in-time production; e.g. a sewing machine in a textile shop. In this case there is not need for $SelectSet()$. A heuristic for $GetTask()$ that tries to minimize average completion time is:

   GetTask1(Φ, π)
   $q = argmin_j\{\alpha\, P_{jk} + \beta\, T_{W2j} \mid j \in \Phi\}$
   $\pi = insert(q, \emptyset, 0)$
   $\Phi = \Phi \backslash \{q\}$
   $while(\Phi \neq \emptyset)$
   $\{$
   $q = argmin_j\{\alpha\, P_{jk} + \beta\, Tardiness(\hat{\pi}) \mid j \in \Phi \wedge \hat{\pi} = insert(j, \pi, |\pi|)\}$
   $\pi = insert(q, \pi, |\pi|)$
   $\Phi = \Phi \backslash \{q\}$
   $\}$

   where α and β are real nonnegative parameters, such that $\alpha + \beta = 1$, $insert(j, \pi, i)$ return a new schedule with element j inserted in schedule π just after place i, and function $Tardiness(\pi)$ calculates the total amount of tardiness time of schedule π. This function is given by the following expression:

$$
\begin{array}{l}
\textbf{Tardiness}(\pi) \\
\quad t = 0 \\
\quad for\, i = 1\, to\, |\pi| \\
\quad\quad t = t + min\{0; Time(\pi, i) - T_{W2\pi(i)}\} \\
\quad return\, t
\end{array}
$$

Function $Time(\pi, k)$ is a function to calculate the execution time of schedule π until task k, given by the following expression:

$$
\begin{array}{l}
\textbf{Time}(\pi, k) \\
\quad t = P_{\pi(1)} \\
\quad for\, i = 2\, to\, k \\
\quad\quad t = P_{\pi(i)} + max\{t; T_{W1\pi(i)}\} \\
\quad return\, t
\end{array}
$$

This heuristic is an adaptation of the one proposed in (Thangiah et al., 1996).

Function $GetTask1()$ provides the optimal solution with respect to minimizing the makespan, under no earliest and lead times (i.e., $T_{W1i} = T_{W2i} = 0, \forall i$) and one or two facilities (i.e., $m \leq 2$). In this case, $GetTask()$ correspond to the Johnson's algorithm (Johnson, 1954).

1.1. Two machines in series. In this case tasks should be sequenced on both machines. An example of this is the sequencing of cutting an sewing in a textile shop. Heuristics for $SelectSet()$ and $GetTask()$, adapted from (Johnson, 1954), in this case are:

$$
\begin{array}{l}
\textbf{SelectSet11}(m, \vec{n}, \Phi) \\
\quad \Phi_1 = \{i \setminus P_{1i} \leq P_{2i}\} \\
\quad \Phi_2 = \{j \setminus P_{1j} > P_{2j}\} \\
\quad \Phi = \{\Phi_1, \Phi_2\}
\end{array}
$$

and

$$
\begin{array}{l}
\textbf{GetTask11}(\Phi, \pi) \\
\quad \textbf{GetTask1}(\Phi_1, \pi_1) \\
\quad \textbf{GetTask1}(\Phi_2, \pi_2) \\
\quad \pi = \pi_1 \bigcup \pi_2^{-1}
\end{array}
$$

where π^{-1} means the inverse order of π.

1.1.1. Same as 1.1, but with more than two machines in series. An example of this is a textile shop with a group of machines that perform given operations -cutting, sewing, finishing, etc.- where a given lot of goods goes through several machines. In this case strategy is grouping first k facilities at the initial virtual facility (FC_1) and the other $m - k$ at the second virtual facility (FC_2). $SelectSet()$ is as follows (Johnson, 1954):

SelectSet111$(m, \vec{n}, \Phi)$
$FC_1 = \{1, \ldots, k\}$
$FC_2 = \{k+1, \ldots, m\}$
$P_{1i} = \sum_{j \in FC_1} P_{ij}, \forall\, i$
$P_{2i} = \sum_{j \in FC_2} P_{ij}, \forall\, i$
SelectSet11$(2, \vec{n}, \Phi)$

and $GetTask()$:

GetTask111(Φ, π)
GetTask11(Φ, π)

This procedure schedules a set of tasks over a line of m facilities and there is no guarantee about the optimally, but we provide a version of $ImproveSchedule()$ to seek a good solution. The $ImproveSchedule()$ logic analyzes all possible subset of facilities and it is as follows:

ImproveSchedule()
$\pi_{min} = \emptyset$
$min = \infty$
$for\; k = 1\; to\; m - 1$
$\{$
SelectSet111$(m, \vec{n}, \Phi)$
GetTask11(Φ, π)
$if\; (Time(\pi, |\pi|) < min)$
$min = Time(\pi, |\pi|)$
$\pi_{min} = \pi$
$\}$
$\}$

2. One facility with set-up time, but no lead time. This is a case where set-up is unavoidable and significant (several hours); lead time has been taken care of in production planning or it is not relevant. Examples of this case are machine scheduling in a paper mill, where each machine is scheduled independently for certain papers and lots to be scheduled are part of a production plan for stock replenishment, which has already consider the timing (www.obarros.cl, 2003); and printing machine schedule in the case where there are not desired completion times. We consider that there exists a set-up time, independently of the number of tasks of the same type to be processed, but depending on the previous task type processed.

   Solution is given by a greedy heuristic (Johnson, 1954), which tries to minimize the sum of the set-up and processing times for the sequence of all tasks, where $GetTask(\Phi, \pi)$ is as follows:

GetTask2(Φ, π)
$q = argmin_i\{min_j\{S_{ijk} + P_{jk}| \, i, j \in \Phi\}\}$
$\pi = insert(q, \emptyset, 0)$
$\Phi = \Phi \backslash \{q\}$
$while \; (\Phi \neq \emptyset)$
$\{$
$q = argmin_j\{S_{qjk} + P_{jk}| \, j \in \Phi\}$
$\pi = insert(q, \pi, |\pi|)$
$\Phi = \Phi \backslash \{q\}$
$\}$

2.1. Same as (2), but with several parallel facilities. An example of this is a group of telephone repairmen, which are assigned repair jobs each morning from a list of pending jobs. Set-up time between repair jobs is the travelling time between repair locations. Each repairmen has to be assigned a set of jobs and a sequence (schedule) of repairs (www.obarros.cl, 2003). We define Ω as the set of facilities; ω a function to order those facilities; and C_k is the maximum allowed capacity for facility k and c_k is the current used capacity at facility k. Then, the heuristic solution for this case is given by:

SelectSet21$(m, \vec{n}, \Phi)$
$\Theta = \{1, \ldots, n\}$
$\tau_k = \frac{1}{n(n-1)} \sum_{i,j}^{n} (S_{ijk} + P_{jk}), \; \forall k$
$while \; (\Theta \neq \emptyset)$
$\{$
$\Omega = \{1, \ldots, m\}$
$while(\Omega \neq \emptyset \wedge \Theta \neq \emptyset)$
$\{$
$k = arg \min_o\{\omega(o) \mid o \in \Omega\}$
$C_k = 0$
$\Omega = \Omega \backslash \{k\}$
$for \; i = 1 \; to \; |\Theta|$
$if(c_k + \tau_k \leq C_k \wedge \Gamma(\Theta_i, k)) \; then$
$\Phi_k = \Phi_k \cup \{\Theta_i\}$
$c_k = c_k + \tau_k$
$\Theta = \Theta \backslash \{\Theta_i\}$
$\}$
$if(\Theta \neq \emptyset) \; then$
$C_k = C_k + \frac{1}{m} \sum_{j=1}^{m} \tau_j, \; \forall k$
$\}$

where Γ is a belonging function, which is *true* if task i can be processed at facility k and *false* otherwise. The *GetTask*() routine is given by:

GetTask21(Φ, π)
$\pi = \emptyset$
$for\; i = 1\; to\; m$
$\{$
GetTask2(Φ_i, π_i)
$\pi = \pi \cup \pi_i$
$\}$

3 One facility whith setup and lead time. This is the typical case of work to order with given completion dates. We consider that each task has two times: T_{W1i} and T_{W2i}. Examples of this are a printing shop that accepts orders with given due dates and a paper mill that produces orders of special papers with promised delivery dates. Solution is:

GetTask3(Φ, π)
$q = arg\min_j\{\min_i\{\alpha\,(S_{ijk} + P_{jk}) + \beta\, T_{W2j} \mid i, j \in \Phi\}\}$
$\pi = insert(q, \emptyset, 0)$
$\Phi = \Phi \backslash \{q\}$
$while\;(\Phi \neq \emptyset)$
$\{$
$t_{min} = \infty; I_{min} = -1; J_{min} = -1$
$for\; i = 1\; to\; |\pi|$
$for\; j = 1\; to\; |\Phi|$
$\hat{\pi} = insert(j, \pi, i)$
$if\;(\alpha\, Time(\hat{\pi}, |\hat{\pi}|) + \beta\, Tardiness(\hat{\pi}) < t_{min})$
$t_{min} = \alpha\, Time(\hat{\pi}, |\hat{\pi}|) + \beta\, Tardiness(\hat{\pi})$
$I_{min} = i; J_{min} = j$
$\pi = insert(J_{min}, \pi, I_{min})$
$\Phi = \Phi \backslash \{J_{min}\}$
$\}$

This heuristic, modified from (Thangiah et al., 1996), which tries to minimize the overall execution and tardiness time, first selects a task with the largest set-up time with respect to the other tasks and the lowest due time. Then, all unassigned tasks are checked at any possible place of π, and that which minimizes the total execution and tardiness is inserted at such a place. This procedure is executed until there are no unassigned tasks. If there is no lead time violation, the heuristic tries to minimize the total set-up and execution time.

In this case function $Time(\pi, k)$ is modified by including the set-up time into its result, and it is given by the following expression.

Time(π, k)
$t = P_{\pi(1)}$
$for\; i = 2\; to\; k$
$t = P_{\pi(i)} + max\{t + S_{\pi(i-1)\pi(i)}; T_{W1\pi(i)}\}$
$return\; t$

3.1. Same as (3) with several machines in parallel. We also consider here only the parallel case for simplification. An example of this case is the scheduling of surgical operations on a set of facilities (www.obarros.cl, 2003). Then, as in case (2.1), the solution is:

SelectSet31$(m, \vec{n}, \Phi)$

$\Theta = \{1, \ldots, n\}$
$\tau_k = \frac{1}{n(n-1)} \sum_{i,j}^{n} (S_{ijk} + P_{jk}),\ \forall k$
$while\ (\Theta \neq \emptyset)$
$\{$
  $\Omega = \{1, \ldots, m\}$
  $while(\Omega \neq \emptyset \wedge \Theta \neq \emptyset)$
  $\{$
    $k = arg\min_o \{\omega(o) \mid o \in \Omega\}$
    $C_k = 0$
    $\Omega = \Omega \backslash \{k\}$
    $for\ i = 1\ to\ |\Theta|$
      $if(c_k + \tau_k \leq C_k \wedge \Gamma(\Theta_i, k))\ then$
        $\Phi_k = \Phi_k \cup \{\Theta_i\}$
        $c_k = c_k + \tau_k$
        $\Theta = \Theta \backslash \{\Theta_i\}$
  $\}$
  $if(\Theta \neq \emptyset)\ then$
    $C_k = C_k + \frac{1}{m} \sum_{j=1}^{m} \tau_j,\ \forall k$
$\}$

Once again, Γ is a belonging function for each task; it could be a machine specialization, geographical location, or any other possible assignment criterion.

Then, the function to schedule each task into an specific facility is given by the following routine.

GetTask31(Φ, π)

$\pi = \emptyset$
$for\ i = 1\ to\ m$
$\{$
  GetTask3(Φ_i, π_i)
  $\pi = \pi \cup \pi_i$
$\}$

What we have presented can also be considered as a pattern that synthesizes and structures knowledge about the solution of very diverse situations in the specified domain, which can be reused by specializing it to specific cases. In the next section we formalize this idea by converting this pattern into a software framework.

4 From Business Process Patterns to Frameworks

From the BPP and business logic of the previous sections, we can derive BOF with BO that incorporate the knowledge about the solution of a relevant problem in the given domain. This BOF has as a purpose to provide a generalized solution to the problem that can be reused to develop an object-based software application for any particular real-life situation in the domain.

The mapping from BPP and business logic to a BOF, as proposed in (Barros, 2002), is as follows:

i) The structure of the business logic of the domain gives a first cut definition of the BO classes that encapsulate the algorithms or heuristics that solve the problem for different cases in the domain. This structure contains, in general, alternative and incremental solutions to different cases in the domain, as shown in Figure 5, for the scheduling problem.

ii) Structure of the BO can then be modelled using UML class diagrams, and operations or methods for classes defined according to business logic.

iii) Data needed to execute operations can then be derived from the parameters included in the business logic.

iv) Data can then be structured into data classes that interact with BO in (*ii*). A complete class diagram with BO and data classes can then be modelled using UML.

We follow steps above for the scheduling problem.

The structure of the business logic in Figure 5 leads us directly into the BO structure of Figure 6, where we also show the operations for each class. Such structure, which is an specialization one, shows that there are methods - *HueristicSchedule*, *SelectSet*, *GetTask*, and *ImproveSchedule* - which are used by all specialization classes. Then the three branches starting a the class *Scheduler*, define three alternative cases, one with lead time, another with setup time, and one with both. Each branch has a method which is an specialization of *GetTask* - $GetTask1()$, $GetTask2()$ and $GetTask3$ - which is inherited by cases immediately below in such branch. Same is true for other branches below this one. All these methods have been specified in the business logic of Section 3.

Derivation of data needed is direct from the algorithms and heuristics of the business logic. Thus task data -number, type, lead time- and set-up data is necessary for the logic. Using well known principles of object oriented design (Pree, 1994) we come out with the class model of Figure 7, where we have integrated it with the BO model. Also we have made same design options, assuming specific implementation technology, separating data and logic in the idea of a web application (Conallen, 1999) and adopting Java as a programming tool. This design has actually been coded using Java and each node in the specialization branches has been made to work with the inherited methods.

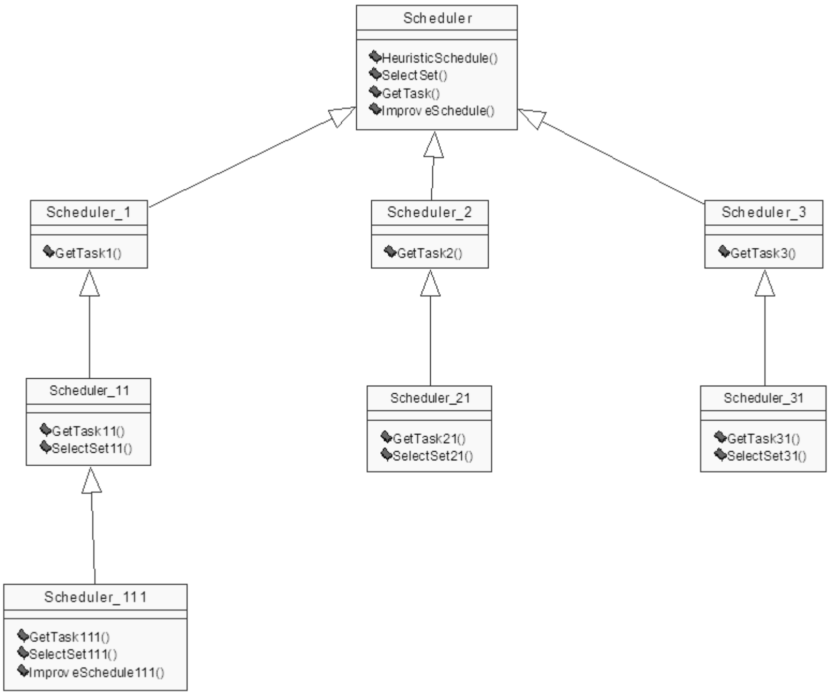

Fig. 6. BO structure

Clearly, the framework is simplified in the sense that includes just what is necessary to run the logic. In real-life situations, other attributes, needed to describe the participating entities, and operations to update data and to elaborate and present the results would be included.

Also this framework has been presented as stand alone, which is not realistic. In some cases this would be integrated with other frameworks for others activities in a process at outlined in Section 2; in others it can be used without integration, but it should be at least connected to the business data bases, which contain data needed by the framework, instead of duplicating it.

We have developed working frameworks for several activities of the process in Figure 1, which contain best practices that can be automated in applications to support such activities. In particular we have frameworks for customer evaluation and order processing which includes automated customer classification based on history and balance sheet information; sales forecasting and planning that includes sophisticated analytical tools, such us ARIMA methods and neural nets; and inventory management that includes JIT and Reorder Point cases, with probabilistic considerations for demand and lead times.

These frameworks are now being routinely used to develop specific applications

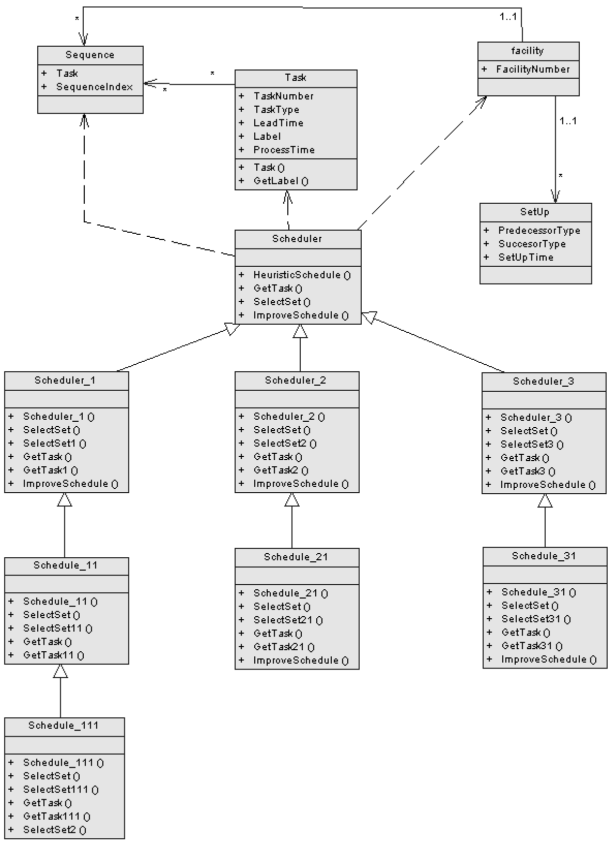

Fig. 7. BOF for Scheduling

for real-life situations in their domains, as part of formal projects performed by students at the University of Chile to help companies in this country to innovate in their management and improve IT support to it (www.obarros.cl, 2003).

5 Using Frameworks for Application Development

In using a framework of the type derived in Section 4 for developing an application to support a real-life case within the frameworks domain, the following procedure is used (Barros, 2002):

- Select relevant substructure of the framework applicable to the case.
- Specialize substructure to the characteristics of the case, adding data and logic as needed.
- Design in detail for an available or selected technology and code.

We illustrate this procedure with the *Scheduling* framework. For this we use a real-life case, which we have actually solved. It deals with the day to day scheduling of telephone repair calls (tasks) for the largest telephone company in Santiago, Chile. In attending such calls, hundreds of repairmen (facilities) are available. Then the problem is to assign calls to each repairman and give him a route to attend the calls. Objective is to minimize sum of all repairmen travel time -equivalent to set-up time between tasks in the general formulation-subject to the maximum work load that can be assigned to each of them. Additionally, we would like that each repairmen has an assigned zone, where he or she will get to be known by customers and develop a good relationship with them; then each repairmen should hopefully be assigned calls in such zone, but trying to keep the work load balanced among repairmen by eventually assigning him -if he has time available- calls from other zones where the repairman is overloaded.

This case corresponds to *Schedule21* (with no lead time) in Figure 7. So the relevant classes for such node in the structure are selected, which are shown in Figure 8.

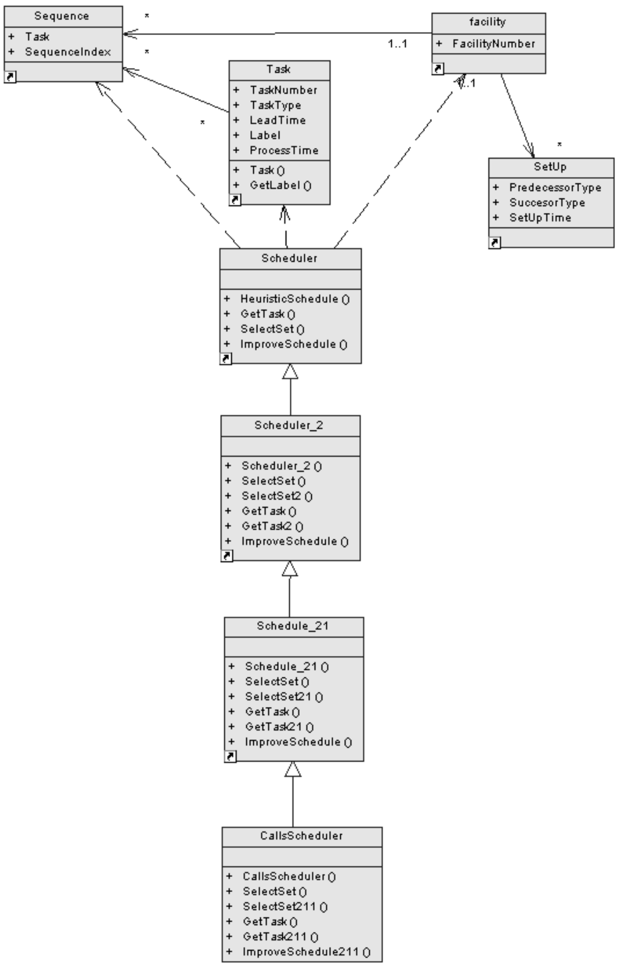

Fig. 8. Relevant Classes for repairmen scheduling

Specialization of Figure 8, according to previous case description, proceeds with adding a logic class *ScheduleCalls* with methods *SelectSet211*, *GetTask211* and *ImproveSchedule211*; a *Repairmen* data class with attribute *zone*; a *RepairCall* data class with attribute *location*; and a class *SetupGenerator*, which calculates distances for all pair of locations of standing calls and creates the data of the *Setup* class. In calculating such distances, a dummy task (node), that represents the location from which all repairmen start, is created. All this classes inherit from the selected structure as shown in Figure 9.

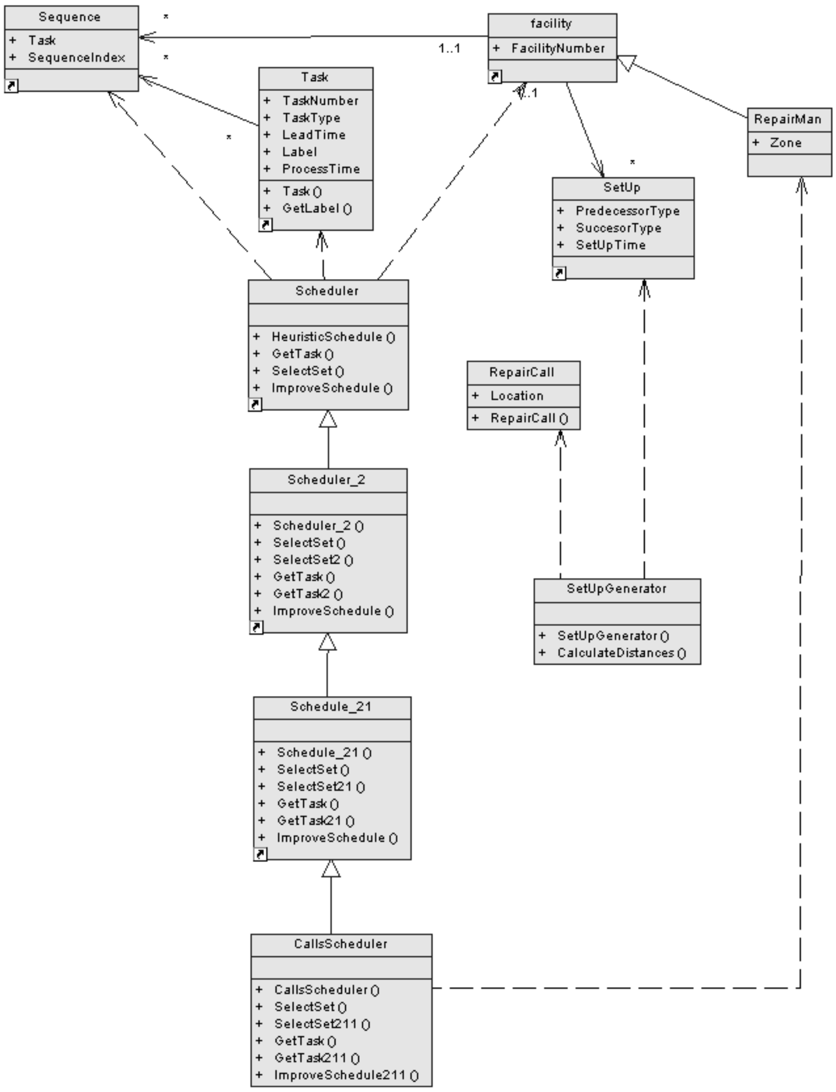

Fig. 9. Specialized Framework for repairmen scheduling

Then the business logic for $SelectSet211$ is obtained from class $SelectSet21$, and the sets are sequenced with $GetTask21(\Phi, \pi)$, where all sequences start at the dummy node defined above. We have added method $ImproveSchedule211()$ to balance the work load among repairmen as follows:

Method ImproveSchedule211$(\vec{n}, m, \vec{\pi})$

$$
\begin{array}{l}
\quad h_k = Time(\pi, |\pi|) - C, \ \forall k \\
\quad \sigma_\pi = \frac{1}{n^2} \sum_{k=1}^{n} (h_k - \bar{h_k})^2 \\
\quad while(\sigma_\pi > \xi) \\
\quad \{ \\
\quad\quad \hat{\pi}_k = \pi_k, \ \forall k \\
\quad\quad \nu = arg \min_k \{h_k\} \\
\quad\quad for \ i = 1 \ to \ |\hat{\pi}_\nu| \\
\quad\quad\quad \kappa = \hat{\pi}_{\nu i} \\
\quad\quad\quad \hat{\pi}_\nu = \hat{\pi}_\nu \backslash \{\hat{\pi}_{\nu i}\} \\
\quad\quad\quad j = 1 \\
\quad\quad\quad while(j \leq m \wedge j \neq \nu) \\
\quad\quad\quad\quad fork = 1 \ to \ |\hat{\pi}_j| \\
\quad\quad\quad\quad\quad \hat{\pi'}_j = inser(\hat{\pi}_{\nu i}, \hat{\pi}_j, k) \\
\quad\quad\quad\quad\quad Slack(\hat{\pi}, \hat{\pi'}_j) \\
\quad\quad\quad\quad\quad if(h_j \leq 0 \wedge \sigma_{\hat{\pi}} \leq \sigma_\pi) \\
\quad\quad\quad\quad\quad\quad \hat{\pi}_j = insert(\hat{\pi}_{\nu i}, \hat{\pi}_j, k) \\
\quad\quad\quad\quad\quad\quad \sigma_\pi = \sigma_{\hat{\pi}} \\
\quad \} \\
\quad \pi_k = \hat{\pi}_k, \ \forall k
\end{array}
$$

We have also coded the solution for this repairmen scheduling problem by taking the Java code developed for the *Scheduling* framework and specializing it according to the additions in this section. In a sequel paper we will report the results obtained in the actual use of the solutions to solve the problem.

6 Conclusions and Future Work

We have shown in detail the workings of our approach for developing BOF based on BPP. This included the application of the example framework to a real life case of moderate complexity. The relevance of such framework and the easiness of its use confirm our claim of its flexibility and reusability in situations were non trivial business logic makes other approaches difficult to implement. So it is apparently feasible to have the best of two worlds in the support of complex business decisions: the advantages of pre built software - with savings in developing costs- and the option to easily customize a solution to the specific characteristics of a given case.

Our research is continuing in several directions. First, we are applying the example framework of this paper to the actual solution of real life assignment and routing problems in companies of the telecommunications industry in Chile. Numerical results of such application will be presented in a sequel paper. Second, such framework is being extended to include cases not included in it; in particular, for situations with several facilities in any configuration. Thirdly, frameworks for other activities in the value chain defined in this paper -customer evaluation, sales predictions and production/supply management- are being perfected. We are also working in the integration of these frameworks; in particular we have developed an integrated framework -which covers the whole value chain- with practices adapted to small and medium sized companies. Finally, we are perfecting the way to deliver these frameworks for practical use, by using technologies such as EJB and web services. A first test of these technologies was done with the framework for small and medium sized companies which was developed using EJB.

Akcnowledgements

Authors appreciate the help of Sebastián Ríos in the coding of the Scheduling framework and its applications to the repairmen scheduling case.

References

Barros, O., 1995. Reingeniería de Procesos de Negocios: Un Enfoque Metodológico. Dolmen.

Barros, O., 2000. Rediseño de Procesos de Negocios mediante el uso de Patrones. Dolmen.

Barros, O., 2002. Componentes de lógica del negocio desarrollados a partir de patrones de procesos. Ingeniería de Sistemas XVI (1), 3–20.

Beck, J., Prosser, P., Selensky, E., June 9-13, 2003. Vehicle routing and job shop scheduling: What's the difference? In: ICAPS 2003 (13th International Conference on Automated Planning and Scheduling). Trento, Italy.

Bohrer, K., Johnson, V., Nilsson, A., Rubin, R., 1998. Business process components for distributed object applications. Communications of the ACM 41 (6), 43–49.

Cline, M., Girou, M., 2000. Enduring business themes. Communications of the ACM 43 (5), 101–106.

Conallen, J., 1999. Modeling web application architectures with UML. Communications of the ACM 42 (10), 63–77.

D'Sousa, D., Nills, A., 1999. Objects Components and Frameworks with UML. Addison-Wesley.

Fan, M., Stallaert, J., Whinston, A., 2000. The adoption and design methodologies of component-based enterprise systems. European Journal of Information Systems (9), 25–35.

Fowler, M., 1996. Analysis Patterns: Reusable Objects Models. Addison-Wesley.
Garey, M., Johnson, D., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman.
Garey, M., Johnson, D., Sethi, R., 1976. The complexity of flowshop and jobshop scheduling. Math. Operation Research (1), 117–129.
Hieleber, R., Kelly, T., Ketterman, C., 1998. Best Practices. Simon & Schuster.
Johnson, S., 1954. Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly (1), 61–68.
Porter, M., 1986. Competitive Strategy. Free Press.
Pree, W., 1994. Design Patterns for Object-Oriented Software Development. ACM Press Books - Addison-Wesley.
Thangiah, S., Potvin, J., Sun, T., 1996. Heuristic approaches to vehicle routing with backhauls and time windows. Internation Journal of Computers and Operations Researc 23 (11), 1043–1057.
www.bwpccoe.org, 2003.
www.ebusinessforum.com, 2003.
www.obarros.cl, 2003.
www.siebel.com/bestpractices, 2003.

Biographies

Oscar Barros has a Ph.D. in Industrial Engineering (Operations Research) from the University of Wisconsin. He is Full Professor an Head of the Master in Business Engineering at the Industrial Engineering Department of the University of Chile. He has published ten books and more than twenty papers in international journals, including two in the Journal of Systems and Software. He has developed the idea of Business Process Patterns, which are models that facilitate the redesign of business process in practice and that have been extensively used in spanish speaking countries.

Samuel Varas has a Ph.D. in Information technology from the George Mason University. He is Assistant Professor at Industrial Engineering Department of the University of Chile. He has developed research work on optimization models applied to information technology, such as workflow programming, materialized views and information updating policy.