BUSINESS PROCESS PATTERNS AND FRAMEWORKS: REUSING KNOWLEDGE IN PROCESS INNOVATION

Oscar Barros
Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Industrial
obarros@dii.uchile.cl – www.obarros.cl

1. Introduction

A common theme in the more recent Business Process (BP) literature is the search for approaches that allow formalizing domain knowledge into structures, patterns or frameworks, which can be reused to facilitate process redesign and support systems development [2, 8, 9, 12, 20, 26, 27]. The objective of these structures is to simplify and accelerate process innovation. These types of structures are also known as Enterprise Architectures (EA) and can be developed for different organizational levels [14]. Here we will consider the levels of BP and IT systems.

For the BP structure, several proposals for different domains have been developed. The better known of these are: SCOR, which are reference models for supply chain management processes that are structured as a tree of typical sub processes and component activities, together with business practices, performance attributes and metrics for them [26]; eTOM, a BP architecture for telecommunication companies, which is similar to SCOR, but oriented to the telecommunication processes domain [27]; and MIT's Process Handbook project, a collection of business practices that are structured along BP of different domains [16, 17].

For the IT systems structure, more relevant proposals are: Fowler's patterns, which are published objects structures in domains such as accounting, billing and payroll [12]; archetype patterns, which are configurable UML models for a given business context [2]; and Model Driven Architecture (MDA), which is an approach, not a structure, to automatically built software by a series of transformations, starting with a business model (Computation Independent Model: CIM) [19], for which an OMG Task Force is developing a BP metamodel [25]. The fist two of these IT structures,

despite their technical orientation, consider some aspects of BP structure, since they are related to a particular BP or business context and contain at least some business logic.

In what follows we present our proposal for the two levels of structure defined above. For the BP structure are propose our Business Process Patterns (BPP), which are generalized process models, including activities and flows connecting them, of how a business in a given domain should be run, according to best practices known [4]. The word pattern is used here for structures that are very different than software patterns [2, 12]. As we will explain in more detail below, BPP can be built for very general domains, such as the value chain of any business. These general BPP can them be specialized for specific domains; e.g., the process pattern associated to the value chain in a hospital. This generates a tree of BPP from more general at the top to more specific at the bottom.

In comparison with the BP structures mentioned above, our BPP are more formal and detailed, since they explicitly consider not only the activities of a BP but their interrelationships by means of well-defined information and physical flows; they also include explicit business logic for all the activities. Of course, this logic will be more detailed when the domain is more reduced. The BPP we propose have been developed during a period of more than ten years, based on several hundreds of real cases of BP redesign, and tested with over one hundred of new cases where the patterns have been used to generate new BP designs [4, 23]. We present the BPP and report on their use in Section 2. We also show how to specify the business logic for such patterns in Section 3.

For the IT system structure, we propose Business Objects Frameworks (BOF), derived from corresponding BPP. Here we adopt the definition of a framework used in software development as "a body of code that implements the aspects that are common across an entire domain, and exposes as extension points those elements of the domain that vary between one application and another" [11]. In developing a framework of this type, associated to a BPP, a UML model that defines the Business Objects (BO) involved is first derived.

Our frameworks differ from traditional ones in that they are mostly oriented to code business logic, in particular complex business decision logic, using analytics, according to current best practices, taken from OR/MS, Business Intelligence and Statistics. This also makes them very different from the IT structures above that only include simple business information processing type of logic that, in

many cases, duplicate logic included in packages such as ERP. This orientation has allowed that, in practical applications of BOF, their advanced flexible logic has complemented or been more effective than the one contained in competitive ERP and other types of packages [23]. We will show how to derive BOF from BPP and their use in Section 4.

We remark that our BOF are formally integrated with BPP, while the other proposals for the two level structures above have been developed independently. To remedy this lack of integration such proposals, a was mentioned above, have extended IT systems structure to include business aspects, but this only allows to consider a few aspects of the more important issues of process relationships. We will illustrate this point when we present our BOF.

Finally, in Section 5 we present a real application of our BPP and BOF and, in Section 6, conclusions.

2. Business Process Patterns

Business Process Patterns are models of how a business in a given domain should be run, according to the best practices known [4]. 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 the best practices literature, reported experiences and direct observation of firms [15, 21, 22, 24, 26]. Our patterns have benefited from the knowledge derived by hundred of cases in which processes of many different companies have been modeled, analyzed and redesigned [23] and previous experience with the formal modeling of Information Systems [3].

We have found that beyond the best practices for a given domain, usually expressed in the form of specific business logic, BPP share a common structure of activities and flows. Thus, products or services provision processes – such as manufactured goods, health, justice and financial services, etc – share such a common structure. A first level of detail of such a process structure of BPP, which is a formalization of what is commonly known as the value chain [18] that we call Macro1, is shown in Figure 1. It includes the sub-processes of *Customer relationship management, Supplier relationship management, Production and delivery management, Production and delivery*

of products or services and *State status.* It also shows the necessary relationships among such sub-processes by means of information and physical flows, such as supplies and products [4]. Now, following the IDEF0 modeling scheme, we can detail such a process by partitioning each of its sub-processes, as it is shown in Figure 2(a) for *Customer relationship management.* Our BPP do not depend on IDEF0, so other modeling approaches can be used, such as the one proposed in [13].

If we want to give further detail of the BPP, 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, we synthesize our experience of many real cases in the following domain definition for the activity *Marketing and customer analysis* of Figure 2(a). We assume situations where businesses sell physical products or services to a large number of customers; specifically we include cases such as retail using any channel and direct sales by manufacturing or service firms.

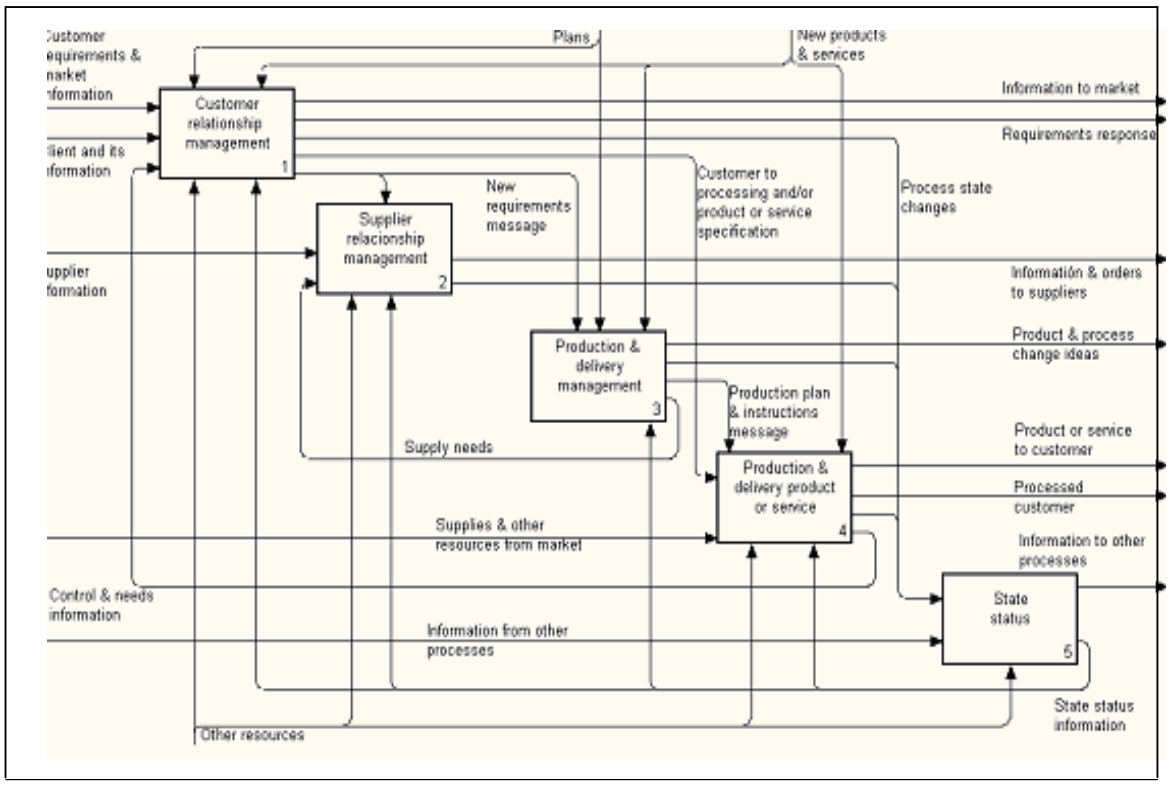
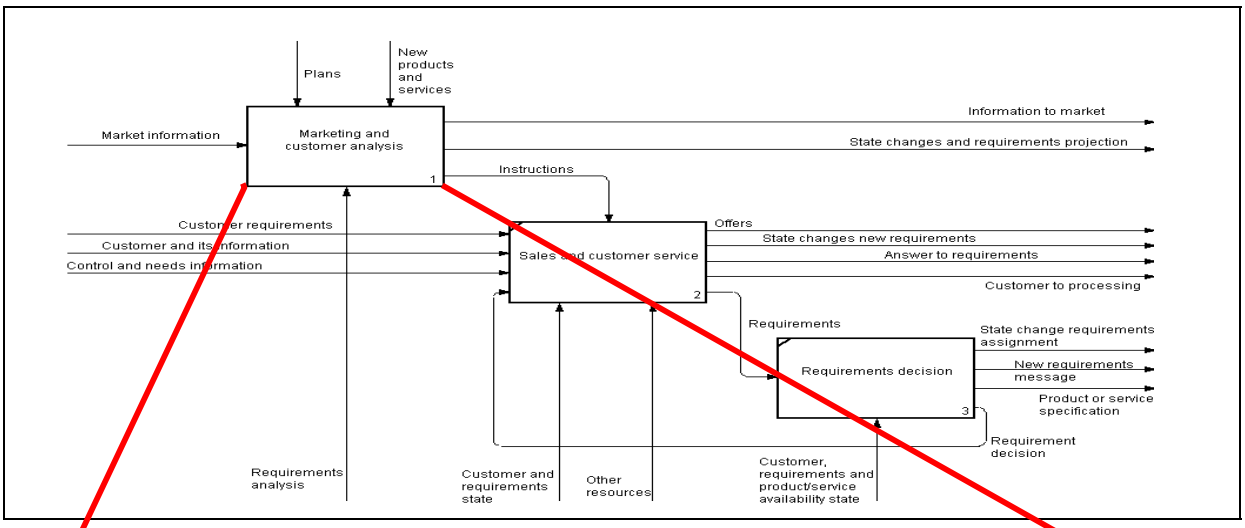
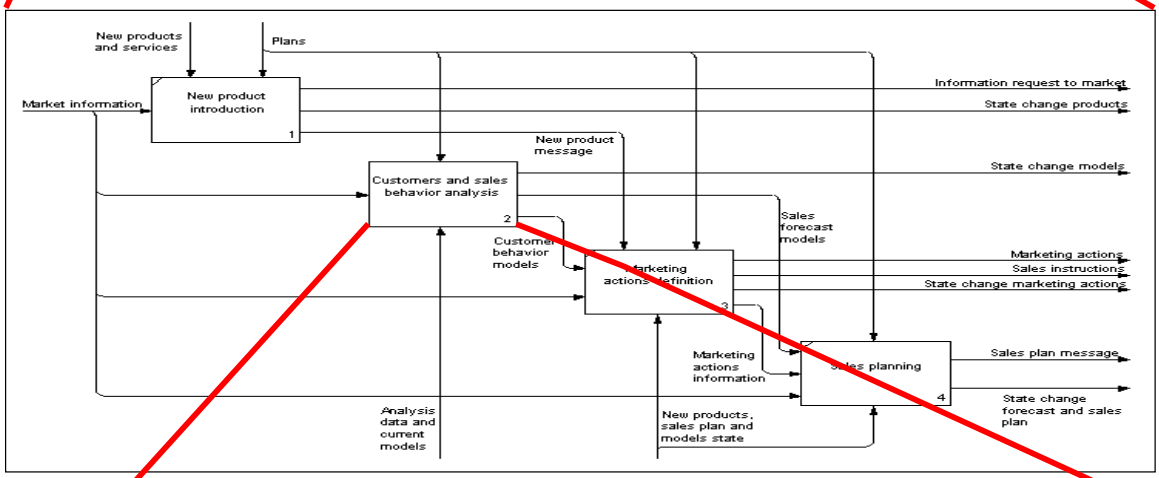


Figure 1. Business Process Pattern for products or services provision (Macro1)

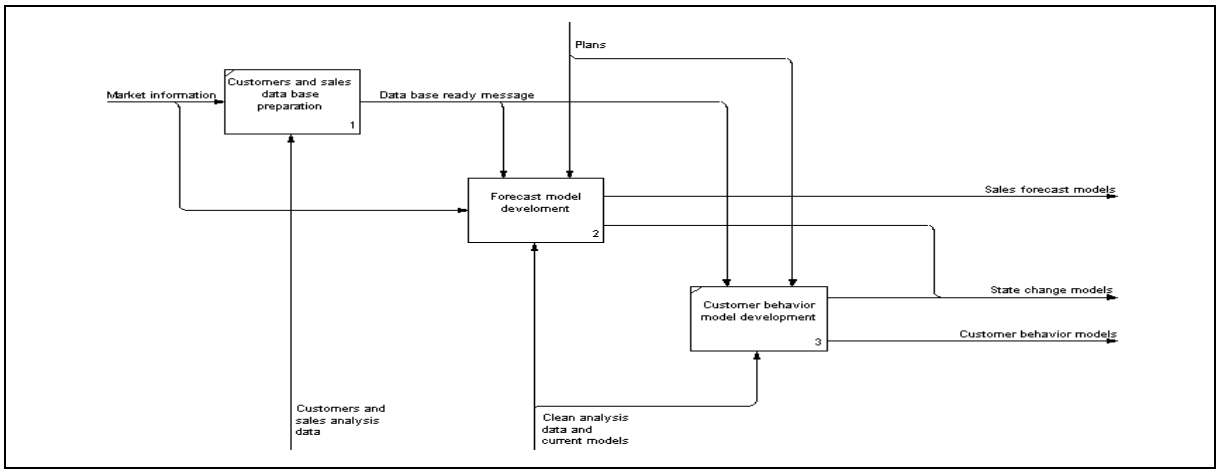
Under previous assumptions, we decompose the said activity in Figure 2(b), where we will concentrate on *Customer and sales behavior analysis.* For such an activity in this specific domain, which is further decomposed in Figure 2(c), we can be very precise about a best practice business logic. Business logic, which guides the action in an activity, determines the exact information flows that are required and that are produced, including the updating of the process changes in *State Status* of Figure 1 and the state information this generates, shown below each activity, that is necessary for its performance. We will show, in the next section, how to specify such logic.



a) Detail of *Customer relationship management*



b) Detail of *Marketing and customer analysis*



c) Detail of *Customer and sales behavior analysis*

Figure 2. Decomposition of C*ustomer relationship management*

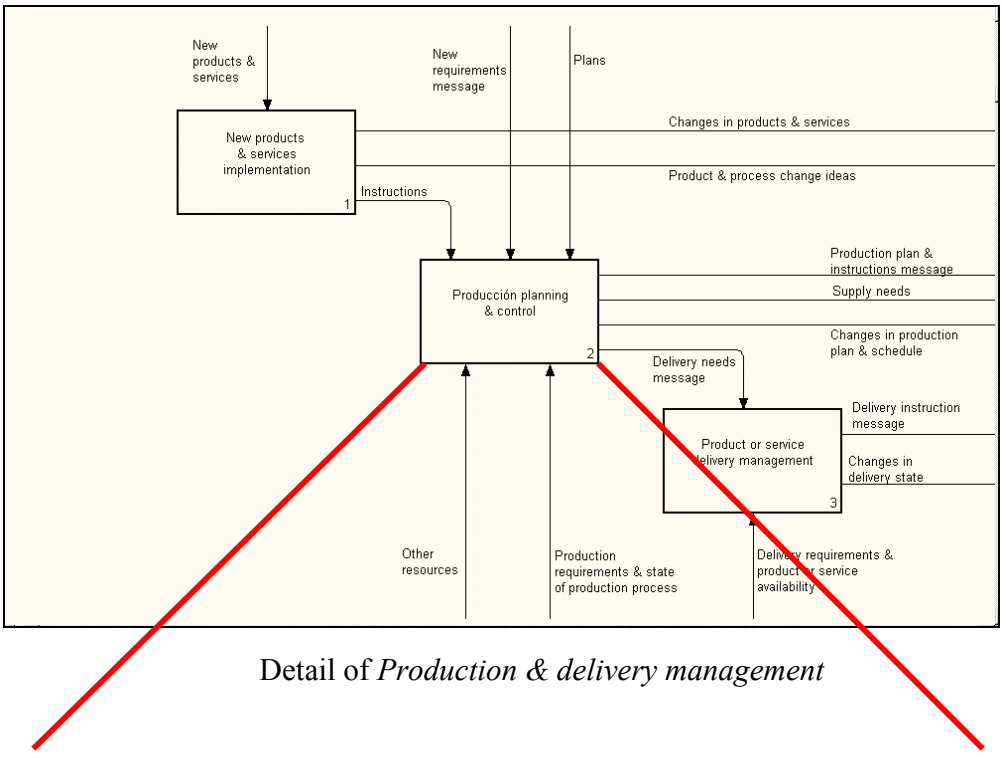
Decomposition can be performed for any of the activities in Figure 1. To further illustrate such task, we decompose *Production and delivery management,* considering the same domain previously defined. This is shown in Figure 3, where we have concentrated on the activity of *Scheduling,* for which we will provide generalized business logic in the next section. Details for the other activities of this BPP are given in [4].

We have illustrated the concept of BPP with just one process pattern for a very general domain. Other BPP have been developed for particular cases, specializations, of the general domain, such as situations in which there is not production and sale of products and services is made from stock bought from others, e.g. distribution companies or telephone companies, manufacturing that sells their own products, and hospital or financial services [4, 23]. Also, BPP have been developed for other types of processes, complementary to the value chain: Management of resources –human, financial and fixed assets- (Macro4), Development of new products (Macro3) and Business planning (Macro3) [4]. As a matter of fact we have found that with these types of patterns it is possible to model any process of any business. These types of BPP and their specializations conform, as advanced above, a tree of BPP from which we can select the closest to a situation one wants to redesign. A partial version of such tree is given in Figure 4, where the root *General Architecture*

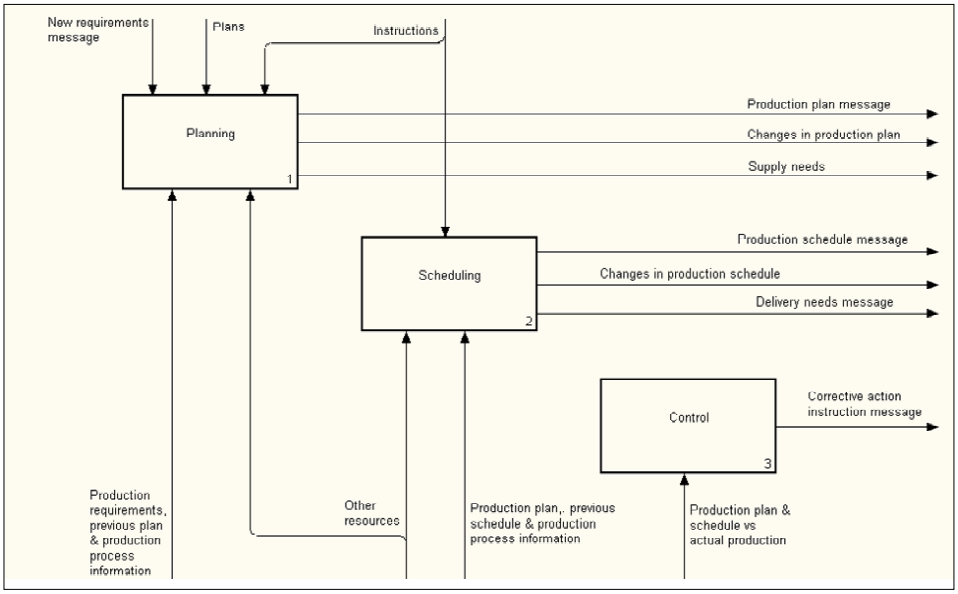
refers to the fact that all the BPP are built starting with a generic structure, well founded from a theoretical and empirical point of view, which we do not detail here [4].

3. Business logic specification

Our aim is to give generalized business logic for activities in a specific domain. Consider the activity of *Forecast model development* of Figure 1(c). Under previous domain assumptions, a forecast based on sales history is possible. We also assume that, previously, a datamart with a relevant and clean history has been set up in the activity *Customer and sales data base preparation*. Then, an analyst that performs former activity will have a system support with a business logic that allows him to do the following:



Detail of *Production & delivery management*



Detail of *Production planning a control*

Figure 3. Decomposition of *Production and delivery management*

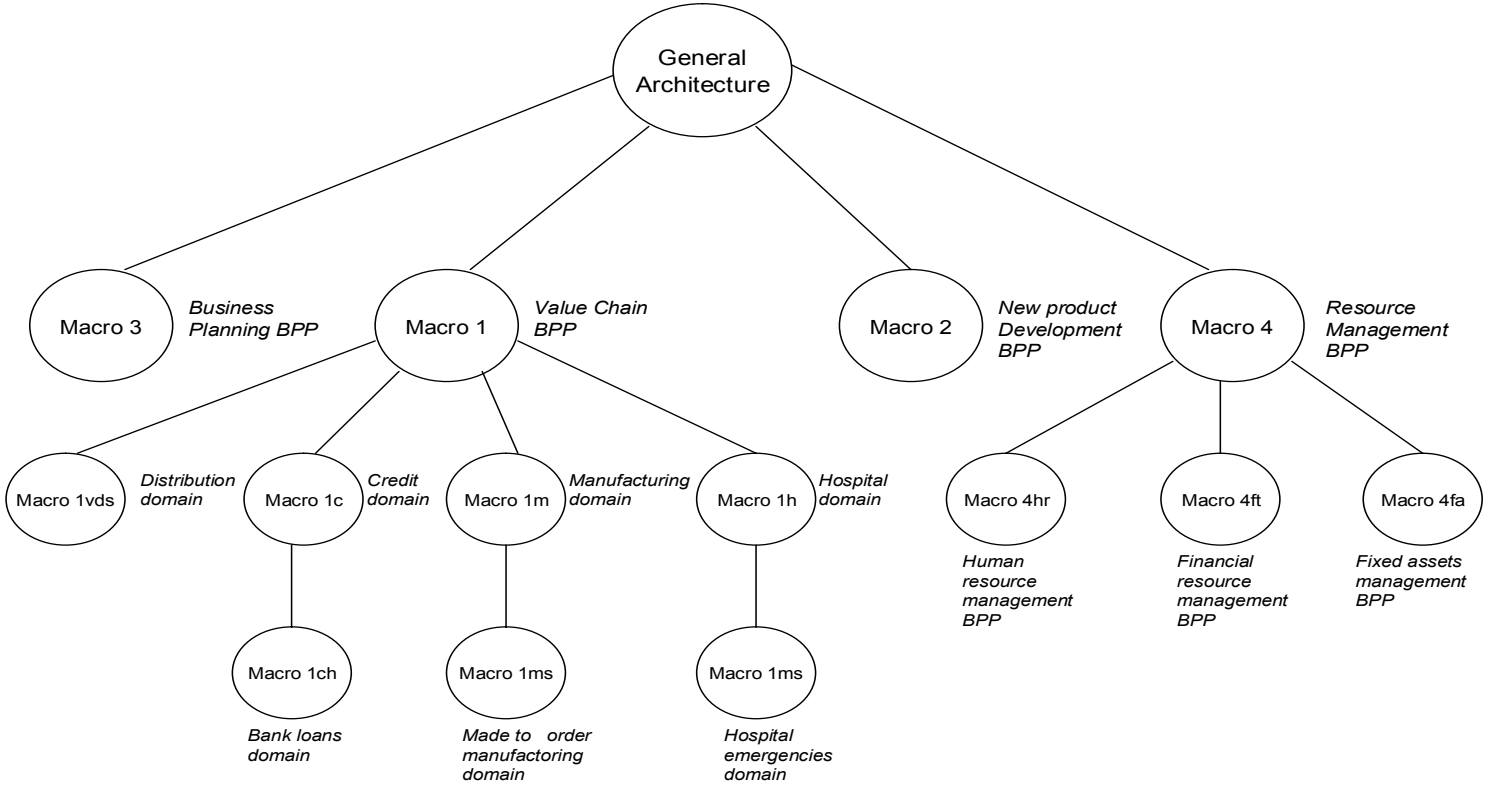


Figure 4. Tree of Business Process Patterns

i) For all current forecast models for sales items, made available through *Clean analysis data and current models,* to calculate forecast error by comparing a selected history of forecasts and actual sales.

ii) For selected sales items and forecast models –e.g. exponential smoothing, Box-Jenkins, neural networks [1]-, to fit historical sales data to models, proposing parameters and providing estimated forecast error.

iii) To update models selected by analyst for routine use in forecasting in *Sales planning* of Figure 1(b).

This a simplification of real cases we have performed where the logic can be more complex, involving model identification and training with more analyst responsibility than the one outlined.

In order to give a flavour of the business logic included in the system support above, we outline a portion of the logic corresponding to the fitting and estimation of a selected model to historical data. This logic, which is shown in Table 1, corresponds to a simplified version of the identification and estimation of a Box-Jenkins model. The logic is mostly of statistical calculations, with some analyst intervention for the more qualitative aspects. Hence it leaves little room for introducing causal business factors and decisions, such as economic environment, promotions and pricing policies. However, other methods, such as neural networks, allow to consider such factors, in which case business logic would explicitly consider the interaction between commercial decisions and forecast [1]. These possibilities are considered in the full version of our BPP.

The key for developing generalized business logic –valid for all particular cases in a domain- is to structure the decisions in the domain in cases, according to certain variables or parameters that determine the relevant algorithm or heuristic for each case. If we apply this idea to the forecasting case, considering, in a simplified way, the variables of sales stability and style of marketing as determining the right type of forecasting model, we show in Table 2 the models that apply for combinations of such variables (cases). This determines the logic for forecasting sales in each case. Then a logic as in Table 1 can be specified for each case. The same idea as above can be applied to the *Scheduling* activity. We consider the general situation where a list of waiting tasks, which can be products to be manufactured or services

//Business logic outline for Box-Jenkins model identification and estimation

//Previously, several tests for determining the suitability of Box- Jenkins against other methods have been performed; e.g. presence of tendency, seasonality and white noise and consideration of the number of observations

//Model identification

Calculate autocorrelations and partial autocorrelation functions
Test for determining if series is stationary
//Autocorrelation decay is evaluated
***If** series is not stationary*
    ***Do** the series difference **until** it is stationary*
    //Times series is differenced corresponds to parameter d
Endif

//Series is stationary
Establish behavior of autocorrelation and partial autocorrelation functions: decay, oscillation, truncation, large particular values, etc.
Identify type of model: MA(q), AR(p), ARMA(p,q) or ARIMA(p,q,d)
//This is based on functions behavior
Show the analyst autocorrelation and partial autocorrelations graphics and the proposed model
Accept the analyst approval of proposed model or own values for parameters p and q

//Models estimation and testing

Estimate constants for model
Perform model goodness test (Box-Pierce)
Test model by using new historical data to forecast and calculate forecast error
Show analyst estimated model and tests
***If** the analyst accepts the model **or** decides that no model can be fitted*
    Update the model
***Else** analyst establishes new analyses to be made*
//This may mean going back to the model identification or selecting a model different from Box-Jenkins.

Table 1. Business logic for the Box-Jenkins identification and estimation

to be performed, is to be assigned to servers – machines, people, trucks, etc – for their completion. In this case structure depends on the variables in Table 3, where the relevant heuristics are also shown.

		DEMAND	
		STABLE	NOT STABLE
MARKETING	*PASSIVE*	Exponential smoothing	Box- Jenkins
	ACTIVE	Exponential smoothing and Box-Jenkins	Box-Jenkins and neural networks

Table 2. Appropriate forecasting models for various cases

In order to show the type of logic for *Scheduling*, we give, in Figure 5, the general logic for all the cases of Table 3 and one example of specific logic for the case with set up times, no completion time and one server. Though they may look mathematically complex, they are a formalization of simple heuristics: for the general case, to first assign tasks to servers using an appropriate criteria and then sequence them in each of the servers in the best possible order; and for the particular case, choose a sequence by selecting always the task with the smallest remaining set up plus processing time (greedy heuristic). The algorithms for the rest of the cases and their justification are given in [7].

		NUMBER OF SERVERS		
		1	**2**	**>2**
NO SET UP TIME BETWEEN TASKS	NO COMPLETION DATES REQUIRED	Greedy heuristic (shortest task first)	(Series servers) Greedy heuristic	(Series servers) Grouping of servers plus greedy heuristic
	COMPLETION DATES REQUIRED	Greedy heuristic with time windows	(Series servers) Grouping of servers plus greedy heuristic with time windows	
SIGNIFICATIVE SET UP TIME	NO COMPLETION DATES REQUIRED	Greedy heuristic with set up times (smallest set up plus processing time task first)	(Parallel servers) Assignment of tasks to servers plus greedy heuristic with setup time for each server	
	COMPLETION DATES REQUIRED	Greedy heuristics with set up times and time windows	(Parallel servers) Assignment of tasks to servers plus greedy heuristic with setup times and time windows for each server	

Set up Time: Time needed between execution of any two tasks; e.g. change of tooling in manufacturing, travel time between locations in telephone repair and cleaning of surgical facilities between operations.

Server: Facilities that execute tasks; e.g. machines, repairmen and surgical facilities

Table 3. Appropriate scheduling heuristics for various cases

Heuristic Schedule (n°, m)

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

$n = n^{\circ}$

SelectSet (m, n, Φ)

GetTask (Φ, π)

If $(m > 1)$ *then*

ImproveSchedule (π)

GetTask2 (Φ, π)

$q = \arg\min_i \{\min_j \{S_{ijk} + P_{jk} \mid i, j \in \Phi \}\}$

$\pi = insert(q, \emptyset, 0)$

$\Phi = \Phi \setminus \{q\}$

while$(\Phi \neq \emptyset)$

{

$q = \arg\min_j \{S_{qjk} + P_{jk} \mid j \in \Phi \}$

$\pi = insert\ (q, \pi, |\pi|)$

$\Phi = \Phi \setminus \{q\}$

}

where

- Routine *SelectSet()* selects a subset of tasks for every facility, where the result is given in the *m*tuple $(\Phi_1, \Phi_2, ..., \Phi_m)$ of subsets of Φ .
- Routine *GetTask*() selects a sequence π_j for the set of tasks Φ_j of each facility $j\ 1 \le j \le m$; its implementation will depend on the specific case.
- Routine *ImproveSchedule*() improves the current schedule π for all facilities.
- $insert(j, \pi, i)$ returs a new schedule with element j inserted in schedule π just after place i
- m is the number of different types of facilities or machines.
- n is the number of distinct tasks types.
- P_{ik} is the processing time of task i at facility k
- S_{ijk} is the set up time if task type i is processed immediately before task type j at facility k .

Figure 5. Example of business logic for *Scheduling*

4. From Business Process Patterns to Frameworks

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

The mapping from BPP and business logic to a BOF is as follows:

i) The structure of the BPP system support and the business logic of the domain give a first cut definition of the BO classes that encapsulate the algorithms or heuristics that solve the problem for different cases in the domain. .

ii) Structure of the BO can then be modeled 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 data included in the business logic.

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

We follow the steps above for the activity *Forecast model development* in Figure 2c.

The structure of the system support in 3i,ii,iii and the business logic in Tables 1 and 2 lead us directly in to the BO structure of Figure 6, where we also show the data classes and the operations for each class. We use common OO conventions for frameworks and adopt some of the ideas in [10] to organize classes. The structure is not complete, since it should be integrated with all the components that support the *Sales planning* of Figure 2, where forecast models are actually run to produce forecasts that are needed for generating sales plans, which we have avoided to simplify presentation.

The BO of the framework can be structured according to the type of cases in the domain, defined in Table 2. Then the analysis can be tailored and made more specific for each particular case. In Figure 7 we show in a simplified way how this is done in our forecasting framework. The key is to structure the *Model analyzer* BO in component cases, which provide

different solutions according to the characteristics of the problem. In Figure 7 such structure is organized according to the variables of stability and the type of marketing previously mentioned. Then, for each case in the structure, an appropriate logic is provided, based on the experience obtained from the results generated with the use of the most important analysis methods [1, 5, 6, 23]. Hence, when using the framework in a particular case, only its relevant parts can be selected.

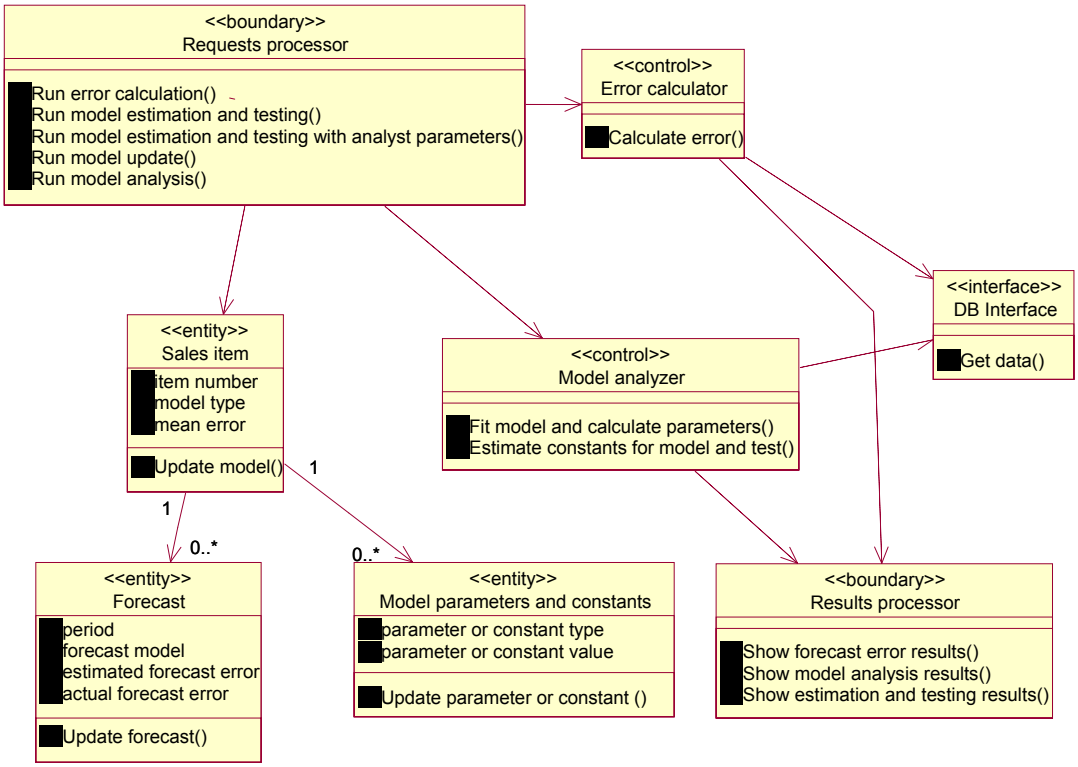


Figure 6. Framework for forecasting model development

In the same way a framework for *Scheduling* can be developed, which is shown in Figure 9. This framework is based on the idea that all the problems in the domain share a common structure (BOF) with several different cases, defined in terms of number of processors and configurations (series, parallel and network), which can be selectively used according to the characteristics of the problem al hand.

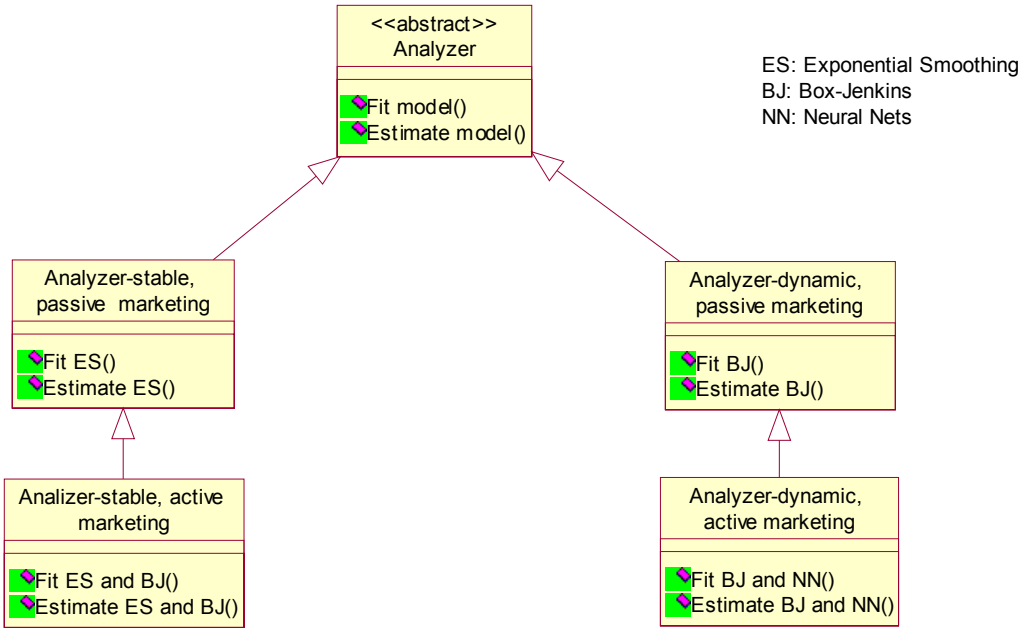


Figure 8. Structure for *Model Analyzer*

An important characteristic of these frameworks is the possibility of having incrementally more complex logic for the cases in a framework, defined as outlined above. Thus, for example, for the *Scheduling* activity, the structure of Figure 9, which is an specialization one, shows that there are methods – *HeuristicSchedule, SelectSet, GetTask*, and *ImproveSchedule* - which are used by all specialization classes. Then the three branches starting at the class *Scheduler*, define three alternative cases, one with lead time, another with setup time, and one with both. Each branch has a method that is an specialization of *GetTask* – *GetTask1*, *GetTask2* and *GetTask3* - which is inherited by cases immediately below in such branch.

The same is true for other branches below this one. All these methods are specified by means of a business logic of the type given in Figure 5. Now each branch has classes increasingly more complex; e.g., the one at the right of Figure 9, corresponding to the case with led time but no set up time, has *Scheduler1* corresponding to the case with one facility, *Scheduler11,* with two facilities and *Scheduler111,* with three facilities in parallel. Each class inherits methods from classes above in the branch and uses them in their own methods. This means that more complex cases are built based on methods of simpler ones. Thus when using the

framework to develop an application for a particular case, just one branch and some of the classes of the branch will be selected: just the ones appropriate for the particular case at hand. The framework can be extended by adding more specialization classes at the last level selected of the branch. We will illustrate this idea with a real case in the next section.

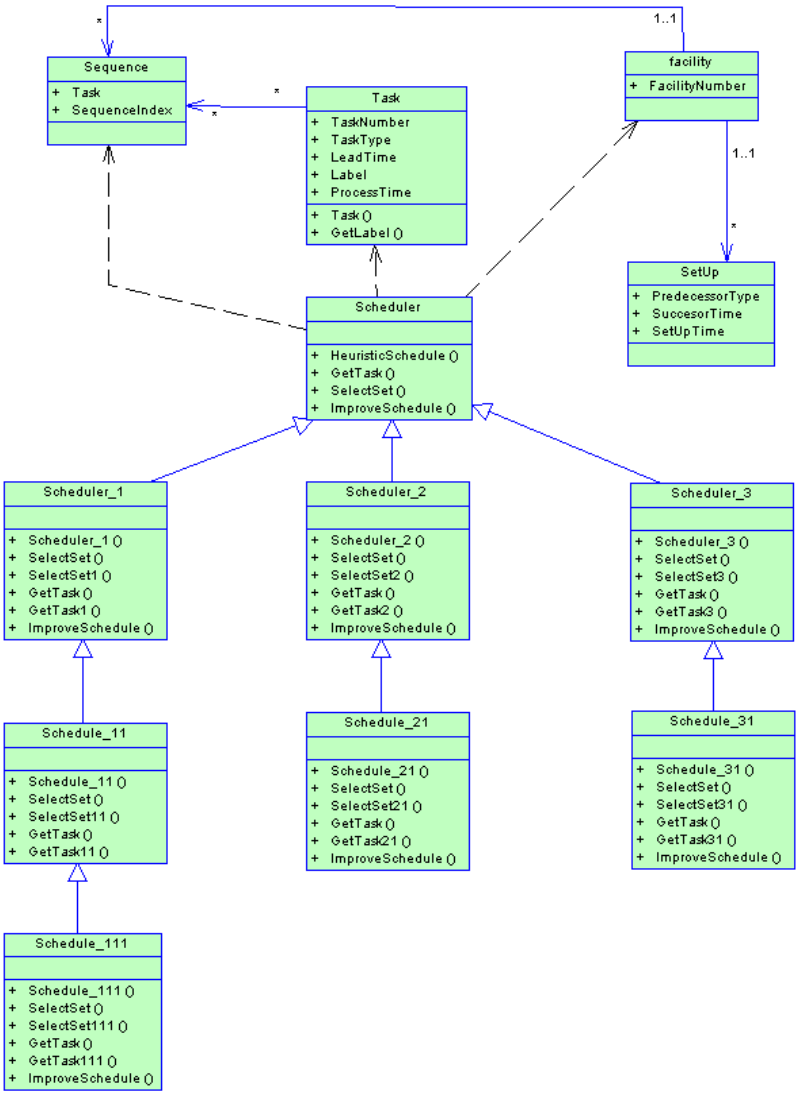


Figure 9 Framework for *Scheduling*

Hence, in the application fo a BOF to a particular situation, the user of the framework may select the minimum level of complexity that solves his problem Thus, for example, some developers may select, in Figure 9, just the class *Scheduler 1,* because they have a case with lead time, no setup time and just one facility.

The implementation of this feature, which allows the selection and use of incrementally more complex solutions for a case, is based on OO inheritance. We have coded our framework based on this feature and determined that is very simple to select and combine the options that they offer and to specialize them to particular applications. This idea of selection from a given framework is related to the notion of configuration of [2], where formal models have been developed to define such general structures and automatically generate valid particular cases. Our approach depends more on analyst specialization.

The frameworks we have used as examples have been presented as stand alone, which is not realistic. In some cases they would be integrated with other frameworks for others activities in a process, as outlined in Section 2; in others cases, they can be used without integration, but they should be, at least, be connected with other applications though business DB.

We have developed working frameworks for several activities of the process in Figure 1, which contain the 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; the frameworks we have presented in a very simplified way in this paper; inventory management that includes JIT and Reorder Point cases, with probability considerations for demand and lead times; and production planning with math mathematical models.

5. Using Frameworks for Application Development

In using a framework of the type derived in Section 4 for developing an application to support a process in a real-life case within the frameworks domain, the following procedure is used [7]:

i) Select relevant substructure of the framework applicable to the case.
ii) Specialize substructure to the characteristics of the case, adding data and logic as needed.
iii) 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 process of attending telephone repair calls and the day to day scheduling of such calls (tasks) for the largest telephone company in Santiago, Chile. In attending 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 9. So the relevant classes for such node in the structure are selected, which are shown in Figure 10.

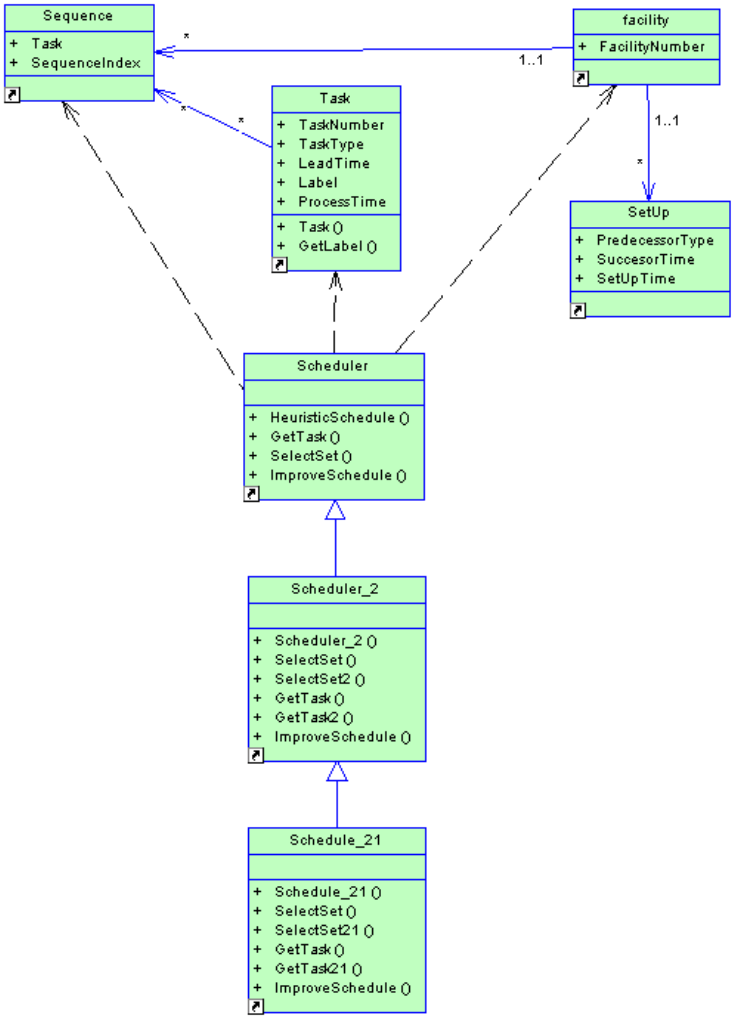


Fig. 10. Relevant classes for repairmen scheduling

Specialization of Figure 10, 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 these classes inherit from the selected structure as shown in Figure 11.

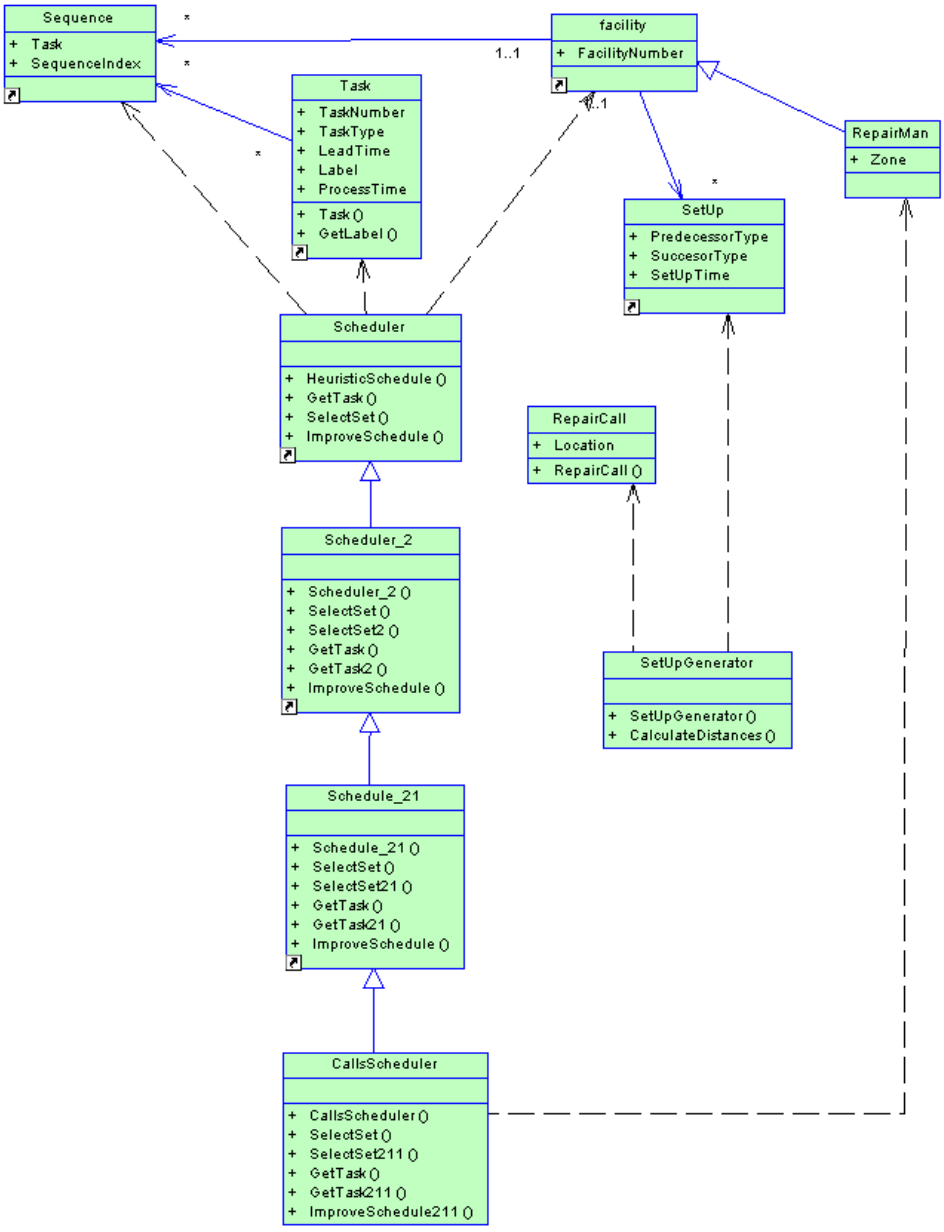


Fig. 11. Specialized framework for repairmen scheduling

Then the business logic for *SelectSet211* is obtained from class *SelectSet21*, and the sets are sequenced with *GetTask21(Φ,π)*, where all sequences start at the dummy node defined above. We have added a method *ImproveSchedule211* to balance the work load among repairmen by resigning calls from overload zones to the nearest one with available time.

The coding and actual use of the framework in solving the repairmen scheduling problem has confirmed the benefits we expected in its practical application. The coding effort of the framework was about 2000 lines of Java code.

Now in terms of its use, it is clear that the framework offers a pre built solution that allows the developer to select, among several possibilities, the functionality that solves his problem and specialize it, if it does not have a complete fit with the problem. So it requires much less work than an ad hoc solution and does not require the ability to develop complex algorithms. On the other hand, a packaged solution has the same advantages of the framework, but it is not customizable by mean of specialization; so if it does not fit the problem, it use is not feasible. In our case we did not find a package that was able to balance load among zones. For specializing the code of the framework in Figure 9, we wrote about 500 lines of code in Java.

6. Conclusions and Future Work.

We have shown in detail the workings of our approach for developing BOF based on BPP. This included the presentation of realistic example frameworks. In particular we have presented a working procedure that can incorporate domain knowledge in providing generalized solutions that are able to be reused and specialized for integrated business process and application design in a given application domain. This also solves in a generalized and rigorous, business design based way, the requirements problem in system development for situations where complex business logic is involved.

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 based on frameworks, with savings in developing costs, and also the option to easily customize and optimize a solution according to the specific characteristics of a given case.

Our research is continuing in several directions. Firstly, we are applying the full version of the example *Scheduling* framework of this paper to the actual assignment and routing of installation request in a cable TV/Internet provider. Secondly, the forecasting framework is being extended to include cases not included in it; in particular for situations with complex variables such as fashion. Thirdly, frameworks for other activities in the value chain defined in this paper are being perfected, such as supply chain management, production and operations planning, and logistics. Also we are working on the integration of these

frameworks; in particular we have developed an integrated BPP and framework, which covers the whole value chain, with practices adapted to small and medium sized companies [23]. Finally, we are perfecting the way to deliver these frameworks for practical use, by using technologies such as EJB and web services.

REFERENCES

1. L. Aburto, R. Weber, Demand Forecast in a Supermarket using a Hybrid Intelligent System, in: A. Abraham et al. (Eds.), *Design and Application of Hybrid Intelligent Systems*, IOS Press, Amsterdam, Berlin, 2003, pp. 1076-1083.

2. J. Arlow, I. Neustadt Enterprise Patterns and MDA, Addison Wesley, 2003

3. O. Barros, Modeling and Evaluation of Alternatives in Information Systems, *Information Systems* 16 (5) (1991) 537-558.

4. O. Barros, *Rediseño de Procesos de Negocios Mediante el Uso de Patrones,* Dolmen, 2000.

5. O. Barros, Busines Information System Design Based on Process Patterns and Frameworks. *BPTrends* (www.bptrends.com), Sept. (2004)

6. O. Barros, A Novel Approach to Business and Information Systems Design, *Journal of Computer and Information System,* (Forthcoming, Spring 2005)

7. O. Barros, S. Varas, Frameworks derived from Business Process Patterns. Technical Report 56, 2004, Industrial Engineering Department, University of Chile (Available at www.obarros.cl)

8. K. Bohrer, V. Johnson, A. Nilsson, R. Rubin, Business Process Components for Distributed Object Applications. *Communications of the ACM* 41 (6)(1998) 43-49.

9. M. Cline, M. Girou, Enduring Business Themes. *Communications of the ACM* 43 (5)(2000) 101-106.

10. J. Conallen, Modeling Web Application Architectures with UML. *Communications of the ACM* 42 (10)(1999) 63-77.

11. S. Cook, Domain–Specific modeling and Model Driven Architecture, MDA Journal, *BPTrends* www.bptrends.com, Jan. (2004)

12. M. Fowler, *Analysis Patterns: Reusable Objects Models*. Addison-Wesley, 1996.

13. N.P. Dalai, M. Kamath, W.J. Kolarik, E. Sivaraman, Torward and Integrated Framework for Modeling Enterprise Processes, *Comunications of the ACM* 47 (3) (2004), 83-87.

14. P. Harmon, Enterprise Architectures, *BPTrends* (www.bptrends.com), Jan. (2004).

15. R. Hieleber, T. B. Kelly, Ch. Ketterman, *Best Practices*, Simon & Schuster, 1998.

16. http://ccs.mit.edu/ph/

17. T. W. Malone, K. Crowston, G. A. Herman, *Organizing Business Knowledge*: *The MIT Process Handbook*, MIT Press, 2003.

18. M. Porter, *Competitive Strategy*, Free Press, 1986.

19. H. Smith, P. Fingar, *Business Process Management: The Third Wave,* Megham-Kiffer Press. 2003.

20. D. F. Sousa, A. C. Nills, *Objects Components and Frameworks with UML*. Addison-Wesley, 1999.

21. www.bwpccoe.org

22. www.ebusinessforum.com

23. www.obarros.cl.

24. www.siebel.com/bestpractices

25. www.omg.org/mda

26. www.scor.com

27. www.telemanagementforum.com