The Nutshell Pattern
A holistic Approach to Object-Orientation

Helfried Pirker, Heinz Pozewaunig, Roland T. Mittermeir
Universität Klagenfurt
Institut für Informatik-Systeme
Universitätsstraße 65-67, 9020 Klagenfurt, Austria
{helfried, hepo, mittermeir}@ifi.uni-klu.ac.at

Abstract

This paper presents the *Parcel Dispatching System* as an effective classroom example for teaching (a) how to find the proper objects, (b) the difference between analysis and design objects, (c) advanced concepts like packaging and collaboration, and, finally, to provide the students (d) with a holistic view of object-orientation. The example is introduced as an instance of the Holistic Nutshell Pattern representing a multi-dimensional approach by interrelating object-oriented concepts.

1 Introduction

Teaching introductory courses on object-orientation (and teaching in general) is in most cases a bottom up process. Basic concepts are incrementally introduced one after the other but in relative isolation to each other. The overall view of the concepts and the relationships between them are introduced later, when course time gets scarce. Hence, students are only faced with a very narrow view of system development. After learning different methods for modeling different object-oriented (OO) views of a system (static, functional, dynamic, and user-interface view), the OO-beginner is in most cases not in the position to grasp the system in its entirety. In general, students at that learning stage are not able to *"get the big picture"* and have problems in understanding that all developed models are just different abstractions of the same system seen from different vantage points. Lack of time forces instructors in many classroom situations to present concepts but to talk about their integration only in a general manner without going into any details. This leaves students alone in struggling with the complexity of integration.

The *Lay of the Land Pattern* [5] focuses on that issue. The goal of this pedagogical pattern is to introduce a large, complex example early in the course, even if the students only have basic knowledge. They should "get the big picture" of the concepts and how they are related.

In many object-oriented analysis- and design methods there is also *no clear and explicit distinction between the analysis and the design phases.* The transition from one phase to another is often blurred and there exists no simple process model which describes the diverse working pieces step by step. Hence, examples used in introductory courses should not only be instances of the Lay of the Land Pattern, but should additionally include some development process focus. These issues are subsumed within the *Holistic Nutshell Pattern* described in section 3.

This paper is based on a three-dimensional perception of classroom examples focusing on the development process, the system models involved and the prerequisite knowledge of the students (cf. section 2). As an example, the *Parcel Dispatching System* is presented in section 4. We used this example in introductory courses on OO Systems Analysis and Design. Some issues, why we think that this example is an effective classroom example, are discussed in section 5. Finally, some conclusions are drawn.

2 Dimensions to classify Classroom Examples

When designing classroom examples, one should always have the context of the example in mind. By context we mean *application related issues* and *situational issues how to apply the example in a course.* The application context is covered by this section, the situational one is described by the nutshell pattern in section 3.

By example related issues we refer to the scope of the example, indicating which models are involved, which development stages are covered and which knowledge is required of the students. Hence our framework uses the following three dimensions to classify examples:

- a *development process* dimension,
- a *system-model* dimension,
- and a *prerequisite knowledge* dimension.

The dimensions sketched above span a three-dimensional space (cf. figure 1) in which classroom examples should be placed. These three dimensions are discussed by the following sections.

Additionally, one should also mention that classroom examples are to be *realistic.* This sounds trivial, but it isn't. This requirement is not only a motivational prerequisite but also a prerequisite to allow students not only to aim for syntactical correctness but also to validate their models against their domain understanding. Hence, the examples need to be real but not too complex. They should allow to build correct initial abstractions that can be refined further without relying too much on the students' domain knowledge.

2.1 Development process dimension

An example should always be designed with some development process focus [1] in mind. The aim is not to teach the students about processes or process models, but to show them that objects built at different development stages represent inherently different things [3].

Objects within an analysis model are abstractions of real world objects. The focus when modeling analysis objects is the real world or the problem domain. Hence analysis objects are part of the problem space and represent the (problem related) requirements for the system to be developed.

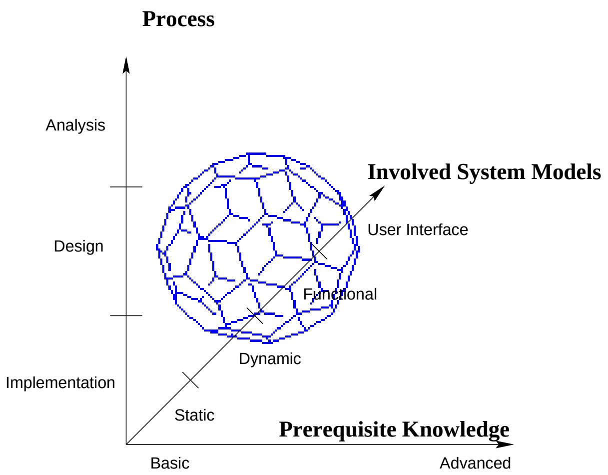

Figure 1: The multidimensional space covered by a classroom example

Objects within a design model are abstractions of the implementation to be built. The focus here is the implementation. Hence design objects are part of the solution space. Building a system design also means that the designer has to consider the desired systems architecture, interfaces to existing systems and possibly using design patterns or reusing existing objects. Hence design objects represent the technical requirements for the system to be built.

One could easily imagine, that a one-to-one mapping from analysis objects to design objects can not always be established, as they are built with a different focus in mind and represent different things. Hence, students should be aware that real world objects, analysis objects, design objects, and objects constituting the source code of the implemented system are identified and treated in a different way.

When considering development processes like the Unified Process [2], this awareness becomes more and more important because of the iterative nature of the process and because of the parallel deployment of the development phases (called *Core Workflows* in the Unified Process terminology).

Hence, when describing examples, the phases of the development process covered should be stated.

2.2 Prerequisite knowledge dimension

This dimension states to which extent knowledge of object orientation is required of the students. The range can be spanned from no knowledge at all to detailed, advanced knowledge of several concepts.

Introductory examples in object-orientation will require no previous OO knowledge at all but at least some programming skills in procedural languages and/or knowledge of concepts like abstract data types, information hiding and so on. For examples concluding an bottom up oriented course, the concepts already known by the students and required by the example should be stated. Hence the range of this dimension can be a very broad one.

2.3 System Model Dimension

When designing object-oriented systems according to methods like OMT [6], UML [7] or others, one has to consider the different models or views of the system and the dependencies among them. In most object oriented methods a static, dynamic and functional model exists.

For classroom examples it should be stated, which model(s) the example is focusing on. If two or more models are involved, it is also important to know, which relationships between the models are covered by the example, e.g. using a functional model to identify and assign methods to classes in the static model.

3 The Holistic Nutshell Pattern

The issues raised in section 1 are resumed within the Holistic Nutshell Pattern. It is described using the format of the pedagogical patterns project [4, 5].

NAME Holistic Nutshell (Holistic in the small)

PURPOSE During introductory courses, students should get the big picture and gather the understanding that all development models are different snapshots of the same system. They should also get awareness of the difference between the different development phases, e.g. the difference between analysis and design objects.

SOLUTION The aim of this pattern is to clarify the difference between various system views, how to develop these views towards an OO-implementation, and how to evolve the product with respect to the software life cycle. A further side effect is to stabilize the ability to model and design in an object-oriented manner, to evaluate design tradeoffs, and to introduce advanced concepts like packaging and collaboration.

CONTEXT/USAGE This pattern is intended as a learning unit after the introductory chapters to object-orientation. It should be applied in a context of at least 4 units. Students form groups from 3 to 5 persons. The first 2 units are dedicated to classical OO conceptual modeling under supervision of the instructor. Then the students try to develop their designs further and enrich them with the necessary technical details. The review and reflecting process are subjects of the last unit.

FORCES This educational pattern fosters a programming-in-the-large approach as a classroom example. Students learn how to merge different conceptual models (views) to a complete system model, and to develop a correct implementation departing from that system model.

PREREQUISITE KNOWLEDGE Students must already be acquainted with the basic concepts of object-oriented system modeling. Also basic programming knowledge is of use, but fluency in OOP is not a necessary precondition.

NOTES/RESTRICTIONS If students are used to a unidirectional, instructor focused lecture style with small spoon-fed increments, this example does not fit. Because of the self experiencing factor of this pattern, students get often stuck with problems concerning detailed questions. The instructor is then responsible for creating the whole picture, the holistic view with the students.

RELATED PATTERNS This pattern can be regarded as an extension to the Lay of the Land Pattern.

EXAMPLE INSTANCES The Parcel Dispatching System described later is an instance of this pattern.

4 The Parcel Dispatching System

In this section, the Parcel Dispatching System example is presented in the format suggested by the OOPSLA'99 Workshop "Quest for effective classroom examples". It is used at Klagenfurt University during the Systems Analysis and Design courses of the Computer Science curriculum. It has also been used for introducing OO methods to practitioners.

4.1 The Parcel Dispatching System Example

NAME Parcel Dispatching System

PURPOSE To demonstrate design issues. The difference between analysis and design objects can also be easily shown. The problem statement is intentionally not extremely object-oriented, but it is easy to find proper objects.

PREREQUISITE KNOWLEDGE Basic OO-modeling and conventional programming knowledge.

DESIGN The following problem statement describes the behavior of the dispatching system (see also fig. 2 for an schematic overview) we want to develop. The aim of this system is to control the diverse active parts, to maintain a history of the activities of the system and to detect occurring anomalies and react in a consistent manner. In the discussion of the system we focus on the controlling part only.

> Incoming parcels circulate continuously on a conveyer belt. Beside the belt is a slot located. If this slot is free, the next circulating parcel drops into the slot and a scanner reads the label on the parcel. The label contains information about the destination and the type of delivery which can be *normal* or *express*.
>
> After this information is gathered, a robot fetches the parcel and puts it in an appropriate compartment onto a shelf on a rack. Every destination has two shelves assigned, one for the normal deliveries and one for express deliveries.
>
> Shelves are emptied by a distributor (a mechanism responsible for one or more shelves). If an express parcel is put on an express shelf, the assigned distributor immediately takes the parcel from the shelf and takes it to further processing. Normal

shelves are filled by the robot until they run out of capacity. In that case, the robot's arm stops and the distributor opens the locked gate of the shelf and empties it completely. After positive acknowledgment from the distributor, the conveyer belt starts over and the robot resumes.

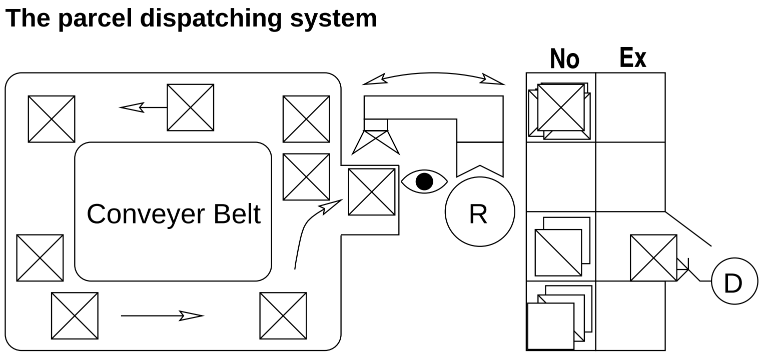

Figure 2: The system overview. The conveyer belt holds some parcels. A parcel is scanned by the integrated scanning mechanism (symbolized by the eye) in the dispatching slot while the robot (R) puts a parcel in its place on the shelf. Meanwhile, the distributor (D) empties a shelf.

RESOURCES The problem statement, analysis and design models for the Parcel Dispatching System, as well as sketched implementation can be found at
http://rurutu.isys-e.uni-klu.ac.at/pedpattern/

NOTES/RESTRICTIONS This example is an instance of the Holistic Nutshell Pattern.

One special issue is the problem of the incompleteness of the problem statement. Particularly, the starvation problem inherent in this example is normally detected in the design phase when event traces are modeled, or already earlier during analysis. This should be discussed with the students to show them the need for process back tracking and for iterations in the process model.

4.2 Developing the System

The following section highlights some issues, problems or hints arising when solving the example in a classroom situation.

4.2.1 Analysis

From the problem statement given above we conclude that the main focus of the analysis should be on static and dynamic aspects. This is justified due to the fact that the only flow we know up to now is concerned with parcels, which are not undergoing any transformation, but are assigned to different locations within the dispatching system.

Static Model We identified twelve different analysis classes which are listed in table 1. This identification can be performed by applying techniques like CRC (Class Responsibility Cards) [8].

Class	Superclass	Attribute
Robot		
Arm		
Scanner		
ConveyerBelt		
Parcel		destination
ExpressParcel	Parcel	
NormalParcel	Parcel	
Distributor		
Rack		
Shelf		destination capacity currNum
ExpressShelf	Shelf	
NormalShelf	Shelf	

Table 1: Identified classes during analysis

The first sketch of the static analysis model is shown in figure 3. An interesting observation is the fact that the object *parcel*, which is obviously important in the analysis phase, can statically not be associated to any other object. Because the requirements do not state the need for knowing the current location of a parcel, the information at what point in time a parcel was at what location in the system is not modeled.

The cardinalities are chosen according to the assumption of the weakest precondition. Therefore the scope of the analysis model is not restricted unless more information is available.

Dynamic Model We used scenario technique as a way to understand the behavior of the most important objects. The students are ordered to sketch some communication examples and create event traces for visualizing them. The event trace of fig. 4 is an example for the normal operation of the system:

> After the object is created and initial values for the destination, type and capacity are entered, it is operationally ready as long as capacity is available. Now it could accept the message *takeParcel*. One of the reactions to this message is to increment the internal *currNum*-counter, which holds the number of the parcels stored in the shelf. If the the maximum capacity is reached, the parcel turns to the state *full* and a message is sent to the robot's arm to stop all operations. After the request of the robot to start emptying the shelf, the dispatcher is informed. The dispatcher must unlock the gate of the shelf and the parcels can be taken. For each piece the dispatcher takes, the internal counter of the shelf object is decremented. Next the dispatcher is responsible to lock the gate and after informing the

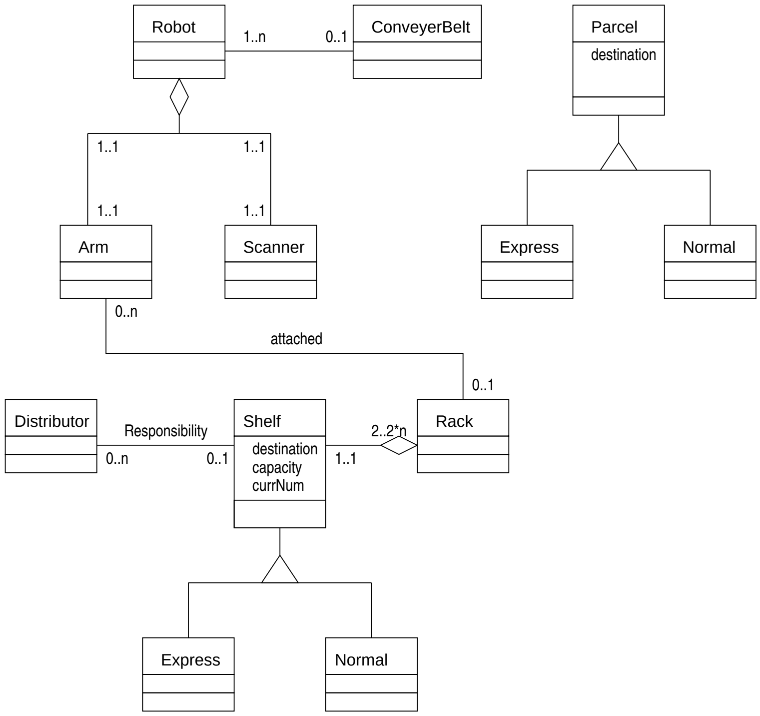

Figure 3: The static analysis model of the system.

robot to resume its work, the shelf object is able to accept further parcels.

.

A rough sketch for the most important objects is produced. For example, in figure 5 one can see the dynamics of the object *shelf*.

Functional Model The functional model is aimed to reveal the overall functionality expected from the system on the basis of data flows. Since we identified only little data processing in this system, the data flow model of the system is very simple. We identified the three processes of queuing, assorting, and distributing a parcel. Although a parcel is handed over to these processes, no computational transformation is performed on it. For this reason no further refinement of the dataflow model is necessary, but the identified process steps must be detectable in the future system. Hence, the data flow model is seen as a specification part and serves as basis for testing and verifying the system later on.

4.2.2 Design

Before starting the detailed design development process, we discuss the impact of various design criteria on the resulting design models. On the basis of the given

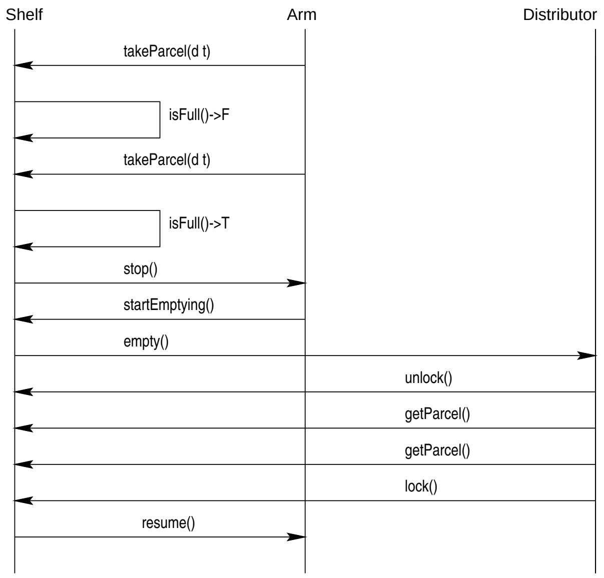

Figure 4: Event trace of normal case of operation.

example we discuss design criteria like extensibility, reusability, maintainability, or performance and possible tradeoffs among them.

E.g., when stressing the importance of reusability, every object must be designed to be self-containing and independent as much as possible from other system parts. This induces the consequence that, in our example, a central controlling object is responsible for most of the system's steering and exception handling. If the contrary, performance is one important goal, such communication overhead may not be acceptable and each object performs a substantial part of the system's logics.

Following that line of discussion, students will become aware of the effects of diverse design criteria. In our opinion it is important to discuss such design variants to stress the point that there is not "THE" final design. The resulting design model is in fact a product of a multi-dimensional optimization process and therefore many reasonable solutions may exist.

Static Design Model For our classroom example we chose **understandability** and **reusability** as high priority design criteria. As a direct consequence we introduced the design objects *Controller* and *SystemObject*. The controller is in charge for containing the specific application logic, which depends on the specific system we want to model. All other functional parts are hidden in the diverse remaining analysis entities. In doing so, we made a sharp distinction between domain objects, which have a structure and behavior not changing rapidly over time, and the application logic, which is the behavior of the whole system according to the functionality demanded by the user.

Furthermore, some of the identified objects have to be deleted, although they play an important part in the domain. The main criteria for deleting objects is a negative answer to the question: Does the object provide structural, behavioral, or informational use for the to system?

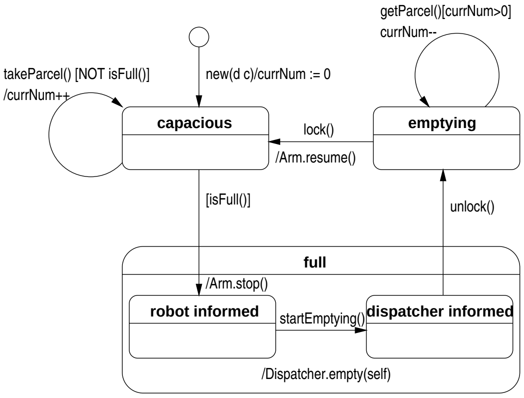

Figure 5: Dynamics of the shelf object.

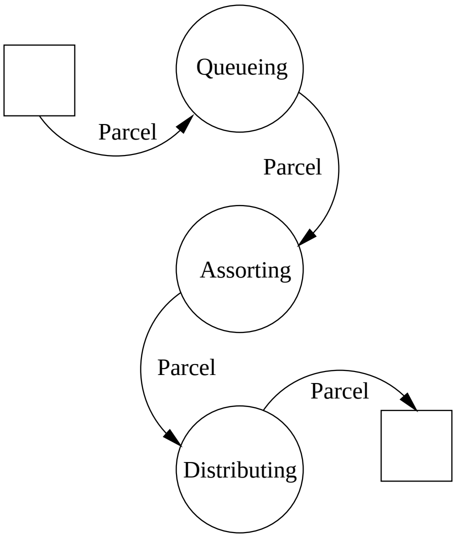

Figure 6: The data flow model of the dispatching system.

Robot and rack are not considered further, because they have no meaning in the context of parcel dispatching. The scanner itself is not important, but we recognized that the method `scan` is indeed part of the arm's functionality. The discussion, why we omit the object *Parcel* can be much more stimulating. But as we only have to know how many parcels of which type are within the system (in the shelves) detailed information about what parcel (its object identity) is when at a certain location, is not required. So we decide to delete the parcel structure. Furthermore, as the express shelf object is an extension to the normal shelf, the normal shelf object is recognized as shelf and its structural representation is assumed by the superclass *Shelf*.

As a direct side effect of the deletion of the classes *Robot* and *Rack* the association from *Robot* to *ConveyerBelt* is now reassigned from *ConveyerBelt* to *Arm*.

Figure 7 summarizes the changes that occurred during the evolution from

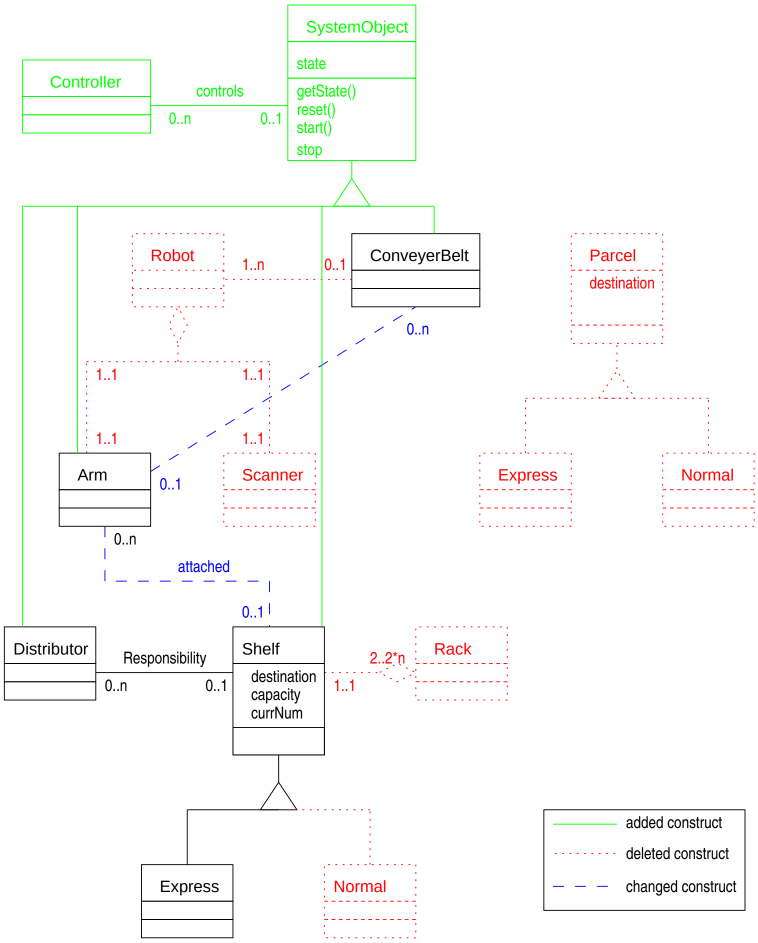

Figure 7: Evolving the static analysis model to the static design one.

the analysis to the design model. The resulting static design with respect to the given design criteria is shown in figure 8. Some of the methods in the objects of the design model are only understandable when considering the dynamics (see section 4.2.2) and statics in parallel.

Dynamic Design Model Thinking about scenarios, the students may recognize a situation where a parcel of type *normal* arrives and is put to its compartment correctly. But if no more parcels for that destination will arrive, this shelf is never emptied, which is normally not intended. Our analysis model allows such behavior, leading to the discussion of incomplete requirements and how to rework the analysis and design models developed so far.

With the students we discuss how to prevent parcel starvation. Shelves should be emptied by a periodical emptying action, analyzing the last access to all shelves. But the question arises: Which object should be responsible for this work? During the discussion of this problem (and bringing further aspects of

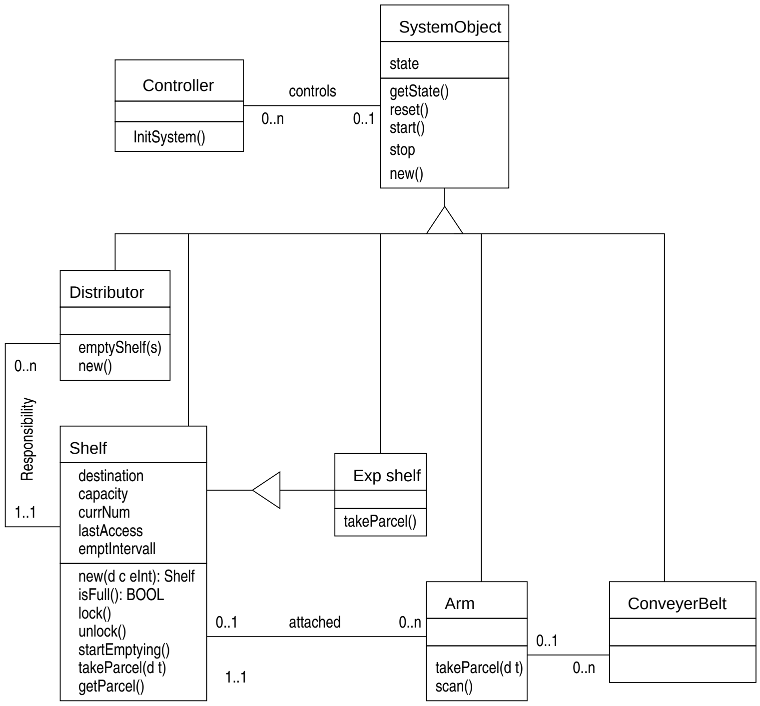

Figure 8: The static Design Model resulting from the revised Analysis Model

exception handling into the design space), the existence of a controller object is justified once more. Alternatively it is also conceivable to shift the responsibility for periodical emptying to the object *Distributor* or to the object *Shelf*.

To foster our main design goal "reusability" we decided to put the responsibility for periodical emptying to the shelf object itself. To enable this approach, an attribute `lastAccess` is attached to the object *Shelf*. This variable is needed for calculating the storage period of each parcel in its compartment. Additionally, the interval determining the period after which emptying is started must be specified. We do so in adding a parameter *eInt* to the *new*-method of the shelf object for initializing the internal attribute *emptIntervall*. The resulting behavior is modeled in fig. 9.

5 Pursued Issues

The following issues show, why in our opinion, the Parcel Dispatching System is an effective classroom example for introductory courses.

5.1 The World is not that OO one always believes

From the modeling perspective, our example is different from "classical introductory" examples, like those used in [6, 8] and other books. In those examples,

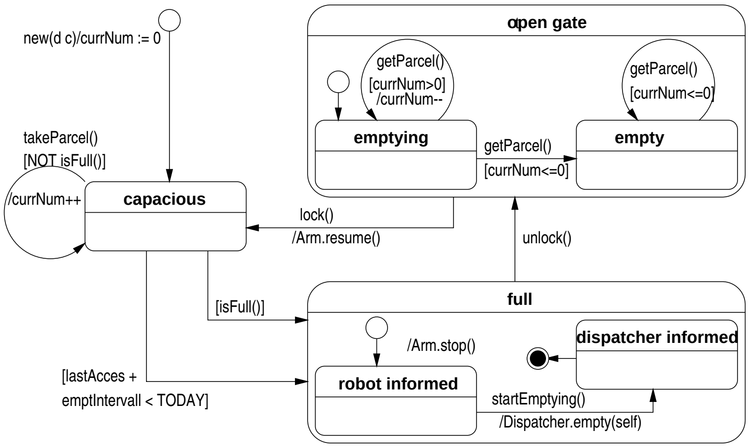

Figure 9: The state chart modeling the dynamics of the object *Shelf*.

the static model can "easily" be derived by reading the problem statement, but this is not always the case in real world problem analysis.

Taking a closer look at our example one can see that there is some potential for students to think more deeply about the problem field and conceive serveral reasonable alternate solutions.

For example, one could identify an object *Parcel*. Further analysis of this object will possibly lead to two further decompositions of this object into an *Express Parcel* and a *Normal Parcel*. But after further reconsideration of these objects and of the problem statement, one should come to the conclusion that *Parcel* is not an object of the system, but data to be processed resp. a message to be passed.

5.2 Analysis Objects vs. Design Objects

As stated in section 2, the development process dimension is also considered when designing classroom examples. We pay attention to this issue, when discussing different design solutions departing from the analysis model of our example. The basis for this discussion is an understanding of several design criteria and strategies. For criteria one could think of reusability, maintainability, performance, the use of several design patterns etc. Some of these criteria are indifferent, complimentary, or contradictory to other criteria. These discussions should not be pursued in depth, but students should gain an awareness that analysis objects need not necessarily be identical to design objects. They may be changed or replaced by other concepts.

Within our example, there are several points of discussion. The *Controller* (see the resources mentioned in section 4.1) object for example, is an object not appearing in the analysis model. It covers, among others, the functionality of a shelf to stop the conveyer belt and the robots arm if its capacity limit is reached. Hence, within the design model, the *Shelf* object is changed and an additional design object (the *Controller*), responsible for exception handling, is introduced.

Following other design criteria, like responsibility driven design [8], there would be no Controller object.

5.3 Further Design and Implementation Issues

When extending the scope of the example on the development process dimension to the implementation, several further issues like packaging or collaborations can be discussed with the students.

5.4 Three Dimensions

- As described in the previous sections, the example covers the analysis as well as the design phase of the development process, hence it satisfies at least two phases of the *process dimension*. It can easily be extended to the implementation level as well.

- The example contains a static, dynamic and functional view. Therefore the lecturer can easily shift the focus on the *system models dimension* if needed.

- Concerning the *prerequisite knowledge dimension*, the example contains various parts/issues demanding knowledge at different levels from the students.

  Basic knowledge in object-orientation is sufficient to identify most of the classes. Taking the *Parcel* class into account, a discussion about class vs. value can be pursued to strengthen these basic skills. Also identifying further abstraction concepts like aggregation or generalization (e.g. *Shelf* or *Robot*) can be performed by beginners.

  Medium knowledge is required for issues like determining the systems boundary during analysis (e.g. is the *Scanner* part of the system or not).

  Detailed knowledge and some experience in OO development is necessary for tasks like finding and assigning methods to classes. This is due to the fact, that this task can only be accomplished by continuously integrating the different views of the system. Another issue is that incomplete requirements forces the students to iterate the whole development process more than one time raising issues like configuration and version management.

The Parcel Dispatching System, when placed within our framework (cf. figure 1) would result in a three-dimensional figure. We refer to such examples as three-dimensional classroom examples. We claim that effective holistic classroom examples should cover the three dimensions of our framework. Which of them are specifically stressed is at the instructors discretion.

6 Conclusions

In this paper the Holistic Nutshell pattern was presented. Designing classroom examples by following this pattern leads to three-dimensional examples. We claim that teaching OO in three dimensions is one way to provide the learner with a holistic view of object-oriented system development.

An instance of this pattern, the Parcel Dispatching System, was presented as a classroom example for introductory courses in object-orientation. It is a three-dimensional example, as it covers several phases of the development process as well as several system models. It requires basic knowledge of the learner.

References

[1] B. Henderson-Sellers and J. Edwards. The Object-Oriented Systems Life Cycle. *Communications of the ACM*, 33(9):142 – 159, Sept. 1990.

[2] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison Wesley, 1999.

[3] H. Kaindl. Difficulties in the transition from oo analysis to design. *IEEE Software*, 16(5):94–102, Sept. 1999.

[4] M. L. Manns, H. Sharp, M. Prieto, and P. McLoughlin. Capturing Successful Practices in OT Education and Training. *Journal of Object-Oriented Programming*, 11(1):29–34, Mar. 1998.

[5] The Pedagogical Patterns Project - Successes in Teaching Object Technology. http://www-lifia.info.unlp.edu.ar/ppp/, (current 18/08/1999).

[6] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall Internationl, 1991.

[7] J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modeling Language Reference Model*. Addison Wesley, 1999.

[8] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. Prentice Hall International, 1990.

A Biography of the Authors

Helfried Pirker works as a university assistant at the department of Informatics-Systems at Klagenfurt University. His research interests include software maintenance, evolution and formal methods. He has three years of experience in teaching introductory lab courses in software design and test, database design and formal methods within the Computer Science curriculum at Klagenfurt University.

Heinz Pozewaunig works as a university assistant at the department of Informatics-Systems at Klagenfurt University. His research interests include software reuse, system analysis and workflow systems. He has three years of experience in teaching introductory lab courses in programming, software design and test, database design and system analysis within the Computer Science curriculum at Klagenfurt University.

Roland T. Mittermeir is professor at the department of Informatics-Systems at Klagenfurt University. His research interests are in various aspects of software engineering. He has substantial teaching experience in software design and object oriented methodology in four continents. The original version of the "dispatcher example" has been developed for a methodology course he taught in the early '90ies at the University of Constantine, Algeria.