Using UML for Business Object Based Systems Modeling

A. Korthaus

Lehrstuhl für Wirtschaftsinformatik III,
Universität Mannheim, D-68131 Mannheim, Germany

Abstract: The development of business information systems based on business object components as defined by the OMG is going to complement and simplify the conventional approach to object-oriented software development. The intention is to enable end users and business experts to assemble "plug-and-play" business objects without the help of IT experts. In spite of the reduction of complexity to be expected, a model-based approach to systems development is still essential. Business objects are not only well suited as modeling concepts in software engineering, they may even be used in the context of business engineering. This paper describes different aspects of modeling business objects with the Unified Modeling Language and evaluates its aptitude for that purpose.

1 Introduction

Today, enterprises need to adapt themselves in a flexible way to a rapidly changing environment, in order to remain competitive. To keep up with the dynamic competition, a considerable number of measures has to be taken, for which buzzwords such as *Business (Re-)Engineering*, *Continuous Improvement* etc. have been coined (e.g. in Hammer and Champy (1994)). The basis for these measures is the modeling of the business and its business processes in order to cope with the complexities of the real world. Based on the business models, software systems have to be built, which are one of the most important enablers of modern successful business activities, not only since new WWW-based processes have become feasible (Davenport (1993)).

Recognizing the close connection between the business perspective and the software perspective, and trying to exploit the advantages of object technology (e.g. modularization, encapsulation, inheritance, polymorphism etc.), Taylor (1995) has propagated an integrated approach called *Convergent Engineering* (see Figure 1). This synergetic approach suggests the application of object-oriented concepts on every level of the system's lifecycle, from business engineering through analysis and design of information systems to the implementation. Thus, the use of information systems to support newly designed business processes is emphasized. The challenge is to ensure a quick implementation of changed or new business requirements, since timing aspects have become essential today (in addition to keeping the quality high and the costs low).

In the field of object-oriented software technology, mechanisms such as model-based specification, component technology, and solution assembly by end users, which are increasingly gaining attention, seem to be very promising in order to achieve these goals. The Object Management Group (OMG), the leading industry

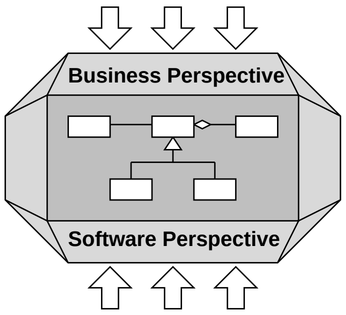


Figure 1: Convergent Engineering

consortium for object technology, is a key driver of these trends. It provides and seeks standards to facilitate the development of distributed information systems. In Nov. 1997, the *Unified Modeling Language* (Rational Software Corp. (1997a)–(1997d)) was adopted as the official standard for modeling languages, and revised proposals for so-called *Business Object* components and a *Business Object Facility* as an infrastructure for those components based on CORBA are currently being evaluated (OMG (1996)).

After clarifying the concepts of UML and OMG Business Objects, this paper will analyze exemplary requirements originating from business object based modeling and implementation of business and software systems.

2 Modeling with the UML

The UML version 1.1 is the new industry standard for object-oriented analysis and design modeling languages. It combines common concepts of earlier analysis and design methods, defines standard notational symbols for those concepts, and enhances the basic set with new modeling elements meeting the requirements of current modeling tasks. The UML is a general purpose language designed to be able to model a wide range of different types of systems, from purely technical (non-software) through software to business systems.

In this paper, the focus is on business and software systems. According to Eriksson and Penker (1998), business systems "describe the goals, the resources (humans, computers etc.), the rules (laws, business strategies, policies etc.), and the actual work in the business (business processes)". Information systems, on the other hand, "store, retrieve, transform, and present information to users. [They] Handle large amounts of data with complex relationships, which are stored in relational or object databases."

Modeling a system with UML means describing different views of the system under construction with the help of a number of diagrams, containing modeling elements. *Use Case Diagrams*, for example, express the externally visible behavior of the system and its interactions with actors, e.g. users or different systems. They are just right for modeling business processes or information system requirements on a very abstract level. *Class Diagrams* are the structural part of the system model, expressing elements or entities of the respective business or software system. A group of diagrams serves for describing behavioral aspects. Among them are *State Diagrams*, *Activity Diagrams*, *Sequence Diagrams* and *Collaboration Diagrams*. The implementation level may be described by *Component Diagrams* and *Deployment Diagrams*.

Compared with its predecessors, the analysis and design methods Booch, OMT and OOSE, UML adds some new elements: extensibility mechanisms (stereotypes, tagged values, constraints), modeling support for patterns and collaborations, Activity Diagrams, refinement, interfaces and components, and the Object Constraint Language (OCL). As will be shown in section 4, these are very valuable concepts in the context of business object based modeling.

3 OMG Business Objects

Using object-oriented modeling for business engineering is relatively new and not very wide-spread at present, although it is broadly acknowledged as an excellent approach of modeling and documenting businesses and business processes (Ferstl and Sinz (1995), Frank (1994), Jacobson et al. (1995), Taylor (1995)). Analyzing, improving, and implementing business processes is tightly bound up with designing and implementing the supporting information systems. Therefore, if the same object-oriented concepts and models are applied on each level of the system lifecycle, the result will be considerable synergistic effects.

The UML can be used for developing business models, which describe, among other things, resources (physical and informational), rules, goals, and actions (workflows), and for developing business software system models as well. Figure 2 shows an example of an incomplete inheritance hierarchy of typical business elements expressed with UML modeling constructs (classes, inheritance arrows, and constraints which indicate that there are possibly more subclasses that are missing in the hierarchy) as it might be used to build a taxonomy of business elements in an enterprise as a basis for communication. These concepts might be used further in object models describing organizational structures and processes of the same enterprise.

Recently, the OMG, whose focus used to be on technical aspects of distributed information systems solely, has been recognizing the fact that distributed object technologies can only become wide-spread if business users and application developers are addressed directly and can be shielded from technical IT details. In 1993, a Special Interest Group for Business Object Management (BOMSIG) was formed within the OMG, which was upgraded as Business Object Domain Task Force (BODTF) afterwards. BOMSIG issued a Request For Proposal for *Common Business Objects* and a *Business Object Facility*, which solicited proposals for the following two items, based on ideas of Sims (1994):

- "*Common Business Objects (CBOs):* Objects representing those business semantics that can be shown to be common across most businesses.

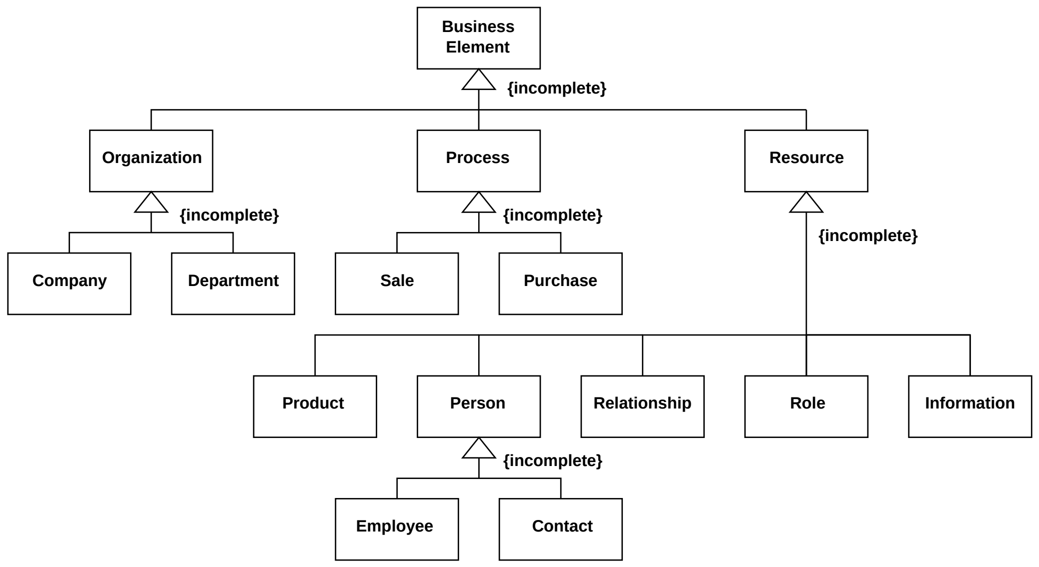


Figure 2: Part of a Specialization Hierarchy of Business Elements

- *Business Object Facility (BOF):* The infrastructure (application architecture, services, etc.) required to support business objects operating as cooperative application components in a distributed object environment." (OMG(1996))

The first part of the RFP asks for Common Business Objects, which represent concepts shared among different business domains (e.g. industry sectors such as finance, insurance, manufacturing etc.). Since it is very hard to reach an agreement on CBOs shared by different domains, the OMG is also trying to standardize domain variations as well as regional variations on CBOs and, of course, purely domain specific business objects not shared between different industry sectors.

After a long debate, the members of the CBO Working Group of the OMG BODTF are now arriving at a common understanding of the term business object. Business objects capture "information about a real world's (business) concept, operations on that concept, constraints on those operations, and relationships between that concept and other business concepts. [...] a business application can be specified in terms of interactions among a configuration of implemented business objects." (OMG (1997a))

Consequently, the term business object is used in two distinct ways, expressing the different areas of application, namely business engineering and software engineering:

1. The term *modeling business objects* designates the usage of business objects in business models to capture business concepts and express an abstract view of the business.

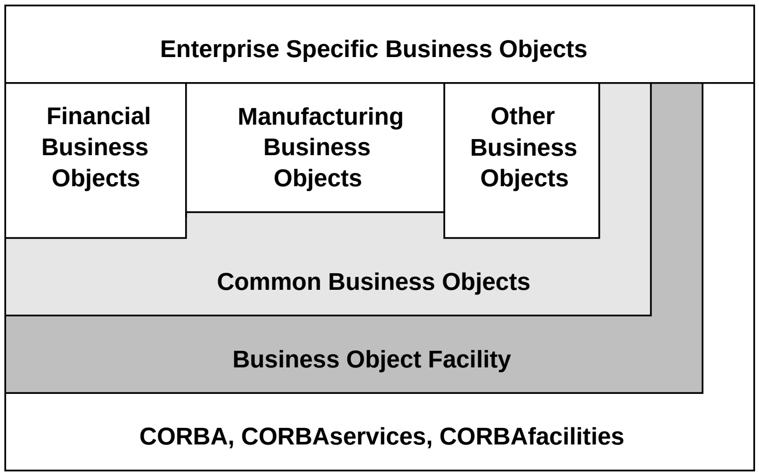

Enterprise Specific Business Objects
Financial Business Objects
Manufacturing Business Objects
Other Business Objects
Common Business Objects
Business Object Facility
CORBA, CORBAservices, CORBAfacilities

Figure 3: Architecture for Business Objects

2. The term *systems business objects* is used in the context of software system designs or program code. Systems business objects reflect the realization of business concepts (represented by modeling business objects) in software.

Business objects are organized into a taxonomy of *Entity Business Objects* (e.g. Involved Party, Location, Product, Agreement, Order, Financial Instrument), *Process Business Objects* (e.g. Order Fulfillment, Procurement, Making a Payment), and *Event Business Objects* (e.g. Account Overdrawn, End of Fiscal Year, Bond Mature).

Figure 3 (OMG (1996)) shows the proposed architecture for business objects. Integrated in OMG's Object Management Architecture (OMA), which includes a reference model for distributed object computing and defines OMG's objectives and terminology, the Business Object Facility (BOF), based on CORBA, provides the infrastructure for CBOs, domain specific BOs and enterprise specific BOs.

The major goals of the OMG Common Business Object (CBO) vision are:

- "Interoperability of OMG Business Objects as both design-time and run-time constructs including the possibility of ad-hoc integration.
- Simplicity, so that design and implementation, configuration and deployment is viable for the average developer." (OMG (1996))

These goals comprise the conceptual and the technical usage of business objects. On the technical level, ad-hoc integration means that run-time business object components (called application components in the RFP) must be able to plug-and-play, i.e. they can be put together in a run-time environment and will then work without any intervention by IT professionals. Furthermore, a newly plugged-in business object will be able to interact with other business objects in order to perform some business function. This interaction can take place even if it was not planned or foreseen by the developers of the business objects. Although originally addressed by the CBO/BOF RFP, efforts of enabling plug-and-play components have been shifted from the responsibility of the BODTF to the Corba Component Initiative (OMG (1997b)) in the meantime.

The CBO vision is aimed at a marketplace for standardized "off-the-shelf" modeling and systems business objects. Generalized descriptions of best business practices and templates for common constructs like Customer, Order, Product etc. will be sold in the form of business object models, supplemented by corresponding software components (systems business objects) which speed up development and are easily integrated with other business objects through the BOF.

4 Requirements for Modeling with Business Objects

Having clarified the context of business object based systems modeling and development, it has to be examined what is required from a modeling language used for this purpose. The requirements analyzed are typical of (but not inevitably restricted to) business object based systems. It is neither a systematic nor a complete evaluation of UML in the business object context, but a discussion of some examplary aspects worthwhile being considered.

4.1 Communication

Communication between domain experts, systems analysts, and software developers is essential for the success of reengineering a business and developing information systems. Divergent vocabulary and different background knowledge on the part of business and IT people often lead to severe problems of mutual understanding. A modeling language used for developing business and software models must be simple enough to allow for this understanding and at the same time detailed enough to specify the system elements unambiguously and sufficiently.

The business object approach emphasizes simplicity and a focus on business issues. Business and software models must be easy to discuss, to verify against the given requirements and easy to maintain. It is very important that concepts from the problem domain are used and expressed in the business terminology.

However, the UML is a general-purpose modeling language which is very complex and provides a considerable number of modeling elements whose semantics must be fully understood by the users of UML. While its complexity lays the foundation for its expressiveness, it also undermines communication with non-experts.

The solution to this problem might be to define different subsets of UML for different modeling purposes on different stages in systems development. For example, when dealing with domain experts, it is advisable to restrict the set of modeling elements to an expressive subset of basic elements useful for conceptual modeling in the business context (e.g. in Class Diagrams: renunciation of association classes, qualifiers, visibility markers etc.). The full variety of constructs is not needed until implementation oriented design tasks are carried out.

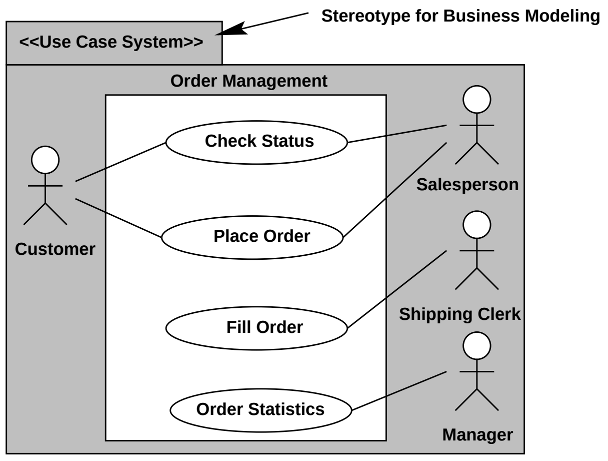


Figure 4: Modeling Business Processes and System Functionality with Use Cases

Another possibility to adjust the UML to the needs of business modeling or business information systems modeling is to use the UML extensibility mechanisms (i.e. stereotypes, tagged values, constraints). Particulary stereotypes applied to an existing metamodel element can help to add new semantics to the language. For example, the stereotype <<business object>> could indicate that a package or a class has certain semantics that are characteristic of business objects. Combining the stereotype with a new presentation icon for the new kind of modeling element may further increase understandability, if the look of the new icon is meaningful to the readers of the model. Rational has already defined an UML extension for business modeling that introduces a number of new stereotypes with corresponding icons and is primarily based on use cases as the central concept for modeling business processes (Jacobson et al. (1995), Rational Software Corp. (1997a)).

4.2 Modeling Goals, Business Processes and Workflows

All business activities are based on strategies, goals and policies which build the framework for the conduct of business and must be determined prior to any other steps during business planning or reengineering. The goals of the business motivate the workflows, where the resources are used in accordance with specified rules. Therefore, modeling activities have to start at the strategic level.

Although UML models can be used to establish a basic common set of concepts and express their relationships and semantics (see Figure 2), the UML is not powerful at this level. Textual commitments and figures will predominate in this context.

The modeling of business processes and workflow, though, can be done very well with the features provided by UML (although the diagrams used for this purpose, i.e. Use Case Diagrams and Activity Diagrams, are not really object-oriented in nature). Figure 4 shows, for example, a Use Case Diagram which represents a

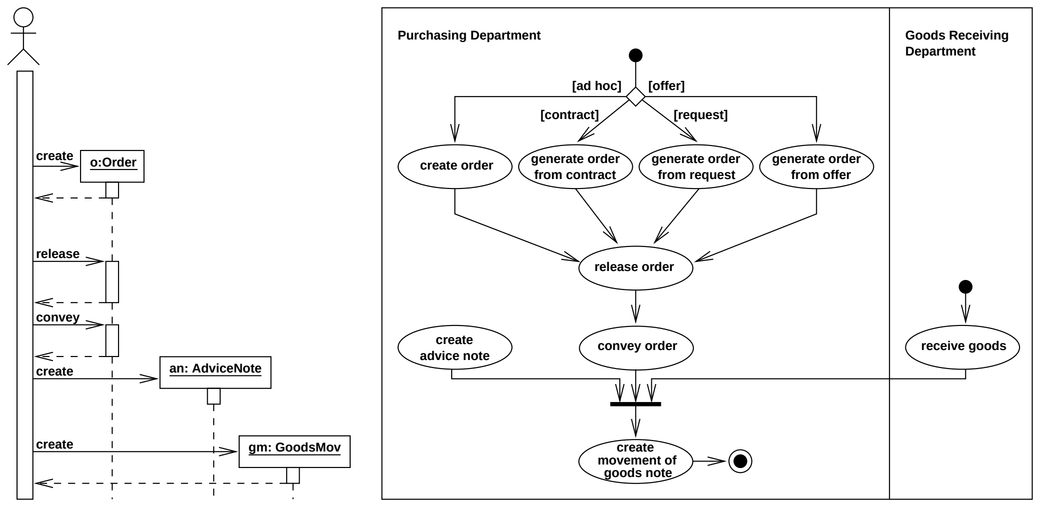


Figure 5: Mapping of Activity Diagrams to Sequence Diagrams

simplified model of the business process "Order Management". Like business objects, use cases can be applied to business modeling, which is portrayed in detail in Jacobson et al. (1995), and also to software systems modeling as described in Jacobson et al. (1992).

Activity Diagrams are a good mechanism for expressing workflow. Workflow designates the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of rules. In future application systems, workflow objects (special business objects) will interconnect other business objects (instead of applications). The OMG seeks proposals for a Workflow Management Facility (WMF) which will have to work together with the BOF at the infrastructure level (OMG (1997d)). Activity Diagrams are used to describe how the work is performed. In addition to showing workflow, they can be used for modeling dynamics of operations, classes, and use cases. The approach to modeling business processes with Activity Diagrams goes hand in hand with use cases or other techniques to capture the requirements of the business system.

A great advantage of Activity Diagrams is their explicit support of modeling concurrent activities, thus helping to reengineer business processes to increase their efficiency (Fowler and Scott (1997)). Activity Diagrams have the potential to replace the wide-spread approach of using Event-Driven Process Chain (EPCs) Diagrams for modeling business processes (for EPCs see Nüttgens et al. (1998)). Due to their flowchart heritage, a disadvantage of Activity Diagrams is the difficult mapping to object-oriented concepts, which is also very difficult for use cases. Figure 5 shows an idealized mapping of the action states in an Activity Diagram to the message flows in the corresponding software systems Sequence Diagram. This will not always be possible, e.g. if different levels of abstractions have been chosen or if parts of the Activity Diagram are performed manually. The mapping of information in Activity Diagrams to classes and objects can be achieved by organizing action states into swimlanes (vertical areas) which indicate the responsibilities of certain classes/objects.

4.3 Partitioning and Refinement

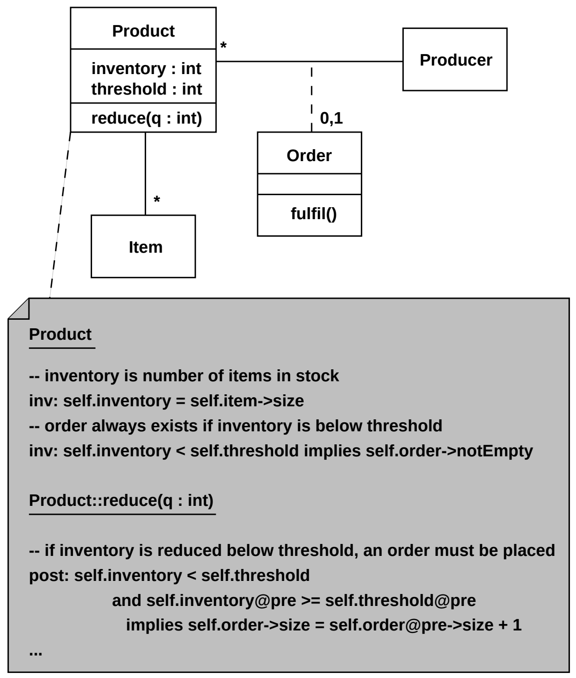
Business systems and their corresponding software systems are usually very complex in nature. Therefore, methods of partitioning and refinement are needed for modeling such systems. The partitioning of large models is not only useful for ergonomical reasons, but also allows for parallel activities of different developers/teams if the dependencies are not too numerous between the different parts of the model. In order to handle a large business object system, a hierarchical approach is needed to refine the basic requirements and incrementally add more detail. Because of the known problems, functional decomposition has to be avoided.

The UML provides a strong mechanism for structuring model parts: packages. Packages are shown as rectangles in a manila folder shape (see left side of Figure 10). By grouping model elements which logically belong together, packages serve for partioning and decomposing the models. Packages may contain other packages (the whole system is contained in a top-level package labeled with stereotype <<system>>), can reference other packages, and inheritance between packages may apply, which means conformity of interfaces and the support of polymorphism. Not only classes, but, in principle, all modeling elements of the UML can be organized in packages. To ensure a loose coupling, dependencies between packages should be minimized. A useful technique supporting this goal is the instantiation of the *Facade* pattern (Gamma et al. (1995)) to hide functional details of a package from other packages. Facades should be labeled with the predefined stereotype <<facade>>. Although the package concept of UML is very extensive, it should be noticed that its specification is not yet free of imprecisions or even contradictions (Schürr and Winter (1998)).

Another UML mechanism for complexity management are refinement relationships which are a special kind of dependency relationships. Like all dependencies, refinements are shown as dashed arrows between two model elements, labeled with the keyword <<refine>>. The refinement construct can be used to express a historical or derivation connection between two elements with a mapping between them.

4.4 Modeling Business Rules

In every organization there are rules which determine how several tasks have to be executed. These business rules define and lay down admissable proceedings, based on ethical, cultural, legal or internal business conditions. They often express policies and refer to administrative processes. Some of the rules are just implicitly part of the employees' know-how, others are explicitly written down



```
Product

-- inventory is number of items in stock
inv: self.inventory = self.item->size
-- order always exists if inventory is below threshold
inv: self.inventory < self.threshold implies self.order->notEmpty

Product::reduce(q : int)

-- if inventory is reduced below threshold, an order must be placed
post: self.inventory < self.threshold
            and self.inventory@pre >= self.threshold@pre
              implies self.order->size = self.order@pre->size + 1
...
```

Figure 6: Expressing Business Rules in Object Constraint Language (OCL)

in organizational handbooks. If information systems have to be built to support the conduct of a business, these rules have to be represented in the applications (Knolmayer and Herbst (1993)).

Since business rules change frequently, they should not be scattered in code as they are in traditional applications, because this has led to poor maintainability. Most of the time, business rules will be implemented in the methods of the business objects affected by those rules. The CBOF proposal to the OMG RFP for CBOs/BOF (OMG (1996)) describes a business object architecture where rules may be expressed as components in their own right and can be plugged in and out as needed (CBOF (1997)). Another aspect to be considered is the fact that domain experts often are aware of business rules, which means that they are a good means of communication. As a result, modeling languages used for modeling business information systems must have an explicit support of business rule modeling.

UML version 1.1 includes a language for the specification of expressions during modeling, called Object Constraint Language (OCL). In the UML documentation, OCL is used to define well-formedness rules expressing the semantics of the UML modeling elements. The main advantage of OCL is its simplicity while at the same time it has got a certain degree of formalism. A considerable disadvantage of OCL might be its lack of rigor. There is no metamodel for OCL in the UML, and the specification of OCL is not always precise and free of ambiguities (see Gogolla and Richters (1998)).

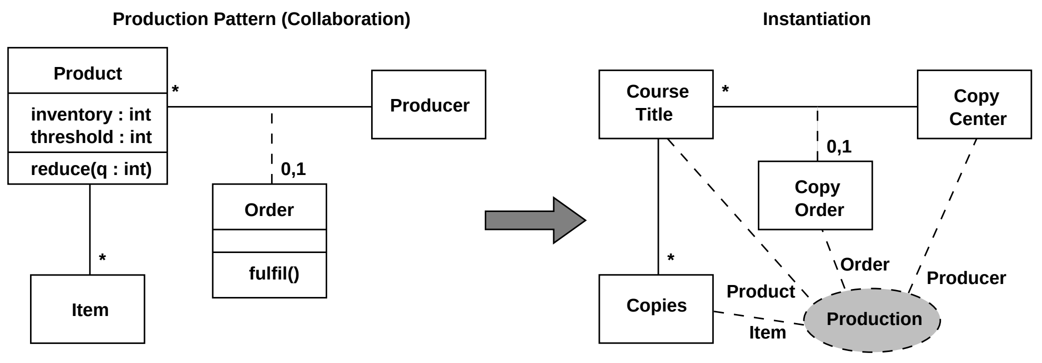


Figure 7: UML's Support for Patterns

To illustrate how OCL can be used to express business rules, Figure 6 provides an example. OCL expressions are very suitable for specifying pre- and postconditions of operations and invariants of operations or classes/types (see subsection 4.7). The example shows a simplified production pattern, which describes the production of different kinds of products by producers, based on orders. For each product type there is an inventory of items, the size of which is stored in the product object's attribute inventory. Furthermore, a threshold value is administrated which determines, when a new order has to be placed. The business rules behind this production pattern might well be defined as constraints in OCL, as can be partially seen in the figure.

4.5 Patterns and Reuse

One of the most important progresses in object-oriented analysis and design are the results of the ongoing research into analysis and design patterns (e.g. Fowler and Scott (1997), Gamma et al. (1995)). These patterns describe best-practice solutions to recurring analysis and design problems and facilitate high level reuse of model specifications. Having in mind the coarse granularity of useful business object components (implemented as collaborations of smaller business objects), patterns and collaborations can help very much in documenting and specifying the semantics of those components, thus laying the foundations for their efficient reuse. Thinking of ready-to-use, "off-the-shelf" systems business object components, patterns of corresponding modeling business objects could serve as documentation of their functionality and possible usage.

Collaboration Diagrams in UML are the means to document behavior implemented by sets of business objects that exchange messages within an overall interaction to accomplish a purpose. Patterns can be defined as parameterized collaborations in UML with types/classes, relationships, and operations as the instantiation parameters. Figure 7 repeats the production pattern of Figure 6 and shows an example of its instantiation into a problem specific UML Class Diagram.

The dashed ellipse symbol contains the pattern name and the dashed lines labeled with the pattern roles are attached to the participants of the pattern.

The use of this notational convenience in UML proves to be very useful, because it represents a design construction on a higher abstraction level than the basic elements, which can be repeatedly applied to different designs. Eriksson and Penker (1998) speak of UML patterns as "generators of design solutions", because they implicitly contain context and interactions, and thus, if not all parts of the design are shown, simplify the models.

Furthermore, building larger models by composing generic patterns is possible with this modeling construct. In this case, if two patterns are to be combined, there is at least one common class/type playing a different role in each pattern. Assuming that the semantics of business object components will be specified by collaborations, the composing of these specifications could be the modeling level preparation for plugging the corresponding systems business object components together. The construction of models and specifications by composition is described further in the Catalysis method (see Catalysis (1997)), D'Souza (1997)).

4.6 Roles and Actors

Separation of concerns is an important principle for design, specification and implementation of business object systems. Different departments and development teams will inevitably access and modify business objects in different models for different reasons. This variety of interest in single business objects was one of the factors which made monolithic enterprise objects impractical. The solution to this problem are roles. Roles represent aspects of objects connected with a certain view or a certain part of a model. A role expresses what an object represents in a special context, not what it is. Roles mirror types that can be adopted and released dynamically (see Reenskaug, Wold and Lehne (1996) for the OOram method, in which role modeling is the central concept).

An actor is a role a person or an external system takes over in regard to the system being modeled (see Figure 4: Customer, Salesperson...). For example, there might be many business partners who place orders, each of them playing the role of a customer. However, they are not restricted to that role, but might adopt other roles at the same time.

One way to express these semantics is the application of dynamic, multiple classification. Fowler and Scott (1997) describe how dynamic, multiple classification can be modeled in UML with the help of stereotypes (see Figure 8). If dynamic, multiple classification is allowed, objects can change their class over time and can belong to more than one class at a time. Thus, the requirements of role modeling are met. However, this concept is rarely used because it is usually not supported by programming languages.

An indirect way of modeling roles without the need for dynamic, multiple classification, is the application of role patterns. Figure 9 shows a delegation based design of the same problem domain as illustrated in Figure 8 on the basis of a role pattern instance.

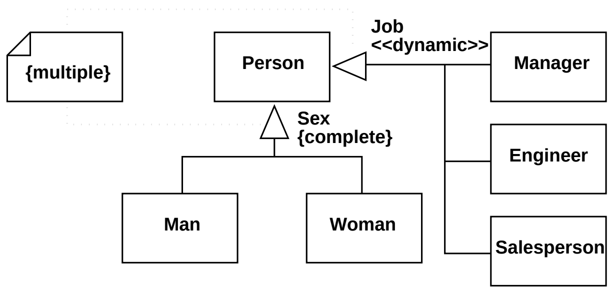


Figure 8: Dynamic, Multiple Classification

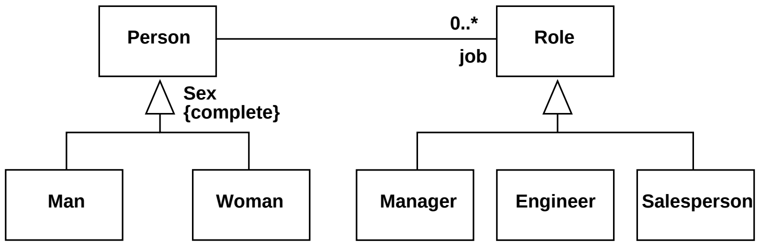


Figure 9: Role Pattern with Delegation

In UML, the concept of roles is supported directly in three more ways: In addition to roles in associations which have simpler semantics, roles are represented in UML by the actors in a use case and by the participants in a collaboration. An Actor element in a Use Case Diagram characterizes the role played by an outside object which may play several roles and therefore be modeled by several actors. In a collaboration, roles must be bound to actual classes when the collaboration is used. The roles may have their own names and possibly show only subsets of the attributes, operations and related objects known to actual classes playing the role. An actual class may play several roles in different collaborations (Rational Software Corp. (1997b)).

4.7 Component Technology

Another aspect which has to be considered is the fact that implemented OMG systems business objects will be runtime plug-and-play components. One goal of the business object approach is to enhance flexibility and extensibility of systems by enabling substitution of one business object component for another (e.g. a newer version of a component replaces an older one, or components from different vendors are interchanged).

To support the component character of business objects at the design stage, it is necessary to know the component model or rather the business object architecture that will finally be adopted. At the time of this writing, proposals for the CBO/BOF

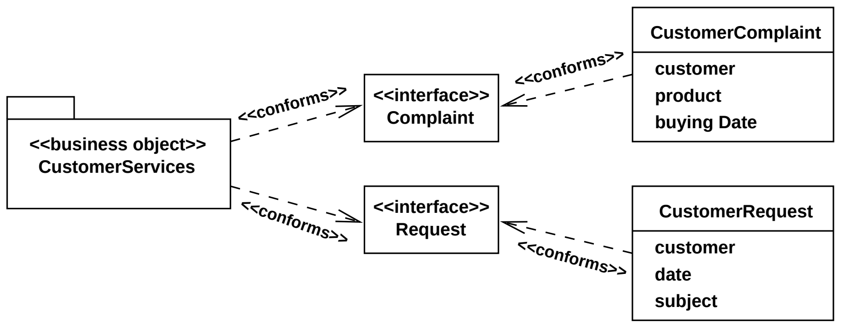


Figure 10: External and Internal View of Business Object Components

RFP (OMG (1997a)) and CORBA Component Model RFP (OMG (1997b)) are being evaluated. One promising BOF submission (CBOF (1997)) will be briefly addressed in section 5.

Bearing in mind current component models, it might be useful to concentrate on modeling with types and interfaces and to ensure separation of concerns and design by contract. Separation of interface from implementation is central in this context. Fortunately, the UML provides a number of techniques for this purpose.

First, the external and the internal view of business object components should be distinguished. Externally visible are the operations or services that are supported by the business object component. The internal realization of the component might be based on an encapsulation of highly interrelated classes, associations, operations, events and constraints. Figure 10 shows a coarse grained business object component CustomerServices modeled as a stereotyped package which exports two interfaces. The classes CustomerComplaint and CustomerRequest on the right belong to the internal realization of CustomerServices and provide those interfaces.

The distinction between an external and internal view is closely connected with the distinction between specification and implementation. Fowler (1997) describes three perspectives in modeling: conceptual, specification, and implementation. The conceptual perspective describes the concepts of the domain under study and is independent of implementation aspects. The specification perspective is concerned with interfaces and types, not implemention classes. Finally, in the implementation perspective, the actual implementation (with classes etc.) is laid bare.

When modeling business object based systems, all three perspectives are relevant. The conceptual perspective should be taken in early phases of systems development (especially business modeling). The specification perspective is very important. Specifying types and interfaces allows for designs that can be used for implementation or assembly of a BOF-based system independently of the source of the business object components that constitute the system. If a component is to be bought externally, its specification must be matched against the model, and its implementation is irrelevant. If the component is to be devel-

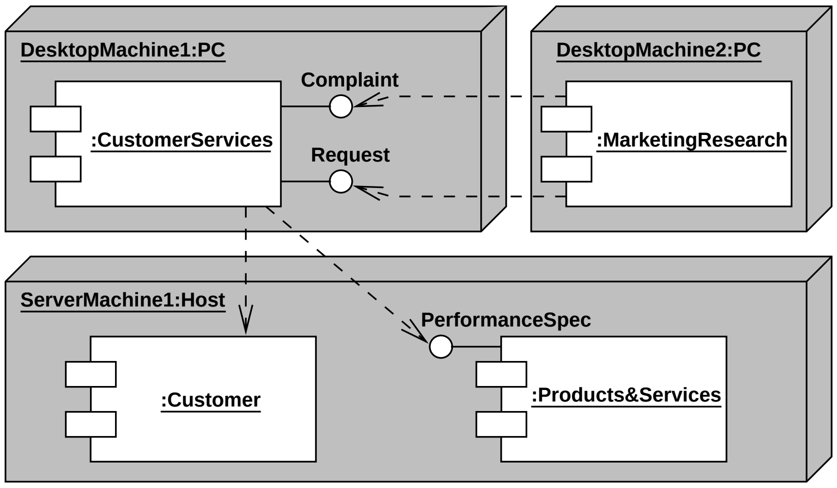


Figure 11: Component Architecture Shown in a Deployment Diagram

oped internally, the specification model has to be refined by a model from the implementation perspective. UML explicitly supports the concepts of types and interfaces. For example, there are two notational ways of presenting interfaces (see Figures 10 and 11), and types can be made explicit by using the predefined stereotype <<type>> in a class symbol. Modeling with types, interfaces and operations implies designing by contract, i.e. specifying boolean assertions which must always be true. There are three kinds of assertions to be distinguished, namely preconditions, invariants, and postconditions. Applied to an operation, a postcondition specifies what has to be the result of the operation without saying how to achieve this result, thus ensuring separation of interface and implementation. Preconditions describe the system state as expected by the operation before execution, thus defining the responsibilities of the caller. An invariant, applied to a class, holds always true at any moment an operation is called. In UML models, assertions should be expressed in OCL. Specifying functionality in this way is essential if implementational aspects are to be abstracted from in order to allow for business component substitution.

In the context of an implementation (and architectural) perspective, only one UML feature shall be mentioned here: Deployment Diagrams, which can be used to show business object components, their physical locations in a distributed environment, and the dependencies between them (see Figure 11). Although describing relevant aspects, Deployment Diagrams appear to be too complex for the benefit they bring.

As can be inferred from the above paragraphs, it is essential to define a suitable process of business object based systems development. The UML standard does not include or prescribe a development process. To allow for a specification based approach of business object component modeling, ideas might be taken from methods such as Catalysis (Catalysis (1997)) and COOL (Short (1997)). Catalysis, which builds on UML, puts collaboration patterns as specifications of components into the center of attention and describes a formalized process of building business models, requirements specs, designs, and code rapidly from reusable generic components and frameworks. Its main advantages (e.g. full model consistency, views and patterns synthesis) stem from its rigor and formality, which are, simultaneously, its main disadvantages, because it is hard to use for non-experts from the business domain.

5 Evaluation and Prospects

In this paper, exemplary aspects of a business object based systems modeling have been analyzed and related to the features and concepts of UML, the new industry standard for object modeling languages. It has been shown that UML is very powerful and provides numerous modeling elements needed for the special purpose under examination. On the other hand, it is necessary both to restrict and to extend certain parts of the UML to provide the semantics needed and to avoid undesired complexity. Furthermore, a systems development process for business object based systems which builds on UML is to be defined to use UML both efficiently and effectively.

With the adoption of a CBO/BOF proposal by the OMG, more information about how these problems have to be solved will become available. One of the most promising responses to the BODTF RFP (OMG (1997a)) is CBOF (1997) which defines a Business Object Architecture (BOA; a metamodel for the constructs and types that are used to build a business object system), a Component Definition Language (CDL; a language for the textual expression of business object specifications that use the metamodel), and an IDL mapping (to specify what interfaces are required to support BOA based models). The proposal is integrated in OMG's Object Management Architecture, i.e. builds on Corba and IDL, but adds the semantics of business components. The rationale for the introduction of CDL is, that it "provides a level of abstraction needed to express the semantics of business components and to combine (compose) medium-grain business components, a capability beyond the OMG's Interface Definition Language. CDL enhances IDL in that it can describe object semantics as well as interfaces. CDL (and semantic definition) is required to support the definition and standardization of common business objects, domain specific business objects and corporate standard business objects." (CBOF (1997))

The BOA model is described as an UML extension, i.e. shared concepts are kept and new concepts are introduced using the UML extension mechanisms. On the other hand, "the BOA and CDL represent a very specific and narrow subset of UML capabilities. Within its scope, however, CDL provides more rigor, more semantic depth for the targeted business object domains, and an unambiguous relationship with OMA." For example, the format of constraints and other expressions is Uninterpreted in UML (i.e. not standardized) whereas the BOA metamodel explicitly models expressions.

If the CBOF proposal should be adopted and a suitable mapping between UML and BOA/CDL metamodels can be defined, UML designs will be the starting point for business object modeling, and will be transformed into CDL specifications automatically. The CDL specification provides the relevant information needed by the business object infrastructure (Business Object Framework, Meta Object Facility (OMG (1997c))) so that business object components can be plugged in, generated by tools or implemented manually. UML models and CDL specifications will then serve as standardized documentation means of the structure and semantics of business object components. Future responses to business object RFPs will require specifications in this form.

References

CATALYSIS (1997): Catalyis—Next Generation Component-Based Development from Object Frameworks. URL: http://www.iconcomp.com/catalysis/.

CBOF (1997): Combined Business Object Facility—Business Object Architecture (BOA) Proposal. OMG Business Object Domain Task Force BODTF-RFP 1 Submission. Revision 1.0. Data Access Technologies et al. OMG Document bom/97-11-09.

COAD, P. and MAYFIELD, M. (1997): Java Design—Building Better Apps & Applets. Yourdon Press Computing Series, Prentice Hall, New Jersey.

DAVENPORT, T.H. (1993): Process Innovation—Reengineering Work through Information Technology. Harvard Business School Press, Boston, Massachusetts.

D'SOUZA, D. (1997): Framework: Java to UML/Catalysis. *Journal of Object-Oriented Programming, 10/5, 10–13.*

ERIKSSON, H.-E. and PENKER, M. (1998): UML-Toolkit. Wiley Computer Publishing, New York.

FERSTL, O.K. and SINZ, E. (1995): Der Ansatz des Semantischen Objektmodells (SOM) zur Modellierung von Geschäftsprozessen. *Wirtschaftsinformatik 37/3, 209–220.*

FOWLER, M. and SCOTT, K. (1997): UML Distilled—Applying the Standard Object Modeling Language. Addison-Wesley, Reading, Massachusetts.

FRANK, U. (1994): MEMO: A Tool Supported Methodology for Analyzing and (Re-) Designing Business Information Systems. In: R. Ege, M. Singh and B. Meyer (eds.): *Technology of Object-Oriented Languages and Systems*. Englewood Cliffs, 367–380.

GAMMA, E. , HELM, R., JOHNSON, R., and VLISSIDES, J. (1995): Design Patterns—Elements of Reusable Object-Oriented Software. Professional Computing Series. Addison-Wesley, Reading, Massachusetts.

GOGOLLA, and RICHTERS, M. (1998): On Constraints and Queries in UML. In: Schader, M. and Korthaus, A. (eds.): *The Unified Modeling Language—Technical Aspects and Applications*. Physica-Verlag, Heidelberg.

HAMMER, M. and CHAMPY, J. (1994): Reengineering the Corporation—A Manifesto for Business Revolution. Nicholas Brealey Publishing, London.

JACOBSON, I., CHRISTERSON, M., JONSSON, P., and ÖVERGAARD, G. (1992): Object-Oriented Software Engineering—A Use Case Driven Approach. Addison-Wesley, Wokingham, England.

JACOBSON, I., ERICSSON, M. and JACOBSON, A. (1995): The Object Advantage—Business Process Reengineering with Object Technology. Addison-Wesley, Wokingham, England.

KNOLMAYER, G. and HERBST, H. (1993): Business Rules. *Wirtschaftsinformatik 35/4, 386-390.*

NÜTTGENS, M., FELD, T., and ZIMMERMANN, V. (1998): Business Process Modeling with EPC and UML: Transformation or Integration? In: Schader, M. and Korthaus, A. (eds.): *The Unified Modeling Language—Technical Aspects and Applications.* Physica-Verlag, Heidelberg.

OMG (1996): Common Facilities RFP-4, Common Business Objects and Business Object Facility. Request for Proposal. OMG TC Document CF/96-01-04. Object Management Group.

OMG (1997a): Business Object DTF—Common Business Objects. OMG Document bom/97-11-11 Version 1.3. Object Management Group.

OMG (1997b): CORBA Component Model RFP. Request for Proposal. OMG Document orbos/96-06-12. Object Management Group.

OMG (1997c): Meta Object Facility (MOF) Specification. OMG Document ad/97-08-14. Object Management Group.

OMG (1997d): Workflow Management Facility RFP. Request for Proposal. OMG Document cf/97-05-06. Object Management Group.

RATIONAL SOFTWARE CORP. (1997a): UML Extension for Business Modeling. Version 1.1, 1 September 1997, Rational Software Corp. et al.

RATIONAL SOFTWARE CORP. (1997b): UML Notation Guide. Version 1.1, 1 September 1997, Rational Software Corp. et al.

RATIONAL SOFTWARE CORP. (1997c): UML Semantics. Version 1.1, 1 September 1997, Rational Software Corp. et al.

RATIONAL SOFTWARE CORP. (1997d): UML Summary. Version 1.1, 1 September 1997, Rational Software Corp. et al.

REENSKAUG, T., Wold, P. and Lehne, O.A. (1996): Working with Objects: The OOram Software Engineering Method. Greenwich, Manning.

SCHÜRR, A. and WINTER, A. (1998): Formal Definition of UML's Package Concept. In: Schader, M. and Korthaus, A. (eds.): *The Unified Modeling Language—Technical Aspects and Applications.* Physica-Verlag, Heidelberg.

SHORT, K. (1997): Component Based Development and Object Modeling. WhitePaper, Texas Instruments Software, Febr. 1997, Version 1.05. URL: http://cool.sterling.com.

SIMS, O. (1994): Business Objects—Delivering Cooperative Objects for Client-Server. IBM McGraw-Hill series. McGraw-Hill Book Company, london.

TAYLOR, D. (1995): Business Engineering with Object Technology. John Wiley & Sons, Inc., New York.