Structuring Specification of Business Systems with UML (with an Emphasis on Workflow Management Systems)

Pavel Hruby

Navision Software a/s
Frydenlunds Allé 6
2950 Vedbaek, Denmark
Tel.: +45 45 65 50 00
Fax: +45 45 65 50 01
E-mail: ph@navision.com
Web site: www.navision.com (click services)

Abstract. *Unified Modeling Language (UML) defines a standard notation for object-oriented systems. Using UML enhances communication between domain experts, workflow specialists, software designers and other professionals with different backgrounds. UML can be used on a general level, which is intuitive for the users of workflow systems. In spite of this, UML symbols also have defined semantics, which means that the visual workflow description can be used as a software specification. This position paper explains how to use UML for specification of workflow management systems, how to trace the description of business processes to the object-oriented software design and how to structure the project repository with UML deliverables.*

Key Words*: UML, Business process, Deliverable, Workflow*

1. Introduction

This paper is organized in the following way. This, the first section, explains the need for a common language among software designers and users of the workflow products. The second section shows how to represent common workflow concepts in Unified Modeling Language (UML). The third section outlines how to connect the description of the business processes with the object-oriented software specification.

The following scenario can often depict the implementation of a new workflow management system in a corporation: a consultant works together with the users to describe the corporate business processes to be supported by a software solution. The team of developers receives the consultant's description, but they have trouble understanding the business terminology and find the description too informal to use for implementing the system. The developers write their own system specification from a technical point of view. When the system specification is presented to the users, they do not quite understand it because it is too technical. They are, however, forced to accept it in order to move forward.

This approach can easily result in a system that does not meet the requirements of the users because often the users, the consultants and the developers don't speak the same language. Such communication problems can make it difficult to turn a description of business processes into a technical software specification that all parties can understand. In addition, because a technical system specification that is not fully understood by the actual users of the system is used, software system becomes difficult to use.

The challenge is to model business processes and business systems in a way that is both precise and user-friendly. Each symbol describing a business process should be intuitive for the user and have defined semantics, so that the developers can use the description as a general, but precise specification of the software system.

UML is rich and complicated notation for describing software systems. The notation is perhaps too rich to be intuitive and user-friendly. However, UML has two advantages, which make it suitable for representing workflow management systems. First, UML is the generally accepted notational standard in the software community and second, UML can be used on a general level, where implementation details are suppressed. The UML diagrams shown in the next section are very similar to those that domain experts are already using intuitively. Moreover, their semantics are defined precisely. The same diagrams can be adorned with implementation details for software design purposes, if necessary.

The description of business systems consists of a description of *processes* and *static structures*. The most intuitive model of a process is a sequence of activities or tasks, performed in order to achieve a goal (Alexander, 1998). Therefore, the UML sequence diagram and the UML activity diagram are suitable for a user-friendly, yet precise, specification of business processes. Static structures, such as an organizational chart, can be represented by UML static structure diagrams without implementation details. Graphical representation of implementation details is often misleading for non-UML experts, for example, navigation arrows are often mistaken for flows. It is advisable to use only a certain subset of UML representation options. For example, it is better to show compositions by placing elements inside each other, rather than by using associations with filled diamonds. Various properties can be represented by text, rather than by UML symbols, for example, the text «refine » is easier to understand than a dashed line with a triangular arrow.

2. Mapping between Workflow Concepts and UML Concepts

This section includes examples of workflow concepts represented in UML. The purpose of this section is only to give a general guideline on how to map workflow common concepts to UML because the details are easy to derive from UML Semantics and Notation Guide (UML Notation Guide, 1997). Every construct of workflow management systems can be depicted by a UML symbol with an appropriate stereotype.

Fig. 1 shows examples of representation of business processes, business objects and team roles in UML. *Business objects* are represented by classes and objects in UML. Classes represent business objects without identity, such as *an invoice*. Objects represent business objects, which have identity, such as *the invoice with the number VM 4/55*. *Business processes*[1] are represented by use cases and use case instances. The use cases are definitions of business processes in terms of goals, responsibilities, preconditions and postconditions. The use case instances are concrete sequences of events. *Workflows* are automated business processes. They can be represented as use cases or use case instances with a stereotype «workflow». *Team roles* are represented by classes and objects in UML. Classes represent types of team roles, objects represent concrete workers playing

[1] In this paper, I assume that business processes are collaborations between business objects, actors and other instances inside a business system. Please see the section "Structuring the project repository" for details.

the role. All the symbols can be adorned with an appropriate stereotype, such as «business object», «business process» and «team role». Each stereotype can be represented either by text or by a specific icon.

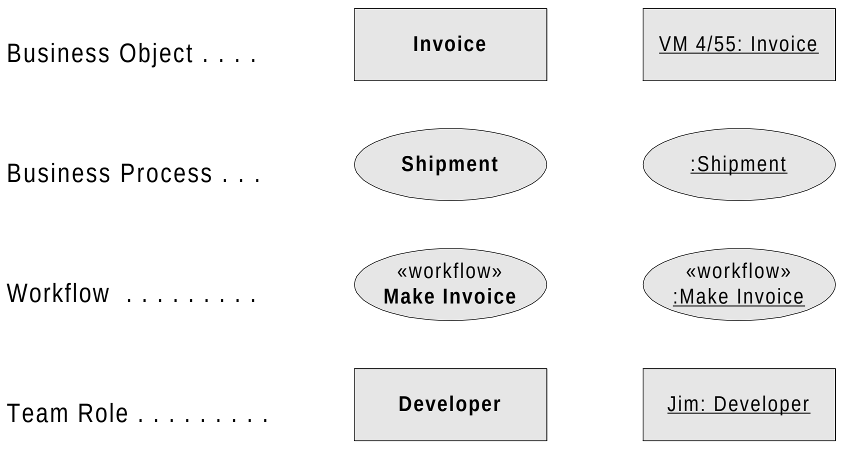


Fig. 1 Representation of a business object, a business process and a team role in UML.

Fig. 2 shows an example of a team structure. The team roles are represented as object instances, which allows for specifying the number of employees in each role. The customer satisfaction team has three developers, two testers, one product manager and one person in the user education role. The group of roles, called the customer satisfaction team, is represented by the package symbol. The group of roles might also be represented as an object – as a composition of roles. If the team is represented as an object and the relationships between the team and the roles are composition relationships, then, according to the UML metamodel, a specific role instance cannot be part of more than one team at the same time. If the team is represented as a package, a specific role instance can be a part of several teams at the same time.

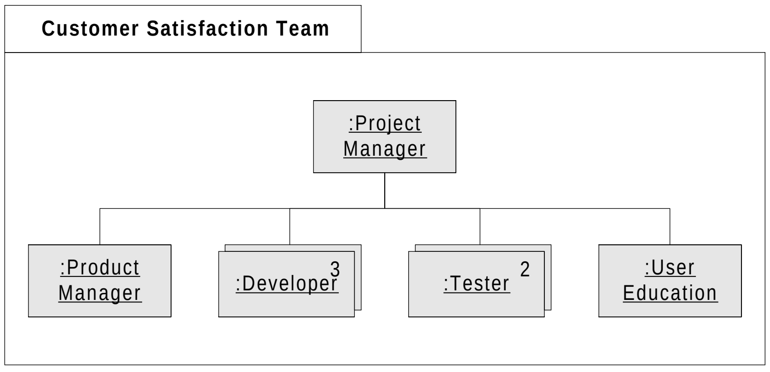


Fig. 2 UML static structure diagram representing a team structure.

Fig. 3 represents the instance of the business process. The actor *customer* places an order, and then an unspecified worker in the *sales* department validates the order. If the order is valid, the worker in *sales* invokes an instance of another business process *company ships an item*. This type of diagram is not explicitly mentioned in the UML Notation Guide. However, it conforms to the UML metamodel. The symbols at the top of the object lifelines represent classifier roles, which in Fig. 3 are the actor role, the object role and the use case role.

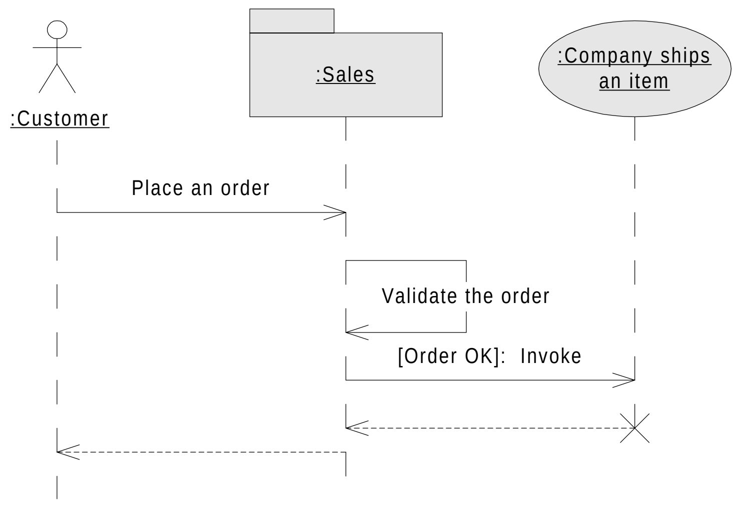


Fig. 3 UML sequence diagram representing the instance of the business process

Fig. 4 is the UML use case diagram, representing static relationships between business processes. The business processes describe collaborations of the *organization* with the actor *customer*. Please note that in UML version 1.1, use cases cannot communicate with each other and they are always started by a signal from the actor. This leads to difficulties in modeling situations in which a use case is started during execution of another use case when a specific condition is met. In such situations, use cases are initiated by the actor through communication with another use case and not by any specific starting signal from the actor. For example, the use case *company ships an item* is started by an object inside the organization if the customer's request has been evaluated as valid. This use case instance is not directly started by the customer. I hope that the next version of UML will weaken this restriction about communications between use cases.

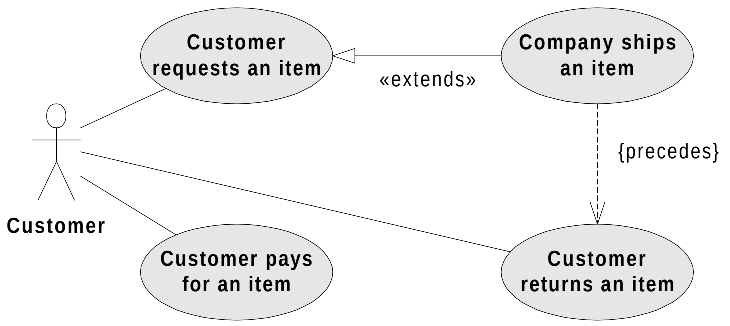


Fig. 4 UML use case diagram representing static relationships between business processes

The UML use case diagram cannot easily represent the order of use case instances, for example, that first the customer requests an item, then the company ships the item and finally, the customer pays for the item. One of the solutions is to use constraints {precedes} or dependencies «precedes» between use cases. Similar relationships exist in OML (OPEN modeling language), see (Henderson-Sellers and Graham, 1997). Robert C. Martin suggested to use the keyword *follows* instead of the *precedes,* please see (Martin R., 1998). The reason for this replacement is that the dependency «follows» points in the opposite direction to the dependency «precedes». The dependency «follows» points in the direction usual for the dependency – from the dependent element to the independent element. Which one is more intuitive is an open question. However, the diagram with constraints or dependencies is still a static structure diagram and does not represent a scenario.

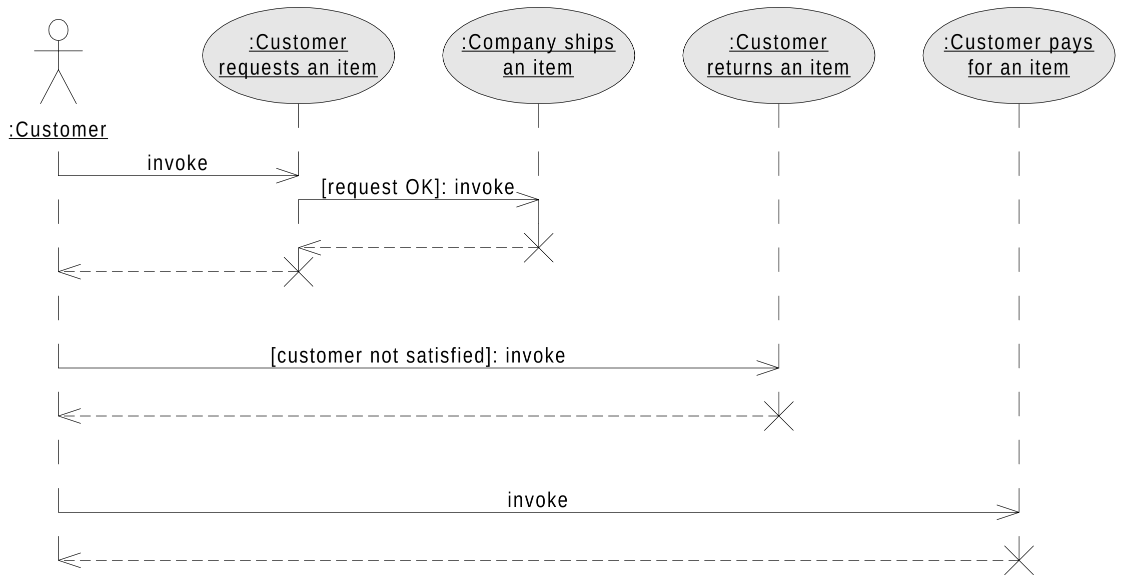

:Customer
:Customer requests an item
:Company ships an item
:Customer returns an item
:Customer pays for an item
invoke
[request OK]: invoke
[customer not satisfied]: invoke
invoke

Fig. 5 UML sequence diagram represents interactions between business processes and actors.

An actor can use a system in a way that initiates use cases in a particular order. Such a scenario – a sequence of use case instances – can be represented by use case sequence or collaboration diagrams, see Figs. 5 and 6. In contrast to object interaction diagrams, where a scenario is described as a sequence of messages, the use case interaction diagram describes the scenario as a sequence of use cases. This diagram is the only UML diagram that can describe a scenario consisting of instances of other scenarios. The messages *invoke* in Fig. 5 represent constructors of the use cases and they map to the signals from the actors to the use cases. They can also be named according to the first operation in each use case, such as *invoke request*, *invoke shipment* and *invoke payment*. Except for these messages, the use case interaction diagram can show other messages exchanged between the actor and the system and describe complete conversations between use case and actor.

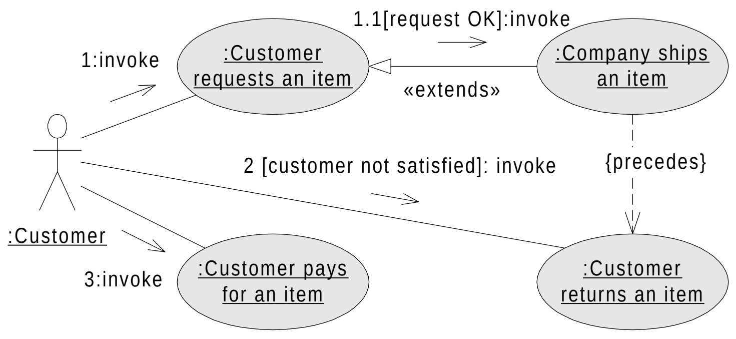

1.1[request OK]:invoke
1:invoke
:Customer requests an item
«extends»
:Company ships an item
2 [customer not satisfied]: invoke
{precedes}
:Customer
3:invoke
:Customer pays for an item
:Customer returns an item

Fig. 6 UML collaboration diagram represents interactions and relationships between business processes and actors.

The use case collaboration diagram is illustrated in Fig. 6. The use case collaboration diagram represents a scenario consisting of instances of business processes. Unlike the use case sequence diagram, the use case collaboration diagram shows both use case relationships and messages exchanged between use case instances and actor instances.

The use case interaction diagrams describe only typical scenarios consisting of use case instances. Therefore, they cannot represent all allowable sequences of use case instances. The allowable order of use case instances belonging to a use case package can be specified in the lifecycle of the use case package. The lifecycle of the use case package is represented by a state diagram, activity diagram of Backus-Naur form (BNF)[2], in which states, action states or BNF statements map to the use cases of the use case package. The use case package lifecycle is a precise representation of the use case package behavior. It can, however, be difficult to develop it correctly, especially in complex cases. The use case interaction diagram is easy to develop, but it describes only typical scenarios consisting of use cases in the package.

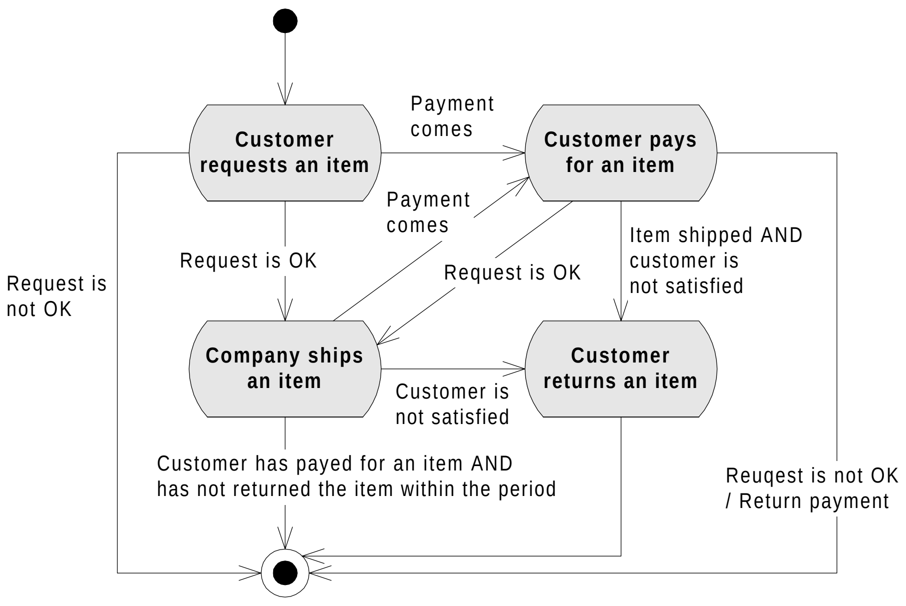

Customer requests an item
Payment comes
Customer pays for an item
Payment comes
Request is OK
Request is OK
Item shipped AND customer is not satisfied
Request is not OK
Company ships an item
Customer returns an item
Customer is not satisfied
Customer has payed for an item AND has not returned the item within the period
Reuqest is not OK / Return payment

Fig. 7 UML activity diagram can represent allowable order of business processes

Fig 7 represents the UML activity diagram representing the lifecycle of the use case package *order management*. The activities on this activity diagram correspond to the use cases discussed in Figs 4, 5 and 6.

Please note that the UML metamodel does not define any mapping from states or action states to use case instances. Such mapping might be defined by the development process, similar to the approach by Martin and Odell, in which states of the subsystem indicate candidate classes in the subsystem, (Martin, J. and Odell, J., 1998). However, other development processes might define the semantics of the use case package lifecycle in a different way. For example, the purpose of the use case package lifecycle might be to specify the allowable order of subsystem interface operations in the scope of the use case package.

There is one more significant difference between the use case interaction model and the use case lifecycle – their placement in the project repository is different and their relationships to other

[2] Please see (Hruby, 1998) for inspiration on how to use BNF for specifying lifecycles.

design artifacts are different. The artifact use case interaction model is related to the artifact use case model. The artifact use case package lifecycle is related to the artifact use case package, which is *one level of abstraction higher* than the corresponding use case model and use case interaction model. Please see Fig. 8 and the next section for details.

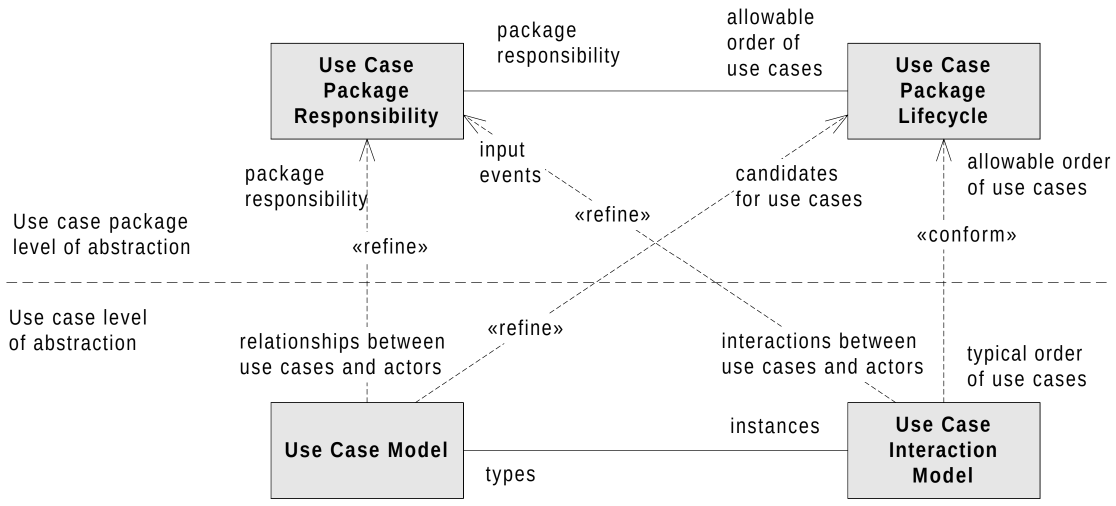

Use Case Package Responsibility
package responsibility
allowable order of use cases
Use Case Package Lifecycle
input events
package responsibility
candidates for use cases
allowable order of use cases
Use case package level of abstraction
«refine»
«refine»
«conform»
Use case level of abstraction
relationships between use cases and actors
«refine»
interactions between use cases and actors
typical order of use cases
Use Case Model
instances
types
Use Case Interaction Model

Fig. 8 Relationships between artifacts in the use case view in the project repository. The notation is modified UML. For better clarity, dependencies were adorned by role ends[3].

3. Structuring the Project Repository

In well-structured design specification, the required information about business processes and software systems can be easily located and closely related information is linked together. The structure should also give an overview about the completeness and consistency between design artifacts. The specification should act as common ground between business experts, consultants and software developers. The structure should also support clear separation of concerns, so that business people - and not the developers - define all business rules. This section identifies relationships between business processes and software design artifacts, which can be used for structuring the project repository. This section gives a guideline for structuring information about business processes and about their relationships to software design artifacts. Please see reference (Hruby, 1998) for concrete examples of the repository structures.

The structure is based on the assumption that business processes (represented as use cases in UML) are collaborations between business objects, actors, workers or other instances in a business system. I assume that definitions of business processes (use cases) are *types* of collaborations, specifying the collaboration responsibility, goal, precondition, postcondition and system operations involved in the collaboration. Business process instances (use case instances) are *instances* of collaborations, specifying concrete sequences of actions and events, system states and state transitions during the collaboration.

During the development process, software architects, designers and developers identify certain information about the software product. Examples of such information are the use cases, the software architecture, the object collaborations and the class descriptions. The information can be

[3] In the UML metamodel version 1.1, dependencies do not have role ends. However, the role ends can be specified as tags of dependencies. Please see the UML tips and tricks on my web page for details.

very abstract, such as the vision of the product, or very concrete, such as the source code. In this paper, I call such pieces of information about the software product *design artifacts.*

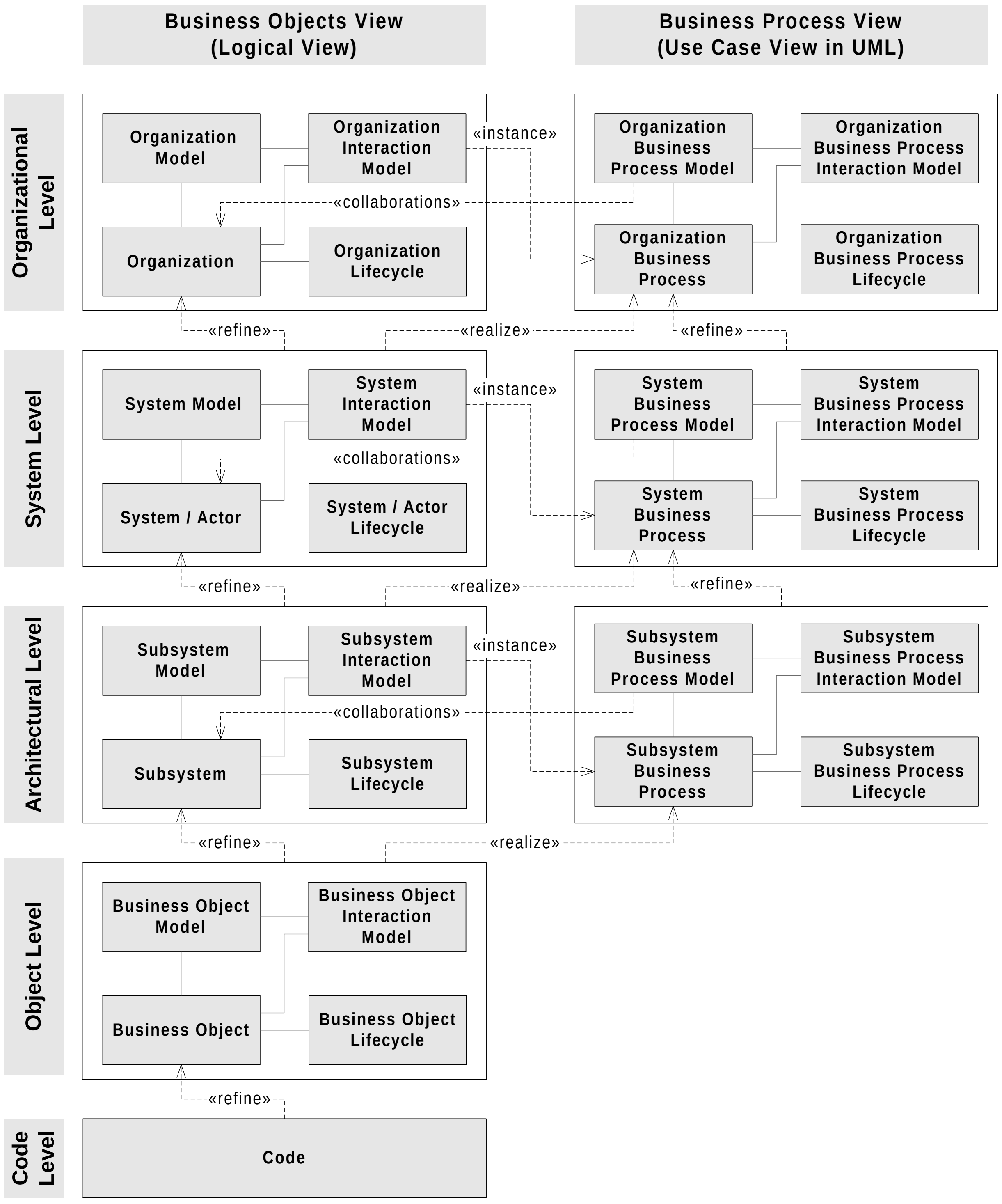
We must realize that there is a difference between a design artifact and its representation. The *design artifact* determines the information about the business system, and the *representation* determines how the information is presented. Some design artifacts are represented in UML, some are represented by text or by tables and some are represented in a number of different ways. For example, the class lifecycle can be represented by a UML statechart diagram, an activity diagram, state transition table or in Backus-Naur form. The object interactions can be represented by UML sequence diagrams or by UML collaboration diagrams. The class responsibility is represented by text.

A useful description of a workflow management system is based on precisely defined design artifacts, rather than on diagrams. This section and the following section discuss a structure of the design artifacts, which provides clear tracing of information describing business processes, business rules, a software architecture and a design of the business system. The structure can easily be extended to cover other aspects of the business system, such as the team structure and the project planning.

Fig. 9 shows relationships between design artifacts specifying business processes and logical design of the software system. Artifacts are structured according to the level of abstraction: the organizational level, the system level, the architectural level and the object level. At each level of abstraction and in each view, the business system can be described by four design artifacts: static relationships between classifiers, dynamic interactions between classifiers, classifier responsibilities and classifier lifecycles. UML classifiers are class, object, interface, subsystem, use case, node and component.

The *classifier model* specifies static relationships between classifiers. The *classifier interaction model* specifies interactions between classifiers. The design artifact called *classifier* specifies classifier responsibilities, roles, and static properties of classifier interfaces (for example, a list of classifier operations with preconditions and postconditions). The *classifier lifecycle* specifies dynamic properties of classifier interfaces (for example, the allowable order operations and events) and classifier state machine. Please see (Hruby, 1998) for more information about applications of the structure to other aspects of software design, such as the testing, database design, user interface and user documentation aspects.

The *organizational level* specifies the responsibility of an organization (such as a company, school, or government body) and the business context of the organization. The artifact *organization* specifies responsibility and relevant static properties of the organization. The artifact *organization model* specifies relationships of the organization to other organizations. The artifact *organization business process* specifies the business process with the organizational scope in terms of the process goal, precondition, postcondition, business rules that the process must meet and other relevant static properties of the process. This business process is a collaboration of the organization with other organizations. All collaborations of the organization with other organizations are described in the artifact *organization business process model,* see the dependency «collaborations» in Fig. 9. The instances of organization business processes are specified in the artifact *organization interaction model* in terms of the interactions of the organization with other organizations. The organization business processes can be refined into more concrete system business processes, see the dependency «refine» in Fig. 9. Allowable order of the system business processes is specified in the artifact *organization business process life cycle*. The *organization business process interaction model* specifies typical sequences of business process instances, see the dependency «instance» in Fig. 9. The realization of the organizational business process is specified by the interactions between the software system and its users (team roles) see the dependency «realize» in Figs. 9 and 10.



Business Objects View (Logical View)
Business Process View (Use Case View in UML)
Organizational Level
System Level
Architectural Level
Object Level
Code Level
Organization Model
Organization Interaction Model
Organization
Organization Lifecycle
Organization Business Process Model
Organization Business Process Interaction Model
Organization Business Process
Organization Business Process Lifecycle
«instance»
«collaborations»
«refine»
«realize»
System Model
System Interaction Model
System / Actor
System / Actor Lifecycle
System Business Process Model
System Business Process Interaction Model
System Business Process
System Business Process Lifecycle
Subsystem Model
Subsystem Interaction Model
Subsystem
Subsystem Lifecycle
Subsystem Business Process Model
Subsystem Business Process Interaction Model
Subsystem Business Process
Subsystem Business Process Lifecycle
Business Object Model
Business Object Interaction Model
Business Object
Business Object Lifecycle
Code

Fig. 9 Design artifacts describing the software system in the logical and business process views.

The *system level* specifies the context of the software system and its relationships to its actors. The artifact *system* specifies the system interface, the system operations with responsibilities, preconditions, postconditions, parameters and return values. The artifact *actor* specifies the actor responsibilities and interfaces, if they are relevant. The *system lifecycle* specifies the allowable order of system operations and events. The *system model* specifies relationships between the software system and actors (other systems or users), and the *system interaction model* specifies interactions between the software system and actors. These interactions are instances of system business processes, see the dependency «instance» in Fig. 9. The artifact *system business process* specifies the business process with the system scope in terms of the process goal, precondition, postcondition, the non-functional requirements of the process, business rules and other relevant static properties. This business process is a collaboration of the system with other systems and users. All collaborations of the system with its actors are described in the artifact *system business process model*, see the dependency «collaborations» in Fig. 9. The dynamic properties of the business process interface, such as the allowable order of system operations in the scope of the business process, are specified in the *system business process life cycle.* The *system business process interaction model* specifies typical sequences of business process instances. The system business processes can be refined into subsystem business processes, see the dependency «refine» in Fig. 9. The realization of the system business process is specified by the subsystems at the architectural level, their responsibilities and interactions, see the dependency «realize» in Fig. 9.

The *architectural level* defines subsystems (components), their responsibilities, interfaces, relationships and interactions. The artifact *subsystem* specifies the subsystem interface, the subsystem operations with responsibilities, preconditions, postconditions, parameters and return values. The *subsystem lifecycle* specifies the allowable order of subsystem operations and events. The *subsystem model* specifies relationships between the subsystem and other subsystems, and the *subsystem interaction model* specifies interactions between the subsystem and other subsystems. These interactions are instances of subsystem business processes, see the dependency «instance» in Fig. 9. The artifact *subsystem business process* specifies the business process with the subsystem scope. This business process is a collaboration of the subsystem with other subsystems, systems and users. All collaborations of the subsystem with is actors are described in the artifact *subsystem business process model*, see the dependency «collaborations» in Fig. 9. The dynamic properties of the subsystem business process interface, such as the allowable order of subsystem operations in the scope of the business process, are specified in the *subsystem business process life cycle.* The *subsystem business process interaction model* specifies typical sequences of business process instances. The realization of the subsystem business process is specified by objects at the class level, their responsibilities and interactions, see the dependency «realize» in Fig. 9.

The *object level* provides details about the design of the subsystem in terms of the business objects, their responsibilities, relationships and interactions. The *business object model* specifies static relationships between business objects. The *business object interaction model* specifies the design of subsystem operations in terms of interactions between business objects. The artifact *business object* specifies the business object responsibility and the static properties of business object interfaces, such as the interface operations with preconditions and postconditions. The *business object lifecycle* specifies the allowable order of interface operations.

Fig. 10 shows design artifacts describing the team structure of the organization. In fact there is no significant difference between the structure of artifacts describing a software subsystem and the structure of artifacts describing a team. Roles of team members can be shown as stereotyped classes in UML. The artifact *role* specifies responsibilities of the worker role and other relevant static properties of the role. The artifact *role lifecycle* specifies dynamic properties of roles, their states and to which events they respond. The artifact *role model* specifies static relationships between roles. The *member level business process* specifies collaborations of the team member role with other team member roles, see the dependency «collaborations» in Fig. 10. The instances of these business processes are specified in the artifact *role interaction model* in terms of interactions between role instances, see the dependency «instance» in Fig. 10.

The design artifact called *team* is a package of roles. The *team* specifies the responsibility of the team and relevant static properties of the team. Dynamic properties of the team are specified in the artifact *team life cycle*. The artifact *team model* specifies static relationships between teams. The *team level business process* specifies collaborations of the team with other teams, see the dependency «collaborations» in Fig. 10. Instances of the team level business processes are specified in the artifact *team interaction model,* see the dependency «instance» in Fig. 10. The realizations of the team level business processes are specified by interactions of the team members and by interactions of the team members with the software system, see the dependency «realize» in Fig. 10. The pattern of design artifacts can be applied at the higher levels of abstraction[4] in a similar way to that shown above.

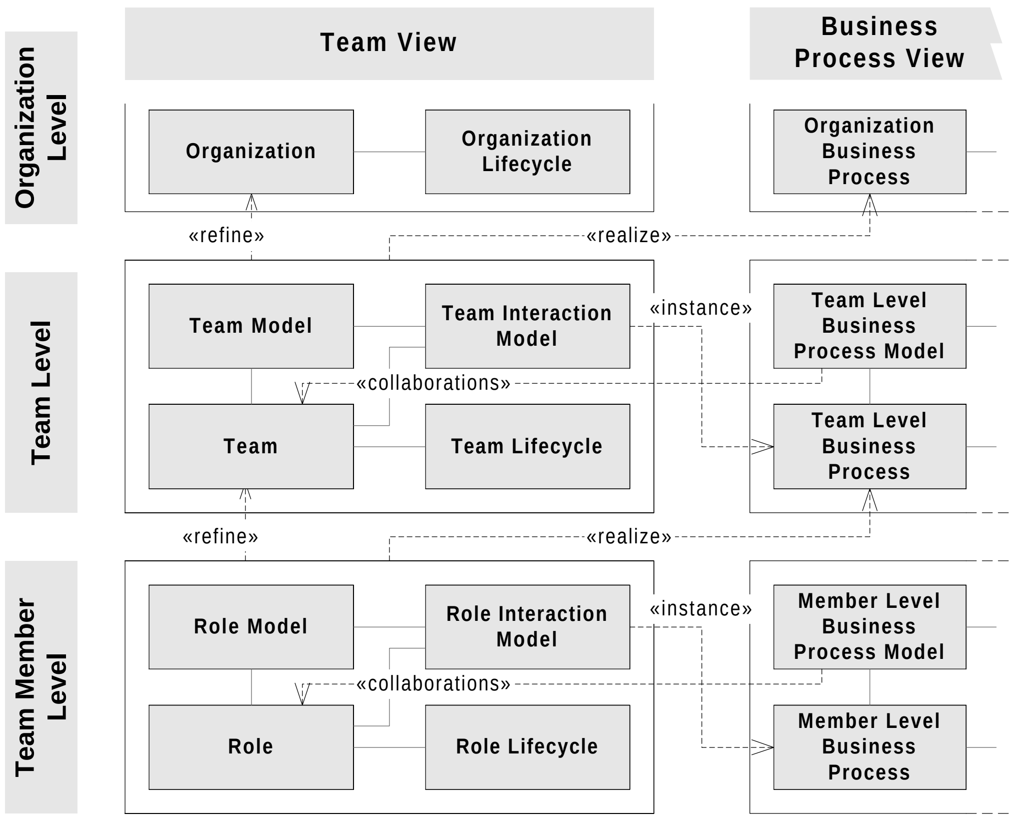


Fig. 10 Design artifacts describing the software system in the team and business process views.

[4] Organizations with deep organizational structure can have a large number of levels, such as the team member level, the section level, the department level, the division level, the company level and the corporation level. Small organizations or the organizations with a flat structure might be sufficiently described by three levels as shown in Fig. 10: the team member level, the team level and the organization level.

The structure of design artifacts described in this section can be simplified in two ways:
- by using only a certain subset of design artifacts.
- by creating larger design deliverables by joining closely related design artifacts.

Using a subset of design artifacts may not lead to the loss of information because the UML system of diagrams is not orthogonal. This means that the same information can be specified in two or more different UML diagrams. For example, both the static structure diagram and the object collaboration diagram specify relationships between objects. Both statecharts and interaction diagrams specify messages between objects. Because the same information can be specified in several places, developers can produce only a certain subset of design artifacts, or the artifacts must be checked for consistency.

For pragmatic reasons, designers can create larger deliverables containing several closely related artifacts. For example, classifier responsibilities and lifecycles are always related to and can be joined into one document. It is also possible to combine organizational, system, subsystem and class use case diagrams into one large use case diagram, providing that use case levels are clearly distinguished. Similarly, system, subsystem and business object static structure diagrams and corresponding interaction diagrams at all levels can be joined into one document. It is also compatible with UML semantics to join the static structure diagrams, the component, node and use case diagrams within each level. In this way designers can show static relationships between use cases, actors, subsystems, business objects, team members and business processes in one large static structure diagram, although the UML notation guide does not explicitly mention such a combined static structure diagram. Please see (Hruby, 1998) for details on how to simplify the structure of design artifacts described in this section.

4. Summary

This paper discussed the representation of business systems in UML, with an emphasis on workflow management systems. The paper described mapping between typical business concepts and UML concepts and suggested the structure of design artifacts for tracing the information between definitions of business processes and the object-oriented software design. The structure assumes that business processes can be considered as collaborations between business objects, team roles and other instances in a business system. The structure is based on a pattern of four mutually related design artifacts that represent classifier relationships, interactions, responsibilities and lifecycles. The pattern was applied at different levels of abstraction and in different views on a business system. The pattern can specify various other aspects of the design of business and software systems.

References

Alexander, I, A.: *A Co-Operative Task Modeling Approach to Business Process Understanding*, workshop on Object-Oriented Business Process Modeling, ECOOP 98, Brussels, Belgium, July 20-24, 1998.
Henderson-Sellers, B, Graham, I., *The OPEN Modeling Language (OML) Reference Manual*, SIGS Books, NY, 1997, available at http://www.csse.swin.edu.au/cotar/OPEN/OPEN.html
Hruby, P.: *Structuring Design Deliverables with UML*, <<UML>>'98, Mulhouse, France, June 3-4 1998, available at http://www.navision.com/default.asp?url=services/methodology/default.asp
Hruby, P.: *A Pattern for Structuring UML-based Repositories*, OOPSLA'98, Vancouver, Canada, October 18-24, 1998.

Martin, J., Odell, J. J.: *Object-Oriented Methods: a Foundation*, Prentice Hall, Inc., 1998
Martin, R. C.: *RE: RE: (OTUG) use cases: abstract and concrete (<<precedes>>)*, Mailing list OTUG, June 1997, at http://www.rational.com/HyperMail/otug/hypermail/9807/0195.html
UML Notation Guide, version 1.1, Rational, 1 September 1997, at http://www.rational.com/uml
Stark, H., Lachal, L.: Ovum Evaluates Workflow, Ovum Ltd. 1995.