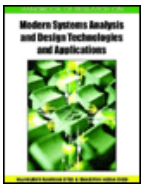
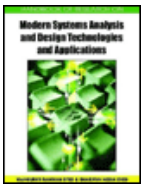


Chapter III: Meta-Modeling for Situational Analysis and Design Methods

Inge van de Weerd,
Utrecht University,
The Netherlands
Sjaak Brinkkemper,
Utrecht University,
The Netherlands

ABSTRACT

This chapter introduces an assembly-based method engineering approach for constructing situational analysis and design methods. The approach is supported by a meta-modeling technique, based on UML activity and class diagrams. Both the method engineering approach and meta-modeling technique will be explained and illustrated by case studies. The first case study describes the use of the meta-modeling technique in the analysis of method evolution. The next case study describes the use of situational method engineering, supported by the proposed meta-modeling technique, in method construction. With this research, the authors hope to provide researchers in the information system development domain with a useful approach for analyzing, constructing, and adapting methods.



Chapter III - Meta-Modeling for Situational Analysis and Design Methods
Handbook of Research on Modern Systems Analysis and Design Technologies and Applications
by Mahbubur Rahman Syed and Sharifun Nessa Syed (eds)
IGI Global © 2009 *Citation*

INTRODUCTION

Many methods for developing information systems (IS) exist. The complexity of the projects in which they are used varies, as well as the situational factors that influence these projects. Over the years, more methods will be developed, as the technology will continue to diversify and new ISs are being developed. However, often methods are too general and not fitted to the project at hand. A solution to this problem is situational method engineering to construct optimized methods for every systems analysis and design situation, by reusing parts, the so-called method fragments, of existing established methods.

In this chapter, an overview of current method engineering research is given. A general approach on situational method engineering is described, as well as a meta-modeling technique, which supports the process of situational method engineering. The technique will be illustrated in two examples. Finally, we describe our future research and conclusions.

BACKGROUND

No IS development method exists that is best in all situations. Therefore, to improve the effectiveness of a method, it should be engineered to the situation at hand, by taking into account the uniqueness of a project situation (Kumar & Welke, 1992). This is defined as *"method engineering:"* "the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems" (Brinkkemper, 1996).

A special type of method engineering is situational method engineering. The term *situational method* is defined as "an information systems development method tuned to the situation of the project at hand" (Harmsen, Brinkkemper, & Oei, 1994). Situational method engineering is often used in combination with route maps, high-level method scenario's, which can be used to tune the method into situational methods (Van Slooten & Hodes, 1996). Different routes are used to represent the different situations: new IS development, COTS (commercial of the shelf) tool selection, re-engineering, and so forth.

Several situational method engineering approaches have been described in literature, by, for example, Brinkkemper (1996); Saeki (2003); Ralyté, Deneckère, and Rolland (2003); and Weerd, Brinkkemper, Souer, and Versendaal (2006). To execute the method engineering process, methods need to be described for which several modeling techniques have been proposed. Saeki (2003), for example, proposed the use of a meta-modeling technique, based on UML activity diagrams and class diagrams, for the purpose of attaching semantic information to artifacts and for measuring their quality using this information. In Rolland, Prakash, and Benjamin (1999) and Ralyté et al. (2003), a strategic process meta-model called Map is used to represent process models.

In all research on situational method engineering, several steps are followed in the process to develop a situational method. By comparing the different approaches, we could distinguish the following generic steps in a situational method engineering approach (Weerd et al, 2006):

- Analyze project situation and identify needs;
- Select candidate methods that meet one or more aspects of the identified needs;
- Analyze candidate methods and store relevant method fragments in a method base; and
- Select useful method fragments and assemble them in a situational method by using route map configuration to obtain situational methods.

The third and fourth steps are supported by a meta-modeling technique, especially developed for method engineering purposes. This technique, in which a so-called *process-deliverable diagram* (PDD) is built, is used in analyzing, storing, selecting, and assembling the method fragments. The meta-modeling technique is adopted from Saeki (2003), who proposed the use of a meta-modeling technique for the purpose of attaching semantic information to the artifacts and for measuring their quality using this information. In this research, the technique is used to reveal the relations between activities (the process of the method) and concepts (the deliverables produced in the process). We will elaborate on this in the next section.

PROCESS-DELIVERABLE DIAGRAMS

This section describes the technique used for modeling activities and artifacts of a certain process. As we are modeling methods and not the artifacts of an IS, this type of modeling is called meta-modeling. We express the meta-models of method in PDDs, which consist of two integrated diagrams. The process view on the left-hand side of the diagram is based on a UML activity diagram (OMG, 2003) and the deliverable view on the right-hand side of the diagram is based on a UML class diagram (OMG, 2003). In this chapter, first the left-hand side of the diagram is explained, then the right-hand side, and finally the integration of both diagrams.

The meta-modeling technique is explained, where notational explanation will be illustrated by an example in practice. Those examples are taken from a project to create a situational method for implementing Web-applications with a content management system (Weerd et al., 2006).

Meta-Process Modeling

Meta-process modeling is done by adapting the UML *activity diagram.* According to Booch, Rum-baugh, and Jacobson (1999), an activity diagram is "a diagram that shows the flow from activity to activity; activity diagrams address the dynamic view of a system." This diagram consists of activities and transitions. Activities can be decomposed into sub-activities, if necessary, and thereby creating a hierarchical activity decomposition. Transitions can be used to show the control flow from one activity to the next. A simple arrow depicts this. Four types of transitions exist: unordered, sequential, concurrent, and conditional activities.

Activity Types

We use different types of activities for several reasons. Firstly, activity types are used for scope definition. Itmay be interesting to know which processes exist adjacent to the process that is focused on. The details of this process are not interesting and are therefore not shown. Secondly, for purposes of clarity of the diagram, some processes are expanded elsewhere. Finally, in some cases the sub activities are not known or not relevant in the specific context.

The following activity types are specified (see Figure 1):

- **Standard activity:** An activity that contains no further (sub) activities. A standard activity is illustrated with a rounded rectangle.
- **Complex activity:** An activity that consists of a collection of (sub) activities. They are divided into:
  - **Open activity:** A complex activity whose (sub) activities are expanded. This expansion can be done in the same diagram or in another diagram. Therefore, we use two notational variants, which are:
    - A rounded rectangle, containing two ore more sub activities and
    - A rounded rectangle with a white shadow, to indicate that the activities are depicted elsewhere.

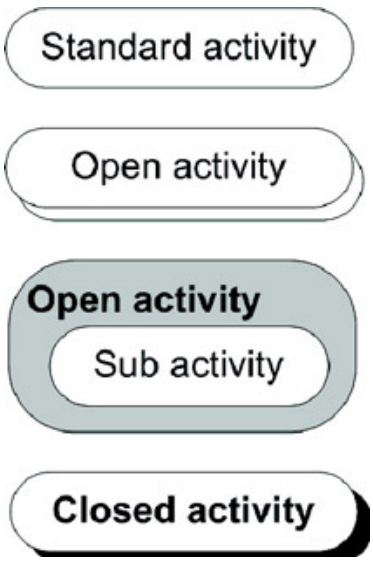


Figure 1: Activities Types

- **Closed activity:** A complex activity whose (sub) activities are not expanded since it is not known or not relevant in the specific context.

In Figure 2 we give some examples of activity types. 'Listfeatures' is a standard activity that has no further sub activities. In case of 'describe candidate requirements,' the activity is open, since we find it interesting to describe its sub activities. 'Define use case' model is closed, since we are not interested in the details of this process. The naming conventions for activity names are that the names always consist of a verb and an object, except when the activity name is literally copied from a method source, such as a book. Activities for the highest level are excluded form this naming convention, since they usually get the name of a stage or phase.

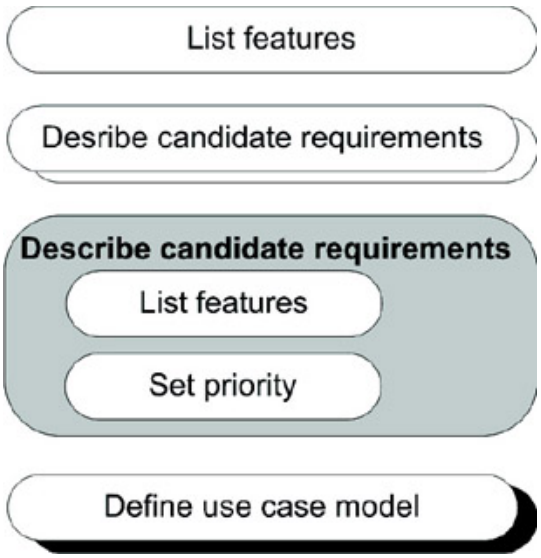


Figure 2: Example activity types

In the following sections, the four activity transitions will be introduced and exemplified.

Sequential Activities

Sequential activities are activities that need to be carried out in *a. pre defined* order. The activities are connected with an arrow, implying that they have to be followed in that sequence, although the activity completion before the next can be started is *not* strictly enforced. Both activities and sub-activities can be modeled in a sequential way. In Figure 3 an activity diagram is illustrated with one activity and two sequential sub-activities. A special kind of sequential activities are the start and stop states, which are also illustrated in Figure 3.

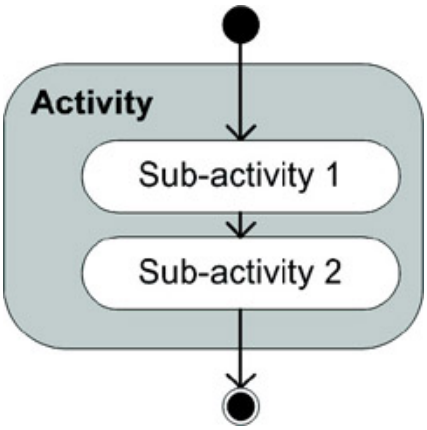


Figure 3: Sequential activities

In Figure 4 an example is illustrated. The example is taken from the requirements capturing workflow, of which the main activity, 'model users & domain,' consists of three activities that need to be carried out in a predefined order.

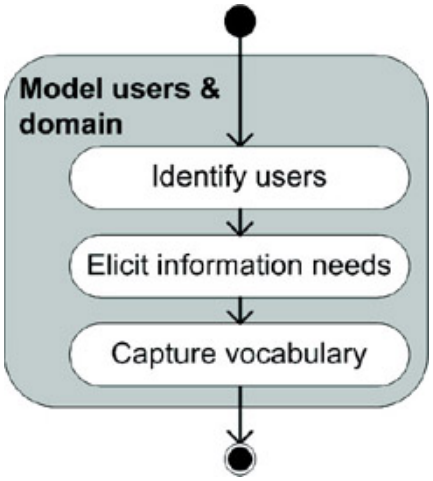


Figure 4: Example equential activities

Unordered Activities

Unordered activities are used when sub-activities of an activity can be executed in any order, that is, they do not have a predefined execution sequence. Only sub-activities can be unordered. Unordered activities are represented as sub-activities without transitions within an activity, as is shown in Figure 5.

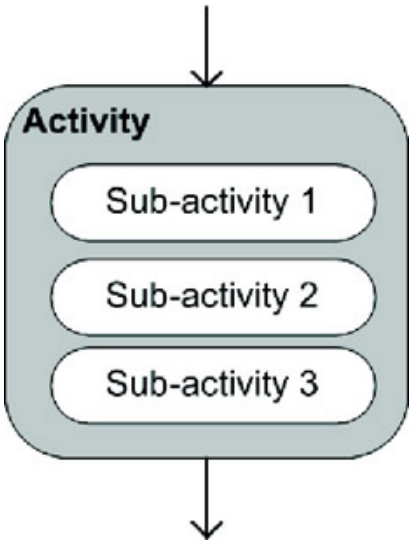


Figure 5: Unordered ctivities

An activity may consist of sequential and unordered activities, which is modeled by dividing the main activity in different parts. In Figure 6 an example is taken from the requirements analysis workflow. The main activity, 'describe candidate requirements,' is dividedintotwoparts. Thefirstpart is a sequential activity. The second part consists of four activities that do not need any sequence in order to be carried out correctly. Note that all unordered activities must be carried out before continuing to the next activity.

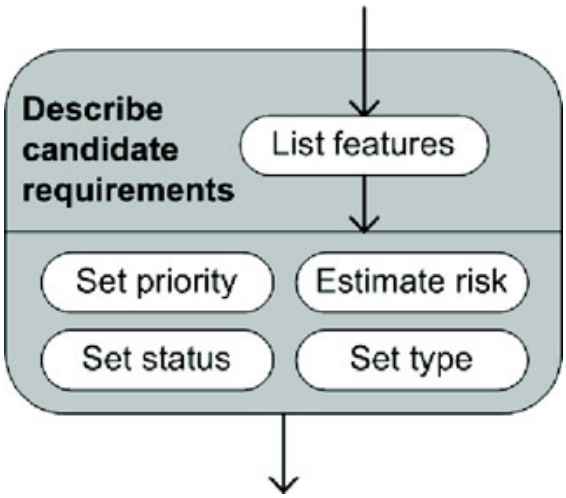


Figure 6: Example of unordered activities

Concurrent Activities

Activities can occur concurrently. This is handled with forking and joining. By drawing the activities parallelin the diagram, connected witha synchronization bar, one can fork several activities. Later on these concurrent activities can join again by using the same synchronization bar. Both the concurrent execution of activities and sub-activities can be modeled. In the example of Figure 7, Activity 2 and Activity 3 are concurrent activities.

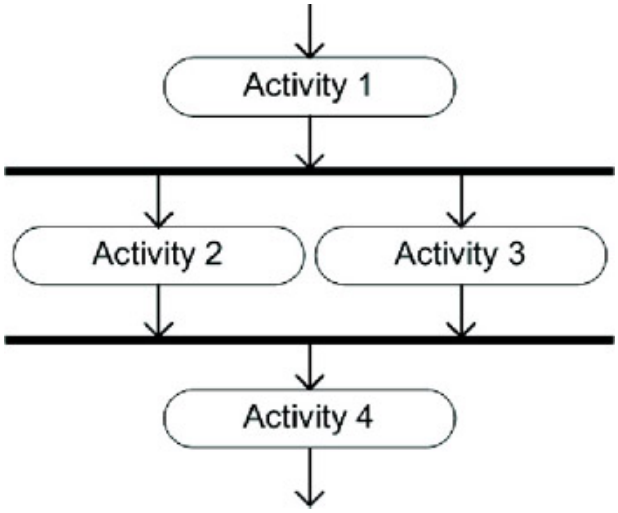


Figure 7: Concurrent activities

In Figure 8 a fragment of the requirements capturing process is depicted. Two activities, ‘define actors’ and ‘defining use cases’ are carried out concurrently. The reasonfor carrying outthese activities concurrently is that definition of actors and of use cases influence each other to a high extend.

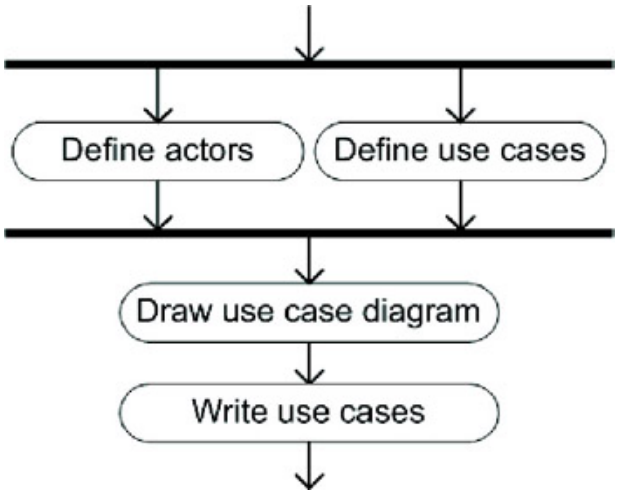


Figure 8: Example concurrent activities

Conditional Activities

Conditional activities are activities that are only carried out if a predefined condition is met. This is graphically repre sented by using abranch. Branche s are denoted with a diamond and can have incoming and outgoing transitions. Every outgoing transition has a guard expression, the condition, denoted with square bracket ‘[ ]’. This guard expression is actually a Boolean expression, used to make a choice which direction to go. Both activities and sub-activities can be modeled as conditional activities. In Figure 9, two conditional activities are illustrated.

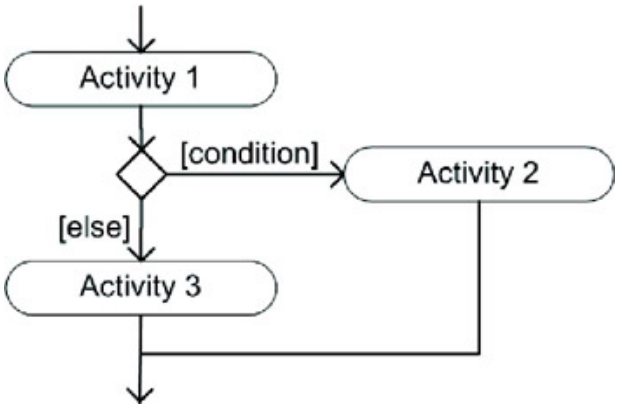


Figure 9: Conditional activities

In Figure 10, an example from a requirements analysis starts with studying the material. Based on this study, the decision is taken whether to do an extensive requirements elicitation session or not. The condition for not carrying outthis requirements session is represented at the left of the branch, namely [requirements clear]. If this condition is not met, [else], the other arrow is followed.

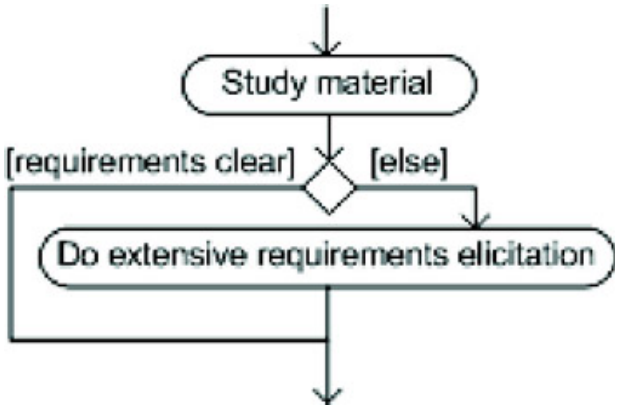


Figure 10: Example conditional activities

Roles

In some methods it is explicitly stated by which individuals or organizational role the process is to be carried out. In this case, the role is indicated in the activity. In Figure 11, an activity with three sub activities is depicted. In the lower right corner, the role is positioned. Please note that the usage of roles is not restricted to activities, but, if necessary, can also be used in sub-activities.

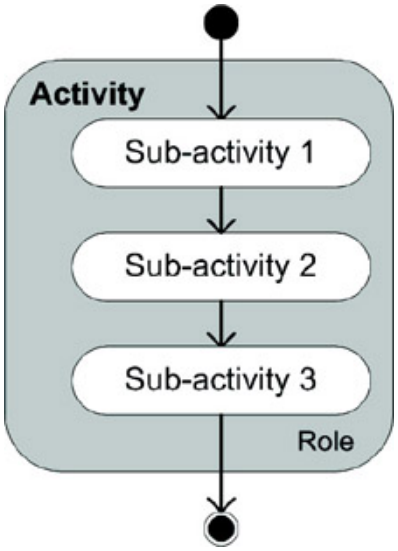


Figure 11: Roles

In Figure 12, the usage of roles is illustrated. The same example as in Figure 4 is used with the difference that the role, in this case ‘consultant,’ is indicated.

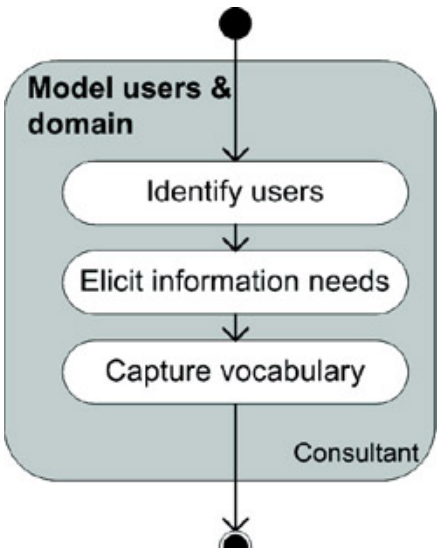


Figure 12: Example role

Meta-Deliverable Modeling

The deliverable side of the diagram consists of a *concept diagram.* This is basically an adjusted class diagram as described Booch et al. (1999). Important notions are concept, generalization, association, multiplicity and aggregation.

Concept Types

A concept is a simple version of a UML class. The class definition of Booch et al. (1999) is adopted to define a concept, namely: "a set of objects that share the same attributes, operations, relations, and semantics."

The following concept types are specified:

- **STANDARD CONCEPT:** A concept that contains no further concepts. A standard concept is visualized with a rectangle.
- **COMPLEX CONCEPT:** A concept that consists of an aggregate of other concepts. Complex concepts are divided into:
    - **OPEN CONCEPT:** Acomplexconcept whose sub concepts are expanded. An open concept is visualized with a white shadow border. The aggregate structure may be shown in the same diagram or in a separate diagram.
    - **CLOSED CONCEPT:** A complex concept whose sub concepts are not expanded since it is not relevant in the specific context. A closed concept is visualized with a black shadow border.

Similar as for activities, this usage of different concept types is introduced for clarity and scope definition.

In Figure 13, the three concepttypes that are used in the modeling technique are illustrated. Concepts are singular nouns and are always capitalized, not only in the diagram, but also when referring to them in the textual descriptions outside the diagram.

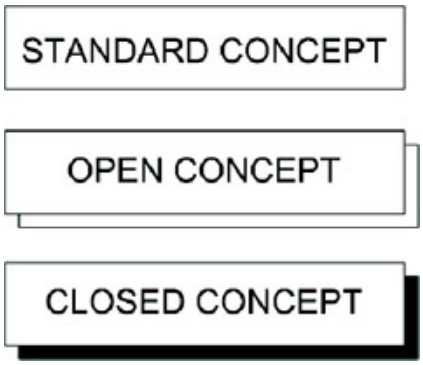


Figure 13: Standard, open and closed concepts

In Figure 14, all three concepttypes are exemplified. Part of the PDD of a requirements workflow is illustrated. The USE CASE MODEL is an open concept and consists of one or more ACTORS and one or more USE CASES. ACTOR is a standard concept, it contains no further sub-concepts. USE CASE, however, is a closed concept. A USE CASE consists of a description, a flow of events, conditions, special requirements, and so forth. Because in this case we decided it is unnecessary to reveal that information, the USE CASE is illustrated with a closed concept.

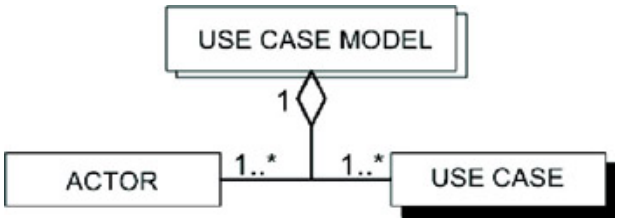


Figure 14: Example of standard, open and closed concepts

Generalization

Generalization is a way to express a relationship between a general concept and a more specific concept. Also, if necessary, one can indicate whether the groups of concepts that are identified are overlapping or disjoint, complete, or incomplete. Generalization is visualized by a solid arrow with an open arrowhead, pointing to the parent, as is illustrated in Figure 15.

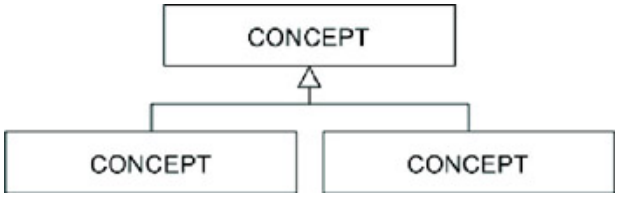


Figure 15: Generalization

In Figure 16, generalization is exemplified by showing the relationships between the different concepts described in the preceding paragraph. CONTROL FLOW and DATA FLOW are both a specific kind of FLOW. Also note the use of the *disjoint* (d) identifier. This implies that occurrence of one concept is incompatible with the occurrence of the other concept; that is, there are no CONTROL FLOWS that are also DATA FLOWS and vice versa. The second identifier we use is *overlapping* (o), which would indicate in this example that there may be occurrences that are CONTROL FLOWS or DATA FLOWS, but also occurrences that are both. Finally, *categories* (c) mean that the disjoint concepts have no occurrences in common and that the decomposition is exhaustive. In the example, this would imply that every occurrence of flow is either a CONTROL FLOW or DATA FLOW. No other FLOWS exist, and there are no occurrence s that are both CONTROL FLOW and DATA FLOW.

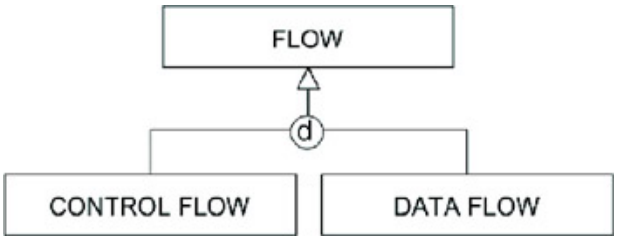


Figure 16: Example generalization

Association

An association is a structural relationship that specifies how concepts are connected to another. It can connect two concepts (binary association) or more than two concepts (n-ary association). An association is represented with an undirected solid line. To give a meaning to the association, a name and name direction can be provided. The name is in the form of an active verb and the name direction is representedby atrianglethatpoints in the direction one needs to read. Association with a name and name direction is visualized in Figure 17.

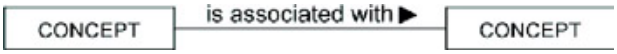


Figure 17: Association

In Figure 18, an example of a fragment is shown of the PDD of the requirements analysis in the Unified Process (Jacobson, Booch, & Rumbaugh, 1999). Because both concepts are not expanded any further, although several sub concepts exist, the concepts are illustrated as closed concepts. The figure reads as "SURVEY DESCRIPTION describes USE CASE MODEL."

Figure 18: Example association

Multiplicity

Except name and name direction, an association has another characteristic. With *multiplicity,* one can state how many objects of a certain concept can be connected across an instance of an association. Multiplicity is visualized by using the following expressions: *1* for exactly one, *0…1* for one or zero, *0…** for zero or more, 7…* for one or more, or for example, *5* for an exact number. In Figure 19, association with multiplicity is visualized.

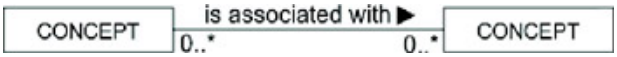


Figure 19: Multiplicity

An example of multiplicity is represented in Figure 20. It is the same example as in Figure 18, only the multiplicity values are added. The figure reads as "exactly one SURVEY DESCRIPTION describes exactly one USE CASE MODEL." This implies inanISprojectthataSURVEY DESCRIPTION will always be relatedtojustone USE CASE MODEL and the other way round.

Figure 20: Example multiplicity

Aggregation

A special type of association is *aggregation.* Aggregation represents the relation between a concept (as a whole) containing other concepts (as parts). It can also be described as a 'has-a' or 'consists of relationship. In Figure 21, an aggregation relationship between OPEN CONCEPT and STAN DARD CONCEPT is illustrated. An OPEN CONCEPT consists of oneormore STANDARD CONCEPTS and a STANDARD CONCEPT is part of one OPEN CONCEPT.

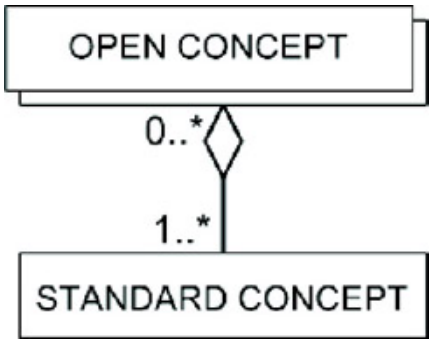


Figure 21: Aggregation

In Figure 22, aggregation is exemplified by a fragment of a requirements capture workflow. A USE CASE MODEL consists of one or more ACTORS and USE CASES.

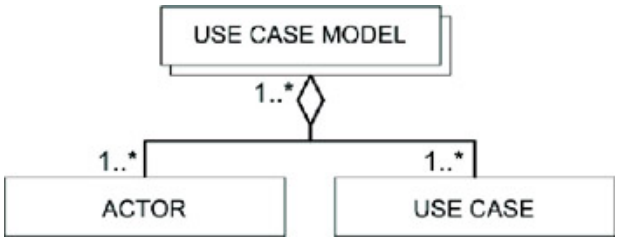


Figure 22: Example aggregation

Properties

Sometimes the need exists to assign properties to CONCEPTS. Properties are written in lower case, under the CONCEPT name, as is illustrated in Figure 23.

Figure 23: Properties

In Figure 24, an example of a CONCEPT with properties is visualized. DESIGN has seven properties, respectively: code, status, author, effective date, version, location, and application.

Figure 24: Example properties

Process-Deliverable Diagram

The integration of both types of diagrams is quite straightforward. Activities are connected with a dotted arrow to the produced deliverables, as is demonstrated in Figure 25. Note that in UML class diagrams, dotted arrows are used to indicate dependency from one class on another. However, this construction is not used in PDDs. We have some extra remarks concerning the PDD. Firstly, all activities in Figure 25 result in deliverables. However, this is not compulsory. Secondly, it is possible for several activities to point to the same deliverable, see for example, sub-activity 2 and 3. Finally, we assume that all artifacts in the diagram are available to all roles.

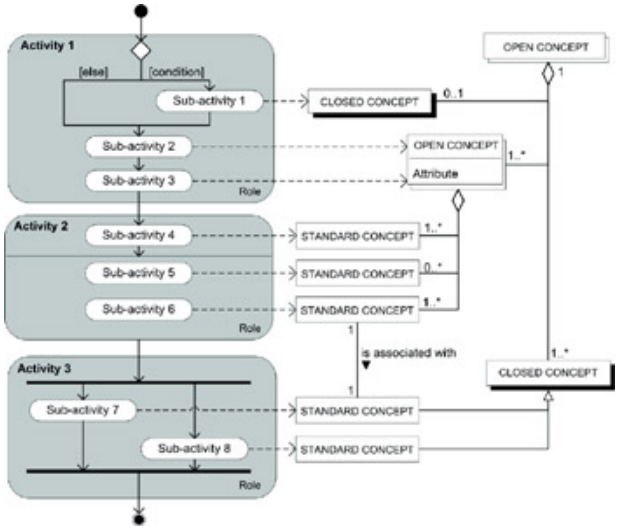


Figure 25: Process-deliverable diagram

Example of a Process-Deliverable Diagram

In Figure 26, an example of a PDD is illustrated. It concerns an example of drawing an object chart, as described in Brinkkemper, Saeki, and Harmsen (1999). This is an example of method assembly, in which an object model and Harel's state chart are integrated into object charts.

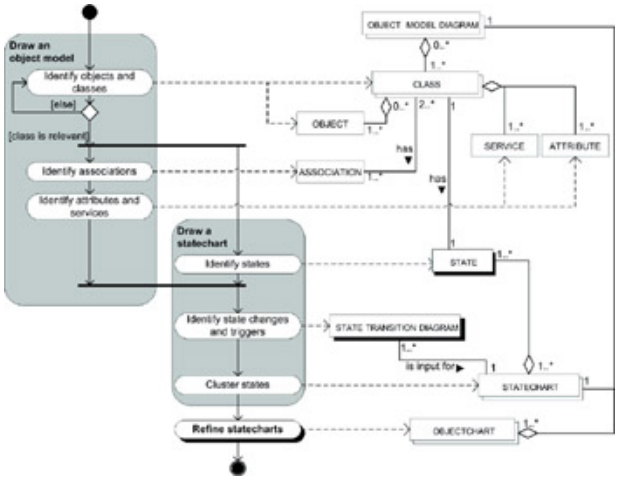


Figure 26: Example PDD—drawing an object chart

Notable is the use of an open concept: STATE TRANSITION DIAGRAM. This concept is open, since it is a complex concept, and should be expanded elsewhere. However, due to space limitations, this is omitted. The activities of ‘drawing an object chart’ are carried out by one person. Therefore, no roles are depicted in the diagram.

In Table 1, the activities and sub-activities, as well as their relations to the deliverables, which are depicted in the diagram, are described.

Table 1: Activities and sub-activities in drawing an object chart

Open table as spreadsheet

Activity	Sub-Activity	Description
Draw an object	Identify objects and classes	Key phenomena that require data storage and manipulation in the IS are identified as OBJECTS. OBJECTS that share the same attributes and services are identified as a

model		CLASS.
	Identify associations	Relationships between CLASSES are identified when instance OBJECTS of different CLASSES are to be related to each other in the IS.
	Identify attributes and services	For each CLASS, the descriptive ATTRIBUTES and the operational SERVICES are identified.
Draw a state chart	Identify states	Lifecycle statuses of OBJECTS that serve a process in the IS are to be identified as STATES. Similarly for CLASSES.
	Identify state changes and triggers	Events that trigger changes of a STATE and the conditions under which they occur are determined.
	Cluster states	STATES that are sub statuses of a higher level STATE are groups together in a cluster. Furthermore, state transitions between STATES are depicted resulting in a STATECHART.
Refine state charts		Based on the OBJECT MODEL DIAGRAM and the STATECHART, an OBJECTCHART is created by linking STATECHARTS to each CLASS.

In Table 2, an excerpt if aconcepttable is shown. Every concept is provided with a description and a reference.

Table 2: Excerpt of a concept table

Open table as spreadsheet

Concept	Description
ASSOCIATION	An association specifies a semantic relationship that can occur between typed instances. It has at least two ends represented by properties, each of which is connected to the type of the end (OMG, 2004).
STATE	A state models a situation during which some (usually implicit) invariant condition holds (OMG, 2004).
OBJECTCHART	An extension of a State chart to model reactive systems from an object-oriented view (Brinkkemper, Saeki, & Harmsen, 1999).
…	

The process in Figure 26, where object model diagrams and state charts are amalgamated into object charts, is an example of situational method engineering for a project situation where is was required to model state charts for each class. Similarly, situational factors may require various kinds of adaptations of the method. Different analysis and design situations in a project give rise to a huge variety of method adaptations: in a simple project just one class diagram may be sufficient to analyze the object domain, whereas in a more complex project, various class diagrams, object models and state charts are required. To support the adaptations, PDDs are very instrumental as both modifications of activities as of concepts can be easily documented.

META-MODELING FOR METHOD EVOLUTION ANALYSIS

Introduction

PDDs canbeusedto analyze the method evolution of a company over the years. In Weerd, Brinkkemper, and Versendaal (2007), general method increments were deducted from literature and case studies. The resulting list of general method increments was then tested in a case study at Infor Global Solutions (specifically the former Baan company business unit), a vendor ofERP (enterprise resource planning) software. The time period that is covered in the case study ranges from 1994 to 2006. We analyzed 13 snapshots of the evolution of the software development process at Baan, with emphasis on product management activities. An overview of these method increments is listed in Table 3.

Table 3: Overview of method increments at Baan
Open table as spreadsheet

#	Increment	Date
0	Introduction requirements document	1994
1	Introduction design document	1996
2	Introduction version definition	May, 1998
3	Introduction conceptual solution	November, 1998,
4	Introduction requirements database, division market and business requirements, and introduction of product families	May, 1999
5	Introduction tracing sheet	July, 1999
6	Introduction product definition	March, 2000
7	Introduction customer commitment process	April, 2000
8	Introduction enhancement request process	May, 2000
9	Introduction roadmap process	September, 2000
10	Introduction process metrics	August, 2002
11	Removal of product families & customer commitment	May, 2003
12	Introduction customer voting process	November, 2004
13	Introduction master planning	October, 2006

In the next sections, we will illustrate two of these increments, namely increment #3 and increment #4.

Snapshot of Increment #3

In Figure 27, increment # 3 of the requirements management and release definition process a Baan is illustrated.

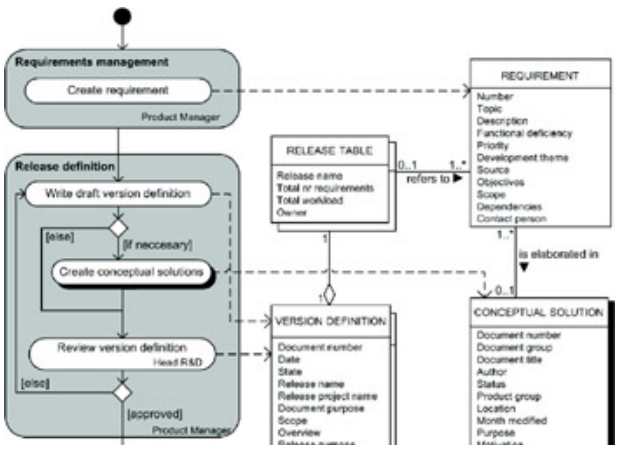


Figure 27: Snapshot of increment #3

We can distinguish two main activities in the figure, namely ‘requirements management’ and ‘release definition.’ The first main activity consists of one sub activity, namely ‘create requirement,’ in which the REQUIREMENTS are created by the product manager. The other main activity consists of three sub activities. Firstly, the product manager writes a first draft of the VERSION DEFINITION. Secondly, the product manager writes, if necessary, a CONCEPTUAL SOLUTION per REQUIREMENT Finally, the VERSION DEFINITION is reviewed by the head of the research & development department. If he approves the VERSION DEFINITION, they can continue with the next activity. If he does not approve it, the product manager has to rewrite the VERSION DEFINITION. A more elaborated description of the activities and the concepts is provided in Table 4 and Table 5.

Table 4: Activity table of increment #3

Open table as spreadsheet

Activity	Sub activity	Description
Requirements management	Create requirement	The product manager adds the REqUIREMENT and its properties to the requirements Excel sheet.
Release definition	Write draft version definition	The product manager writes a draft of the VERSION DEFINITION.
	Create conceptual solution	If necessary, the product manager elaborates on the requirements in a CONCEPTUAL SOLUTION.
	Review version definition	The head of the research & development department reviews the RELEASE DEFINITION. Either he approves it, in which case the process continues in a new activity, or he disapproves it, in which case the product manager rewrites the VERSION DEFINITION.

Table 5: Concept table of increment #3

Open table as spreadsheet

Concept	Description
REQUIREMENT	A REQUIREMENT is a functional description of a new functionality that is to be implemented in the new release of a product.
RELEASE TABLE	The RELEASE TABLE is part of the VERSION DEFINITION, and lists the references to the REQUIREMENTS that are implemented in the new release.
VERSION DEFINITION	A document with the listing of REQUIREMENTS of the new release along with the needed personnel resources. (Natt och Dag, Regnell, Gervasi, & Brinkkemper, 2005)
CONCEPTUAL SOLUTION	A document with a sketch of the business solution for one preferred or more REQUIREMENTS. (Natt och Dag et al., 2005)

Note that the activity ‘create conceptual solutions’ and the concept CONCEPTUAL SOLUTION are both closed. This implies that they are both complex, that is, consisting of sub activities or concepts, respectively. In this work, it is not relevant to elaborate further on both elements. However, they are specified in Weerd et al. (2006).

Requirements Management and Release Definition Increment

In Figure 28, the snapshot of method increment #4 is depicted. Again, the snapshot consists of two main activities: ‘requirements management’ and ‘release definition.’ The first main activity now consists of three unordered sub activities: the product manager creates MARKET REQUIREMENTS, releases independent BUSINESS REQUIREMENTS, and he maintains the product line.

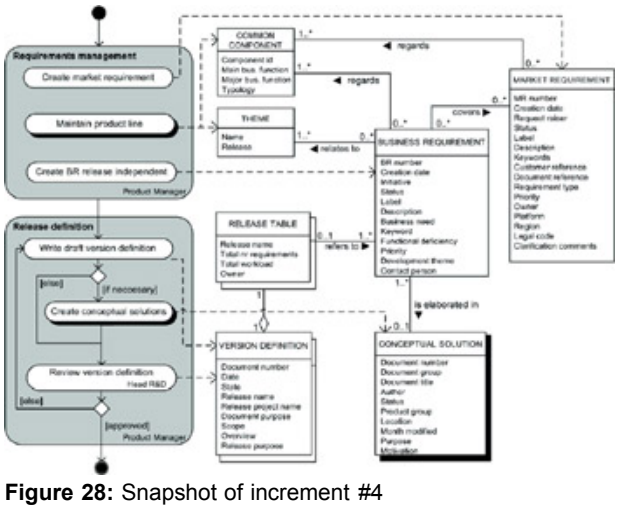


Figure 28: Snapshot of increment #4

Notable in this part of the snapshot is the division between MARKET REQUIREMENTS and BUSINESS REQUIREMENTS. Natt och Dag et al. (2005) give a good elaboration of this separation. They state that MARKET REQUIREMENTS are expressions of

the perceived market need, written in the form of a customer's wish. They change frequently, according to the market need changes, BUSINESS REQUIREMENTS, on the other hand, are written from the company's perspective. These requirements are those that find the company worthwhile to implement one of the following product releases. Both types of requirements are important, MARKET REQUIREMENTS for communicating with the customer and tracing which customer wishes actually get implemented in the product, PRODUCT REQUIREMENTS are important as a basis for release planning, for conducting feasibility studies, and for a functional description of the feature to be implemented. The separation of the two types ensures a solid basis for decision-making about future releases, as well as a clear tracing device which improves the communication with the customer (Natt och Dag et al, 2005).

The other change in the 'requirements management' activity is the insertion of an activity called 'maintain product line.' This activity is not further elaborated in this work, but globally, the product manager has to ensure that requirements are linked to themes and common components, in order to maintain a unambiguous product line. More information can be found in Weerd et al. (2006).

The rest of the snapshot is the same as the snapshot of increment # 4. Concluding, the contents of the 'requirements management' activity have been changed from one to three sub activities. Secondly, the REQUIREMENT concept has been split up into two new concepts: MARKET REQUIREMENT and BUSINESS REQUIREMENT. Finally, the concepts COMMON COMPONENT and THEME have been added. In Table 6 and Table 7, the activities and concepts are further specified.

Table 6: Activity table of increment #4

Open table as spreadsheet

Activity	Sub activity	Description
Requirements management	Create market requirement	The product manager adds a customer wish to the requirements database.
	Maintain product line	Requirements are structured according the ThEME they relate to and the COMMON COMPONENT that is affected, in order to maintain the product line.
	Create BR release independent	The product managers adds bUSINESS REqUIREMENTS to the requirements database, by linking MARkET REqUIREMENTS that cover the same subject and by describing the requirement from the company's perspective.
Release definition	Write draft version definition	The product manager writes a draft of the VERSION DEFINITION.
	Create conceptual solution	If necessary, the product manager elaborates on the requirements in a CONCEPTUAL SOLUTION.
	Review version definition	The head of the research & development department reviews the RELEASE DEFINITION. Either he approves it, in which case the process continues in a new activity, or he disapproves it, in which case the product manager rewrites the VERSION DEFINITION.

Table 7: Concept table of increment #4

Open table as spreadsheet

concept	Description
MARKET REQUIREMENT	A customer wish related to current or future markets, defined using the perspective and context of the user. (Natt och Dag et al., 00)
BUSINESS REQUIREMENT	A generic customer wish to be covered by a product described in the vendor's perspective and context. (Natt och Dag et al., 00)
RELEASE TABLE	The RELEASE TABLE is part of the VERSION DEFINITION, and lists the references to the REQUIREMENTS that are implemented in the new release.
COMMON COMPONENT	A software component that is shared by multiple products in the product line.
THEME	A predefined release THEME that is set by the board.
VERSION DEFINITION	A document with the listing of BUSINESS REQUIREMENTS of the new release along with the needed personnel resources. (Natt och Dag et al., 00)
CONCEPTUAL SOLUTION	A document with a sketch of the business solution for one preferred or more BUSINESS REQUIREMENTS. (Natt och Dag et al., 00)

META-MODELING FOR METHOD CONSTRUCTION

Introduction

The meta-modeling example described in this section covers the assembly of anew method for implementing CMS-based Web applications. The research was carried out at GX creative online development, a Web technology company in the Netherlands. The company implements Web applications, using GX WebManager, a generic CMS-based Web application tool that enables people without a specific technological background in creating, maintaining, and integrating several dynamic Web sites and portals. In additionto their product, GX also provides a service, which is creating a Web application 'around' the CMS-based Web application. The development of this Web application is currently carried out by a proprietary method. However, the need existed to optimize this method in order to save time and money. Also, the need for a standardized Web application development method exists, which can be used by implementation partners of the company. At time of the research, no methods existed for implementing CMS-based Web applications. Development methods for conventional information systems, as well as for Web applications, do not cover the needs for this particular type of development. Therefore, we developed the GX Web engineering method (WEM).

Assembly-based method engineering We applied an assembly-based situational method engineering approach to develop the new method, consisting of the following steps:

1. Analyze implementation situations and identify needs.

   Three implementation situations were identified, namely standard, complex, and migration situations. In Table 8, example needs are given for the standard and complex implementation situations.

   Table 8: Example implementation situation needs
   Open table as spreadsheet

Situation	Need
Standard & Complex	The method should deliver a requirements document that is understandable for the customer and informative for the stakeholders at GX.
Standard	Standard project often have a small budget. This implies that the amount of time for specifying the requirements is limited. Therefore, the method should make it possible to translate the requirements quickly into Web manager solutions.
Complex	A solution has to be found to the problem of changing requirements after the contract is signed. Although one can expect the requirements to change during the requirements analysis, the customer often does not understand that this affects the budget.

2. Select candidate methods that meet one or more aspects of the identified needs.

   The candidate methods that were selected are: the unified software developmentprocess (UP), UML-based Web-engineering (UWE), and the proprietary company method (GX).

3. Analyze candidate methods and store relevant method fragments in a method base.

   We analyzed the methods by modeling them into PDDs. The method base was filled with four GX PDDs, 2 UP PDDs, and four UWE PDDs. In total, 10 process data diagrams were stored in the method base.

4. Assemble a new method from useful method fragments and use route map configuration to obtain situational methods.

   Based on the implementation situations needs, we chose the method fragments for the new method. The resulting method consists of three phases (acquisition, orientation, and definition) and three routes (standard, complex, and migration). The rest of the method (design, realization, and implementation) were subject to further research. In the next section, we will further elaborate on the resulting method.

Routemap of the Definition Phase

Instead of showing the entire PDD, we will depict the standard and complex routes in one diagram, to make clear what the differences are between the two implementation situations. Therefore, we omitted the data-side of the diagram, as can be seen in Figure 29. The main activities in the diagram are marked to indicate the original method. A checked pattern indicates thatthis method fragment originates from the proprietary method at GX; grey indicates that it is a UWE fragment; and, finally, white indicates a unified process origin. Note that the roles in this diagram are omitted, because all activities are handled by the same

person, namely the consultant.

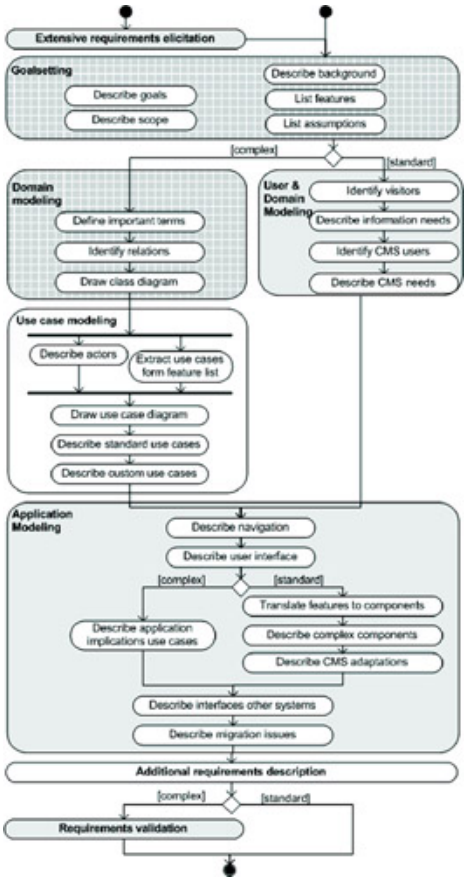
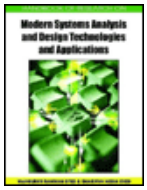


Figure 29: Routemap of the definition phase in WEM

The main difference between the standard and complex route is, nextto the extensive requirements elicitation and validation, the use of use case modeling. In the complexroute, this isusedto describe the people who will interact with the Web application, as well as how they will interact with the system. In the standard route, this is partly handled in the user and domain modeling fragment and partly in the application modeling.

Method Implementation

WEM was developed with input of the requirements management workgroup. The goal of this workgroup was an overall improvement in the requirements process at GX. Members of the workgroup were consultants and project managers of GX and one external consultant. After validating the WEM method in an expert review and two case studies, the method was implemented in the company. Firstly, templates were written for every document that had to be delivered after completing a method stage. Secondly, explanations were provided with the templates. This information was then published on the intranet of the company and presented for the developers, consultant, architects, and project managers.



FUTURE TRENDS

History has proven that new types of information systems are conceived frequently. Emergent technologies for mobile systems, Web applications, and intelligent agent solutions have triggered innovative IS applications. At the same time, new methods are developed to design and support these information systems. Existing method knowledge can play an important role, as parts of it can be reused in new project situations, and is codified into new methods. This has also happened when the well-known state transition diagrams where reused into UML (OMG, 2004) for the analysis and design phases of object-oriented IS.

The trend is to build method knowledge infrastructures, that is, a Web-enabled knowledge resource that can be accessed, shared, and enriched by all IS project workers. Those knowledge infrastructures can be open to anyone on the Internet, such as the open modeling language (Firesmith, Henderson-Sellers, & Graham, 1998), an object oriented modeling language.

Many large IT service companies have development methods on their corporate Internet, where every IS project on any location worldwide, can be executed according to the same method. Usually, these corporate methods are enriched by borrowing new concepts from the public ones on the Internet. In the future, more and more open method knowledge infrastructures for specific types of IS will be established and gradually expanded into full-blown environments to execute complete IS development projects. The facilities for situational method engineering to adapt the method to specific project circumstances are to be included. An example for the product software vendors is the product knowledge software infrastructure that enables product software companies to obtaina custom-made advice that helps them to mature their processes (Weerd et al, 2006).

CONCLUSION

PDDs have proven to be effective means for the meta-modeling of methods, especially for the analysis and design stages. The meta-modeling can serve different purposes: in the examples, two purposes are explained, namely method analysis or method construction. Other possible applications are method comparison and method adaptation. Providing the explicit description of the activities and concepts of a method in a PDD allows for a more formal addition of activities and deliverables. Method engineering tools for modeling and adapting methods will aid in the creation of high quality situational methods.

REFERENCES

Booch, G.,Rumbaugh, J., &Jacobson, I. (1999). *The unified modeling language user guide.* Redwood City, CA: Addison Wesley Longman Publishing Co., Inc.

Brinkkemper, S. (1996). *Method engineering: engineering of information systems development methods and tools. Information and Software Technology,* 38(4), 275-280

Brinkkemper, S.,Saeki, M., &Harmsen, F. (1999). *Meta-modelling based assembly techniques for situational method engineering. Information Systems,* 24(3), 209-228

Coleman, F.,Hayes, F, &Bear, S. (1992). *Introducing objectcharts or how to use Statecharts on object-oriented design. IEEE Trans Software Engineering,* 18(1), 9-18

Firesmith, D.,Henderson-Sellers, B. H.,Graham, I., &Page-Jones, M. (1998). *Open modeling language (OML)—reference manual. SIGS reference library series.* Cambridge, etc.: Cambridge University Press

Harmsen, F,Brinkkemper, S., &Oei, J. L. H. (1994). *Situational method engineering for informational system project approaches. In Proceedings of the IFIP WG 8.1 Working Conference on Methods and Associated Tools for the IS Life Cycle* (pp. 169-194)

Jacobson, I.,Booch, G, &Rumbaugh, J. (1999). *The unified software development process.* Redwood City, CA: Addison Wesley Longman Publishing Co., Inc.

Karlsson, F (2002). *Bridging the gap between method for method configuration and situational method engineering.* Skövde, Sweden: Promote IT

Natt och Dag, J.,Regneil, B.,Gervasi, V., &Brinkkemper, S. (2005). *A linguistic-engineering approach to large-scale requirements management. IEEE Software,* 22(1), 32-29

Object Management Group. (2004). *UML 2.0 superstructure specification (Technical Report ptc/04-10-02)*

Ralyté, J.,Deneckère, R., &Rolland, C. (2003). *Towards a generic model for situational method engineering. Lecture Notes in Computer Science,* 2681, 95

Rolland, C,Prakash, N., &Benjamen, A. (1999). *Amulti-model view of process modelling. Requirements Engineering,* 4(4), 169-187

Saeki, M. (2003). *Embedding metrics into information systems development methods: an application of method engineering technique. In Proceedings of the 15th Conference on Advanced Information Systems Engineering* (pp. 374-389)

Weerd, I. van de,Brinkkemper, S.,Souer, J., &Versendaal, J. (2006). *A situational implementation method for Web-based contentmanagement system-applications: Method engineering and validation in practice [Accepted for a special issue]. Software Process: Improvement and Practice.*

Weerd, I. van de,Brinkkemper, S., &Versendaal, J. (2006). *Incremental method evolution in requirements management: A case study at Baan 1994-2006 (Technical report UU-CS-2006-057).* Institute of Computing and Information Sciences, Utrecht University

Weerd, I. van de,Brinkkemper, S., &Versendaal, J. (2007). *Concepts for incremental method evolution: Empirical exploration and validation in requirements management. 19th Conference on Advanced Information Systems Engineering.* Trondheim, Norway

Weerd, I. van de,Versendaal, J., &Brinkkemper, S. (2006, June 5-6). *A product software knowledge infrastructure for situational capability maturation: Vision and case studies in product management. In Proceedings of the Twelfth Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'06).* Luxembourg: Grand-Duchy of Luxembourg

KEY TERMS

Activity: A process step that is used for capturing the process-view of a method.

Concept: A set of objects that share the same attributes, operations, relations, and semantics, used for capturing the deliverable-view of a method.

Method Engineering: The engineering discipline to design, construct, and adapt methods, techniques, and tools for the development of information systems.

Method Fragment: A coherent piece of an IS development method (Brinkkemper, 1996). Methods fragments are distinguished in process fragments, for modeling the development process, and product fragments, for modeling the structure of the products of the development process.

Process-Deliverable Diagram: A process-deliverable diagram (PDD) consists of two integrated diagrams. The left-hand side of the diagram is based on a UML activity diagram and the right-hand side of the diagram is based on a UML class diagram.

Route Map: A route map represents apredefined path of a method where method fragments are combined to form new situational methods.

Situational Method: An information systems development method tuned to the situation of the project at hand.

PROCESS-DELIVERABLE DIAGRAMS

This section describes the technique used for modeling activities and artifacts of a certain process. As we are modeling methods and not the artifacts of an IS, this type of modeling is called meta-modeling. We express the meta-models of method in PDDs, which consist of two integrated diagrams. The process view on the left-hand side of the diagram is based on a UML activity diagram (OMG, 2003) and the deliverable view on the right-hand side of the diagram is based on a UML class diagram (OMG, 2003). In this chapter, first the left-hand side of the diagram is explained, then the right-hand side, and finally the integration of both diagrams.

The meta-modeling technique is explained, where notational explanation will be illustrated by an example in practice. Those examples are taken from a project to create a situational method for implementing Web-applications with a content management system (Weerd et al., 2006).

Meta-Process Modeling

Meta-process modeling is done by adapting the UML *activity diagram.* According to Booch, Rum-baugh, and Jacobson (1999), an activity diagram is "a diagram that shows the flow from activity to activity; activity diagrams address the dynamic view of a system." This diagram consists of activities and transitions. Activities can be decomposed into sub-activities, if necessary, and thereby creating a hierarchical activity decomposition. Transitions can be used to show the control flow from one activity to the next. A simple arrow depicts this. Four types of transitions exist: unordered, sequential, concurrent, and conditional activities.

Activity Types

We use different types of activities for several reasons. Firstly, activity types are used for scope definition. Itmay be interesting to know which processes exist adjacent to the process that is focused on. The details of this process are not interesting and are therefore not shown. Secondly, for purposes of clarity of the diagram, some processes are expanded elsewhere. Finally, in some cases the sub activities are not known or not relevant in the specific context.

The following activity types are specified (see Figure 1):

- **Standard activity:** An activity that contains no further (sub) activities. A standard activity is illustrated with a rounded rectangle.
- **Complex activity:** An activity that consists of a collection of (sub) activities. They are divided into:
  - **Open activity:** A complex activity whose (sub) activities are expanded. This expansion can be done in the same diagram or in another diagram. Therefore, we use two notational variants, which are:
    - A rounded rectangle, containing two ore more sub activities and
    - A rounded rectangle with a white shadow, to indicate that the activities are depicted elsewhere.

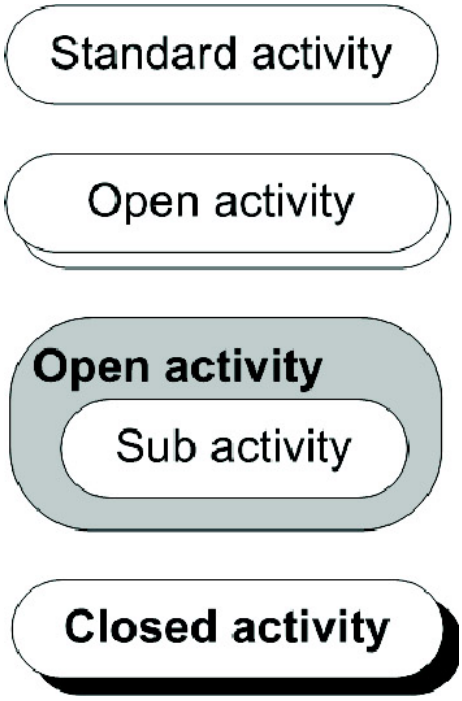


Figure 1: Activities Types

  - **Closed activity:** A complex activity whose (sub) activities are not expanded since it is not known or not relevant in the specific context.

In Figure 2 we give some examples of activity types. 'Listfeatures' is a standard activity that has no further sub activities. In case of 'describe candidate requirements,' the activity is open, since we find it interesting to describe its sub activities. 'Define use case' model is closed, since we are not interested in the details of this process. The naming conventions for activity names are that the names always consist of a verb and an object, except when the activity name is literally copied from a method source, such as a book. Activities for the highest level are excluded form this naming convention, since they usually get the name of a stage or phase.

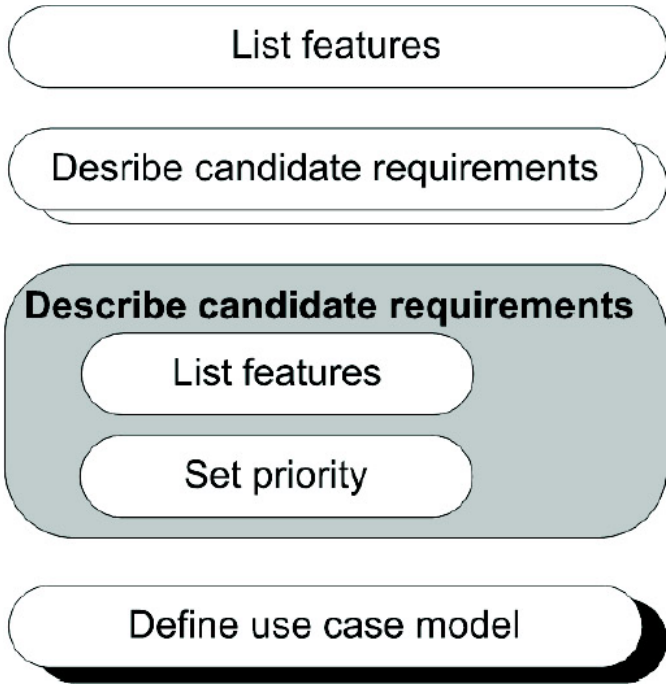


Figure 2: Example activity types

In the following sections, the four activity transitions will be introduced and exemplified.

Sequential Activities

Sequential activities are activities that need to be carried out in *a. pre defined* order. The activities are connected with an arrow, implying that they have to be followed in that sequence, although the activity completion before the next can be started is *not* strictly enforced. Both activities and sub-activities can be modeled in a sequential way. In Figure 3 an activity diagram is illustrated with one activity and two sequential sub-activities. A special kind of sequential activities are the start and stop states, which are also illustrated in Figure 3.

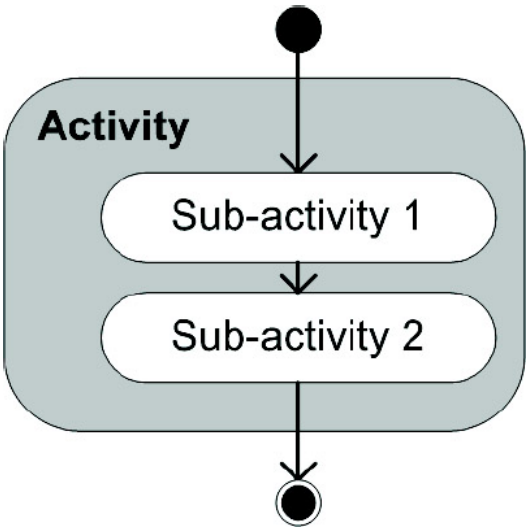
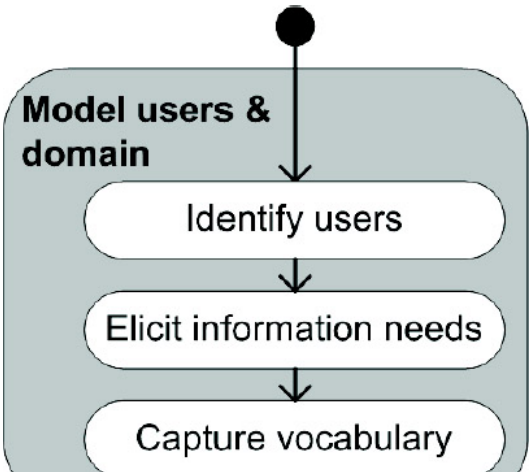


Figure 3: Sequential activities

In Figure 4 an example is illustrated. The example is taken from the requirements capturing workflow, of which the main activity, 'model users & domain,' consists of three activities that need to be carried out in a predefined order.



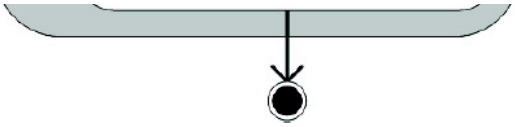


Figure 4: Example equential activities

Unordered Activities

Unordered activities are used when sub-activities of an activity can be executed in any order, that is, they do not have a predefined execution sequence. Only sub-activities can be unordered. Unordered activities are represented as sub-activities without transitions within an activity, as is shown in Figure 5.

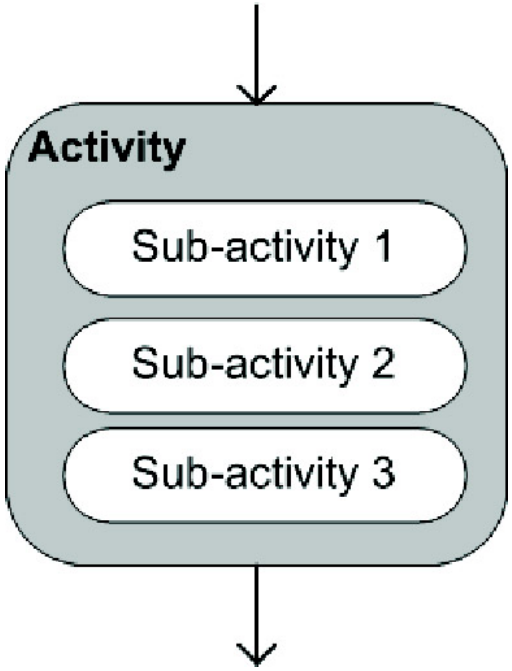


Figure 5: Unordered ctivities

An activity may consist of sequential and unordered activities, which is modeled by dividing the main activity in different parts. In Figure 6 an example is taken from the requirements analysis workflow. The main activity, 'describe candidate requirements,' is dividedintotwoparts. Thefirstpart is a sequential activity. The second part consists of four activities that do not need any sequence in order to be carried out correctly. Note that all unordered activities must be carried out before continuing to the next activity.

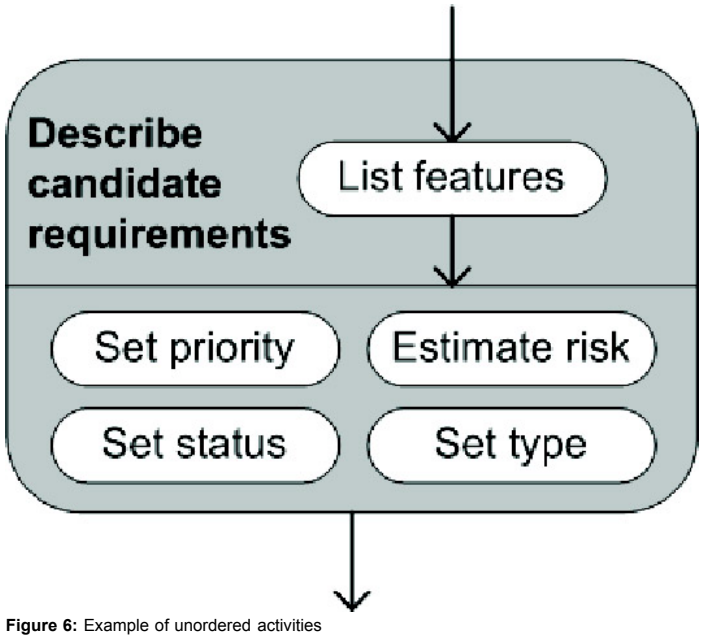


Figure 6: Example of unordered activities

Concurrent Activities

Activities can occur concurrently. This is handled with forking and joining. By drawing the activities parallelin the diagram, connected witha synchronization bar, one can fork several activities. Later on these concurrent activities can join again by using the same synchronization bar. Both the concurrent execution of activities and sub-activities can be modeled. In the example of Figure 7, Activity 2 and Activity 3 are concurrent activities.

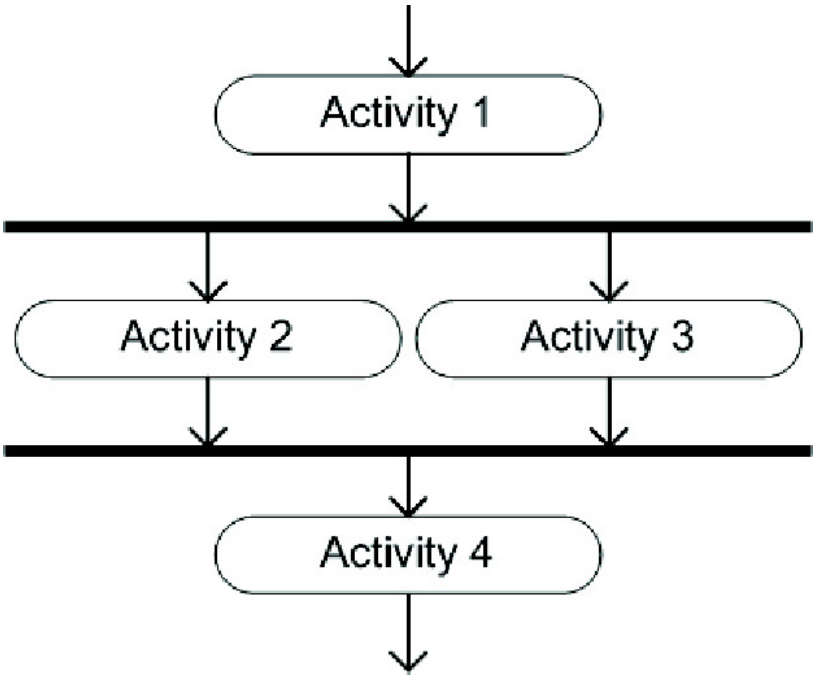


Figure 7: Concurrent activities

In Figure 8 a fragment of the requirements capturing process is depicted. Two activities, ‘define actors’ and ‘defining use cases’ are carried out concurrently. The reasonfor carrying outthese activities concurrently is that definition of actors and of use cases influence each other to a high extend.

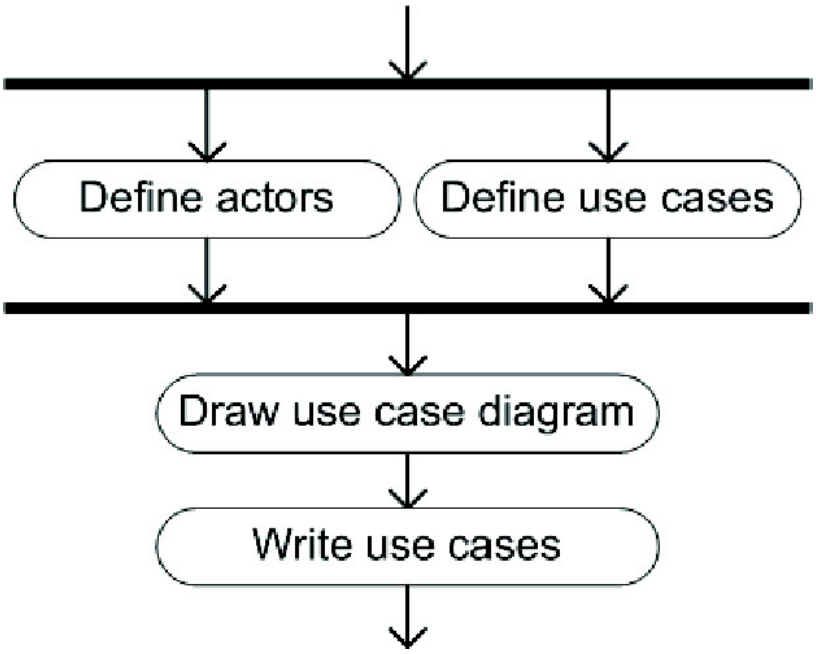


Figure 8: Example concurrent activities

Conditional Activities

Conditional activities are activities that are only carried out if a predefined condition is met. This is graphically repre sented by using abranch. Branche s are denoted with a diamond and can have incoming and outgoing transitions. Every outgoing transition has a guard expression, the condition, denoted with square bracket ‘[ ]’. This guard expression is actually a Boolean expression, used to make a choice which direction to go. Both activities and sub-activities can be modeled as conditional activities. In Figure 9, two conditional activities are illustrated.

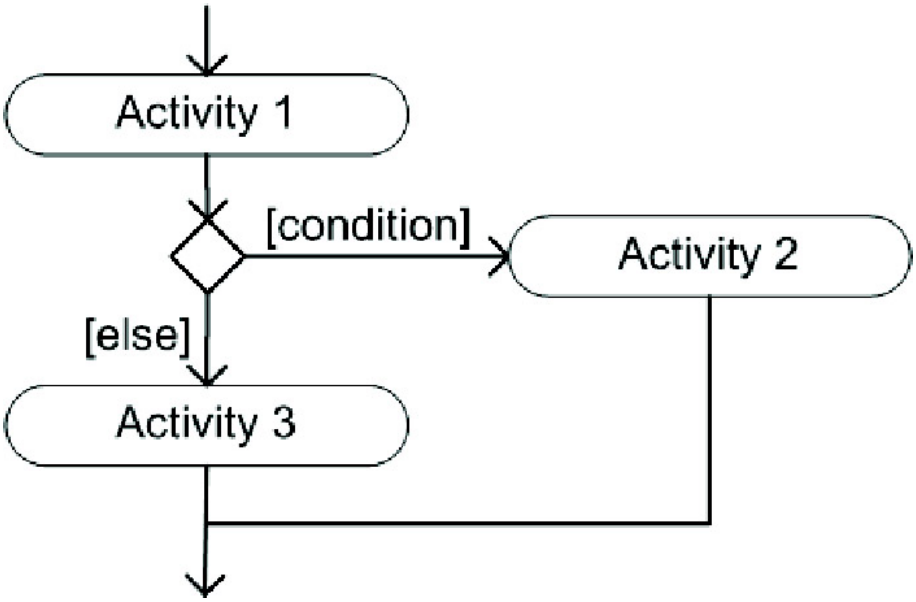


Figure 9: Conditional activities

In Figure 10, an example from a requirements analysis starts with studying the material. Based on this study, the decision is taken whether to do an extensive requirements elicitation session or not. The condition for not carrying outthis requirements session is represented at the left of the branch, namely [requirements clear]. If this condition is not met, [else], the other arrow is followed.

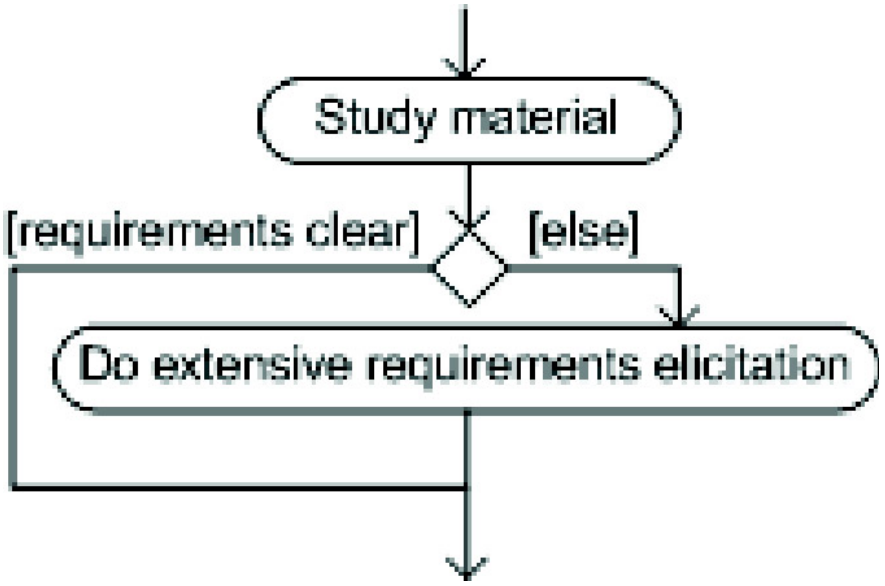


Figure 10: Example conditional activities

Roles

In some methods it is explicitly stated by which individuals or organizational role the process is to be carried out. In this case, the role is indicated in the activity. In Figure 11, an activity with three sub activities is depicted. In the lower right corner, the role is positioned. Please note that the usage of roles is not restricted to activities, but, if necessary, can also be used in sub-activities.

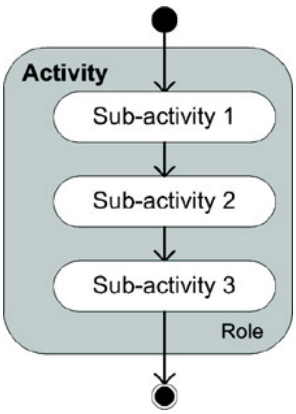


Figure 11: Roles

In Figure 12, the usage of roles is illustrated. The same example as in Figure 4 is used with the difference that the role, in this case ‘consultant,’ is indicated.

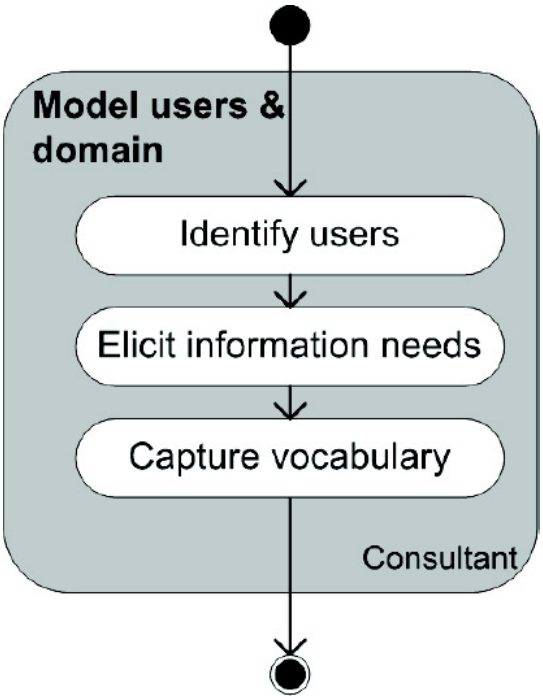


Figure 12: Example role

Meta-Deliverable Modeling

The deliverable side of the diagram consists of a *concept diagram.* This is basically an adjusted class diagram as described Booch et al. (1999). Important notions are concept, generalization, association, multiplicity and aggregation.

Concept Types

A concept is a simple version of a UML class. The class definition of Booch et al. (1999) is adopted to define a concept, namely: "a set of objects that share the same attributes, operations, relations, and semantics."

The following concept types are specified:

- **STANDARD CONCEPT:** A concept that contains no further concepts. A standard concept is visualized with a rectangle.
- **COMPLEX CONCEPT:** A concept that consists of an aggregate of other concepts. Complex concepts are divided into:
    - **OPEN CONCEPT:** Acomplexconcept whose sub concepts are expanded. An open concept is visualized with a white shadow border. The aggregate structure may be shown in the same diagram or in a separate diagram.
    - **CLOSED CONCEPT:** A complex concept whose sub concepts are not expanded since it is not relevant in the specific context. A closed concept is visualized with a black shadow border.

Similar as for activities, this usage of different concept types is introduced for clarity and scope definition.

In Figure 13, the three concepttypes that are used in the modeling technique are illustrated. Concepts are singular nouns and are always capitalized, not only in the diagram, but also when referring to them in the textual descriptions outside the diagram.

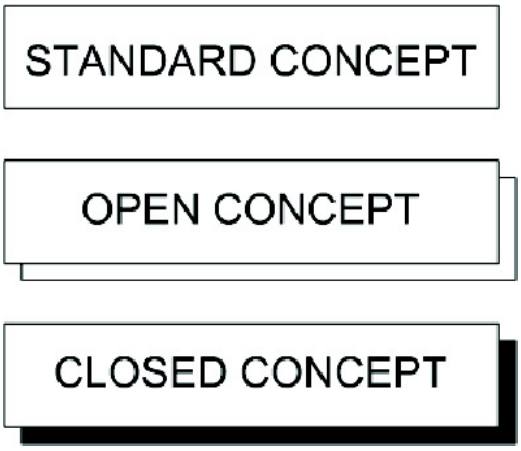


Figure 13: Standard, open and closed concepts

In Figure 14, all three concepttypes are exemplified. Part of the PDD of a requirements workflow is illustrated. The USE CASE MODEL is an open concept and consists of one or more ACTORS and one or more USE CASES. ACTOR is a standard concept, it contains no further sub-concepts. USE CASE, however, is a closed concept. A USE CASE consists of a description, a flow of events, conditions, special requirements, and so forth. Because in this case we decided it is unnecessary to reveal that information, the USE CASE is illustrated with a closed concept.

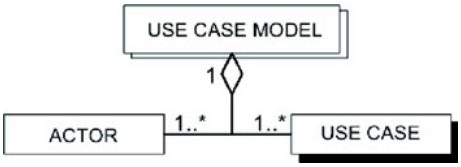


Figure 14: Example of standard, open and closed concepts

Generalization

Generalization is a way to express a relationship between a general concept and a more specific concept. Also, if necessary, one can indicate whether the groups of concepts that are identified are overlapping or disjoint, complete, or incomplete. Generalization is visualized by a solid arrow with an open arrowhead, pointing to the parent, as is illustrated in Figure 15.

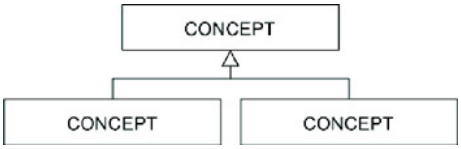


Figure 15: Generalization

In Figure 16, generalization is exemplified by showing the relationships between the different concepts described in the preceding paragraph. CONTROL FLOW and DATA FLOW are both a specific kind of FLOW. Also note the use of the *disjoint* (d) identifier. This implies that occurrence of one concept is incompatible with the occurrence of the other concept; that is, there are no CONTROL FLOWS that are also DATA FLOWS and vice versa. The second identifier we use is *overlapping* (o), which would indicate in this example that there may be occurrences that are CONTROL FLOWS or DATA FLOWS, but also occurrences that are both. Finally, *categories* (c) mean that the disjoint concepts have no occurrences in common and that the decomposition is exhaustive. In the example, this would imply that every occurrence of flow is either a CONTROL FLOW or DATA FLOW. No other FLOWS exist, and there are no occurrence s that are both CONTROL FLOW and DATA FLOW.

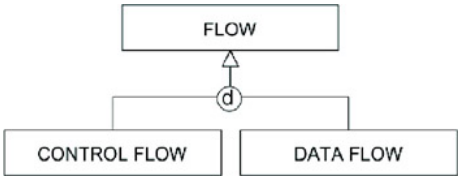


Figure 16: Example generalization

Association

An association is a structural relationship that specifies how concepts are connected to another. It can connect two concepts (binary association) or more than two concepts (n-ary association). An association is represented with an undirected solid line. To give a meaning to the association, a name and name direction can be provided. The name is in the form of an active verb and the name direction is representedby atrianglethatpoints in the direction one needs to read. Association with a name and name direction is visualized in Figure 17.

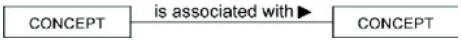


Figure 17: Association

In Figure 18, an example of a fragment is shown of the PDD of the requirements analysis in the Unified Process (Jacobson, Booch, & Rumbaugh, 1999). Because both concepts are not expanded any further, although several sub concepts exist, the concepts are illustrated as closed concepts. The figure reads as "SURVEY DESCRIPTION describes USE CASE MODEL."

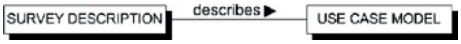


Figure 18: Example association

Multiplicity

Except name and name direction, an association has another characteristic. With *multiplicity,* one can state how many objects of a certain concept can be connected across an instance of an association. Multiplicity is visualized by using the following expressions: *1* for exactly one, *0…1* for one or zero, *0…** for zero or more, 7…* for one or more, or for example, *5* for an exact number. In Figure 19, association with multiplicity is visualized.

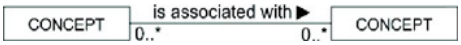


Figure 19: Multiplicity

An example of multiplicity is represented in Figure 20. It is the same example as in Figure 18, only the multiplicity values are added. The figure reads as "exactly one SURVEY DESCRIPTION describes exactly one USE CASE MODEL." This implies inanISprojectthataSURVEY DESCRIPTION will always be relatedtojustone USE CASE MODEL and the other way round.

Figure 20: Example multiplicity

Aggregation

A special type of association is *aggregation.* Aggregation represents the relation between a concept (as a whole) containing other concepts (as parts). It can also be described as a ‘has-a’ or ‘consists of relationship. In Figure 21, an aggregation relationship between OPEN CONCEPT and STAN DARD CONCEPT is illustrated. An OPEN CONCEPT consists of oneormore STANDARD CONCEPTS and a STANDARD CONCEPT is part of one OPEN CONCEPT.

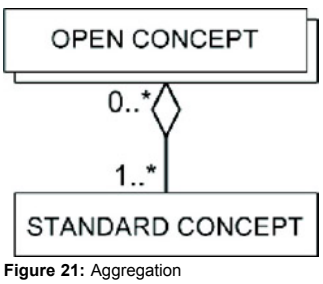


Figure 21: Aggregation

In Figure 22, aggregation is exemplified by a fragment of a requirements capture workflow. A USE CASE MODEL consists of one or more ACTORS and USE CASES.

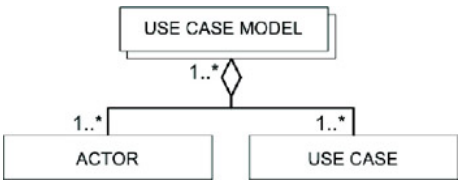


Figure 22: Example aggregation

Properties

Sometimes the need exists to assign properties to CONCEPTS. Properties are written in lower case, under the CONCEPT name, as is illustrated in Figure 23.

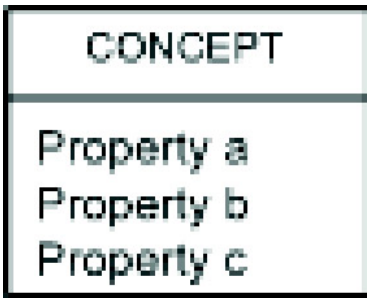


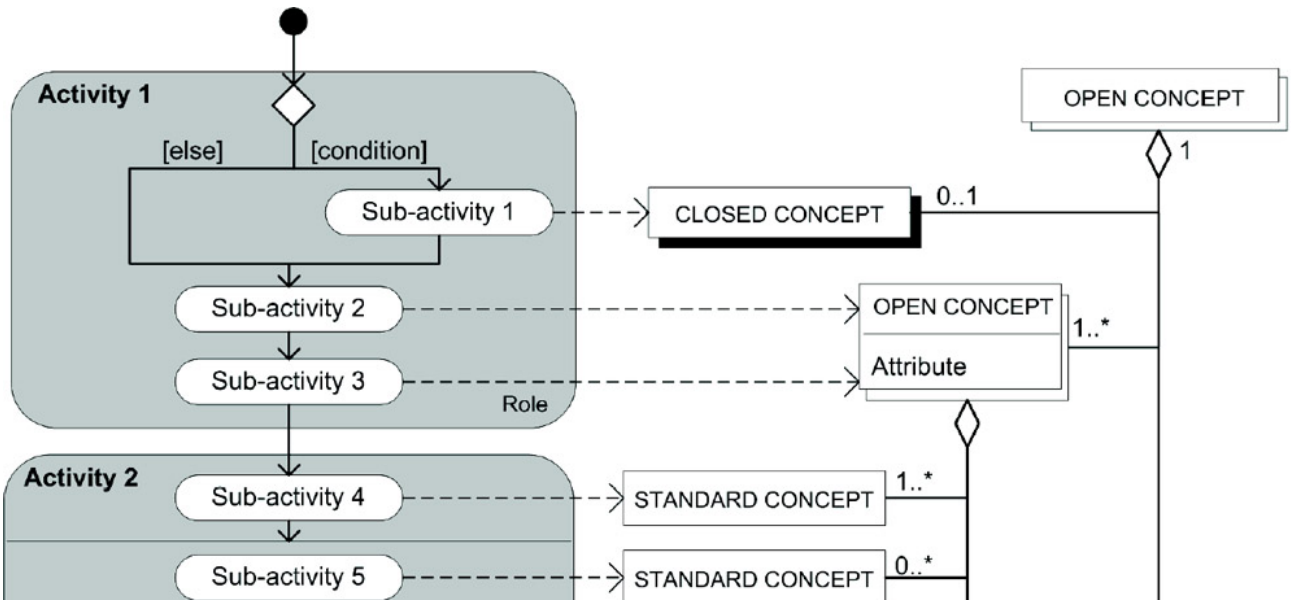
Figure 23: Properties

In Figure 24, an example of a CONCEPT with properties is visualized. DESIGN has seven properties, respectively: code, status, author, effective date, version, location, and application.

Figure 24: Example properties

Process-Deliverable Diagram

The integration of both types of diagrams is quite straightforward. Activities are connected with a dotted arrow to the produced deliverables, as is demonstrated in Figure 25. Note that in UML class diagrams, dotted arrows are used to indicate dependency from one class on another. However, this construction is not used in PDDs. We have some extra remarks concerning the PDD. Firstly, all activities in Figure 25 result in deliverables. However, this is not compulsory. Secondly, it is possible for several activities to point to the same deliverable, see for example, sub-activity 2 and 3. Finally, we assume that all artifacts in the diagram are available to all roles.



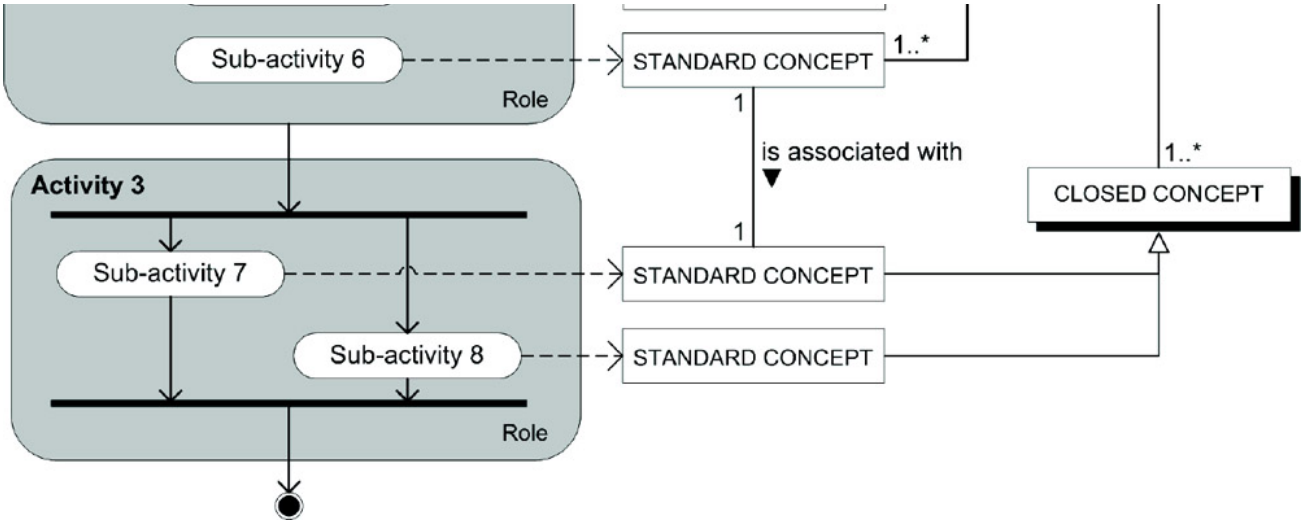


Figure 25: Process-deliverable diagram

Example of a Process-Deliverable Diagram

In Figure 26, an example of a PDD is illustrated. It concerns an example of drawing an object chart, as described in Brinkkemper, Saeki, and Harmsen (1999). This is an example of method assembly, in which an object model and Harel's state chart are integrated into object charts.

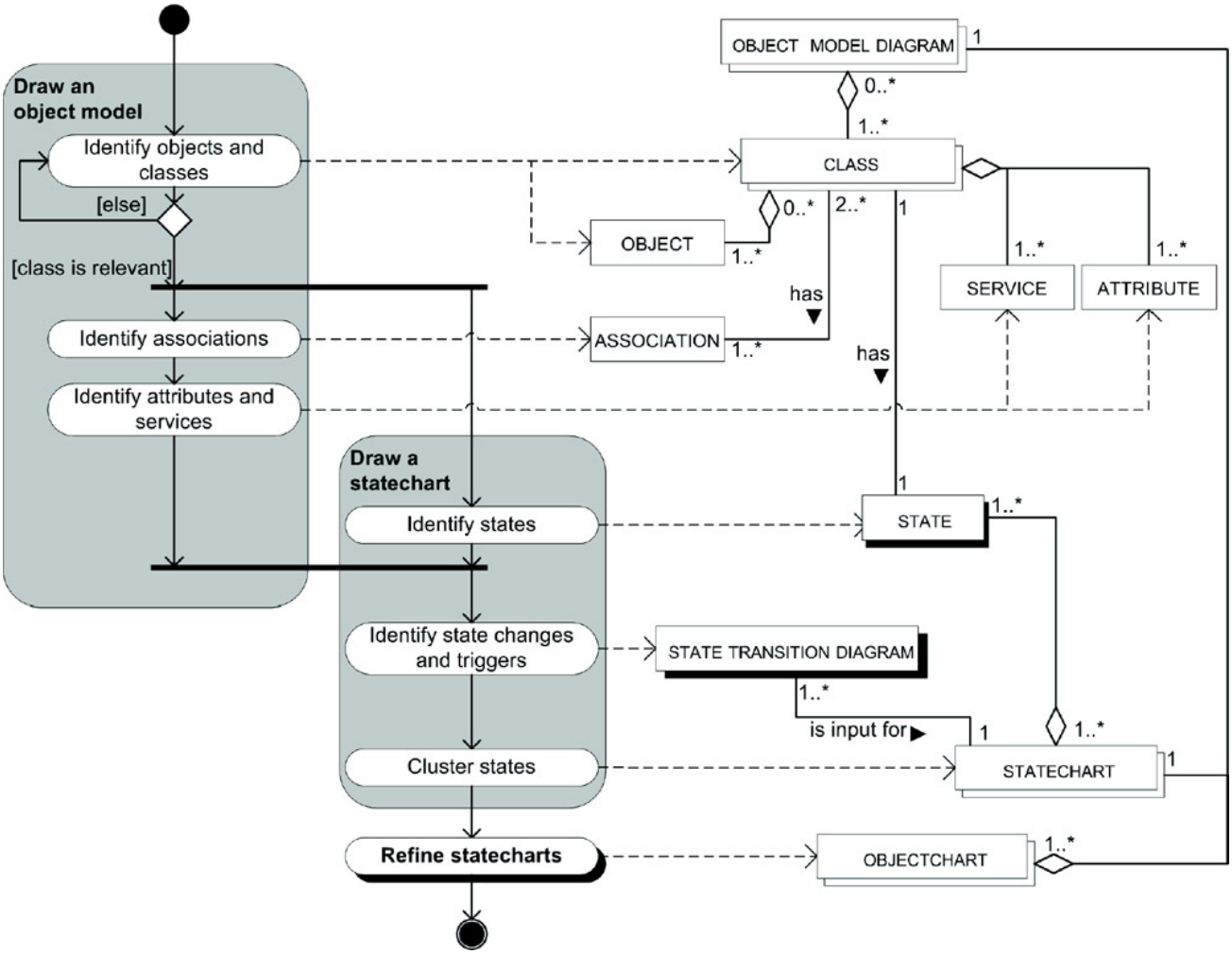


Figure 26: Example PDD—drawing an object chart

Notable is the use of an open concept: STATE TRANSITION DIAGRAM. This concept is open, since it is a complex concept, and should be expanded elsewhere. However, due to space limitations, this is omitted. The activities of 'drawing an object chart' are carried out by one person. Therefore, no roles are depicted in the diagram.

In Table 1, the activities and sub-activities, as well as their relations to the deliverables, which are depicted in the diagram, are described.

Table 1: Activities and sub-activities in drawing an object chart

Open table as spreadsheet

Activity	Sub-Activity	Description
Draw an object model	Identify objects and classes	Key phenomena that require data storage and manipulation in the IS are identified as OBJECTS. OBJECTS that share the same attributes and services are identified as a CLASS.
	Identify associations	Relationships between CLASSES are identified when instance OBJECTS of different CLASSES are to be related to each other in the IS.

	Identify attributes and services	For each CLASS, the descriptive ATTRIBUTES and the operational SERVICES are identified.
Draw a state chart	Identify states	Lifecycle statuses of OBJECTS that serve a process in the IS are to be identified as STATES. Similarly for CLASSES.
	Identify state changes and triggers	Events that trigger changes of a STATE and the conditions under which they occur are determined.
	Cluster states	STATES that are sub statuses of a higher level STATE are groups together in a cluster. Furthermore, state transitions between STATES are depicted resulting in a STATECHART.
Refine state charts		Based on the OBJECT MODEL DIAGRAM and the STATECHART, an OBJECTCHART is created by linking STATECHARTS to each CLASS.

In Table 2, an excerpt if aconcepttable is shown. Every concept is provided with a description and a reference.

Table 2: Excerpt of a concept table

Open table as spreadsheet

Concept	Description
ASSOCIATION	An association specifies a semantic relationship that can occur between typed instances. It has at least two ends represented by properties, each of which is connected to the type of the end (OMG, 2004).
STATE	A state models a situation during which some (usually implicit) invariant condition holds (OMG, 2004).
OBJECTCHART	An extension of a State chart to model reactive systems from an object-oriented view (Brinkkemper, Saeki, & Harmsen, 1999).
…	

The process in Figure 26, where object model diagrams and state charts are amalgamated into object charts, is an example of situational method engineering for a project situation where is was required to model state charts for each class. Similarly, situational factors may require various kinds of adaptations of the method. Different analysis and design situations in a project give rise to a huge variety of method adaptations: in a simple project just one class diagram may be sufficient to analyze the object domain, whereas in a more complex project, various class diagrams, object models and state charts are required. To support the adaptations, PDDs are very instrumental as both modifications of activities as of concepts can be easily documented.