Deriving Use Case Diagrams from Business Process Models

Remco M. Dijkman
University of Twente, Faculty of Computer Science
P.O. Box 217, 7500 AE Enschede
The Netherlands
dijkman@cs.utwente.nl

Stef M.M. Joosten
Ordina Finance Utopics, and
Open University of the Netherlands
P.O. Box 2960, 6401 DL Heerlen
The Netherlands
joosten@anaxagoras.com

Abstract

In this paper we introduce a technique to simplify requirements capture. The technique can be used to derive functional requirements, specified in the form of UML use case diagrams, from existing business process models. Because use case diagrams have to be constructed by performing interviews, and business process models usually are available in a company, use case diagrams can be produced more quickly when derived from business proces models. The use case diagrams that result from applying the technique, specify a software system that provides automated support for the original business processes. We also show how the technique was successfully evaluated in practice.

1. Introduction

Capturing requirements is widely considered to be one of the most difficult tasks in software engineering. At the same time, errors made in this phase are among the most difficult to detect and the most expensive to correct [1]. Therefore, much can be gained from requirements capturing techniques that speed up the creation of requirements specifications, and produce robust requirements specifications. In this paper we propose such a technique.

The technique we propose is based on the observation that one of the forms of requirements engineering, the use case based requirements engineering [3, 4, 12, 10], bears much resemblance to business process modeling. The resemblance between use case based requirements engineering and business process modeling becomes obvious when inspecting the following definitions of use case and business process.

According to Jacobson [10] a UML use case: '... specifies a sequence of actions, including variants, that the system can perform, and that yield an observable result of value to a particular actor.'

According to Davenport [5] a business process is: 'A structured, measured set of activities designed to produce a specified output for a particular customer or market.'

According to their definitions, both a business process and a use case consist of activities (or actions). In both definitions, the activities are ordered in some way. In the definition of business process, they are structured and measured, and in the definition of use case, they are sequential. Also, in both definitions, the activities are aimed at delivering a particular result.

In addition to the similarities that exist between the definitions, it has been noted that business processes can be described by use case models [15, 10, 11, 16].

These observations give rise to the assumption that it is possible to translate business process models to use case models. The benefit of having such a technique is that, while use cases have to be captured by performing interviews in an organization, business processes are often available in the form of working instructions or administrative handbooks. A study at a large international bank, for example, showed that about 350 of the 1000 business processes used in the bank were well described and used for reference in the daily conduct of business.

Because of the obvious similarities between use cases and business processes, and the expected benefits of deriving use case diagrams from business process models, we studied the exact relation between UML use case diagrams and business process models. Our main goal was to specify a procedure to transform business process models into UML use case diagrams that can serve as a basis for further systems development.

The approach chosen is to create metamodels for use case diagrams and for business process models, and compare the two metamodels. The relations between the metamodels give rise to a mapping that forms a basis for the transformation procedure from business process models to use case diagrams.

The remainder of this paper is organized as follows. Sec-

tion 2 and section 3 briefly introduce use case modeling and business process modeling respectively. Also, these sections give the metamodels of both modeling techniques. Section 4 describes a mapping between the two metamodels. Section 5 explains a procedure for transforming business process models into use case diagrams. This procedure is based on the mapping from section 4. Section 6 explains industrial experience we have with the technique. Section 7 presents the conclusions of this paper.

2. UML Use Case Modeling

This section introduces UML use case modeling. Subsection 2.1 gives an overview of use case modeling. Subsection 2.2 gives a metamodel of use case modeling.

2.1. Use Case Modeling Overview

A use case diagram describes how an entity may use the system under development. To this end, a use case diagram contains actors and use cases. An actor is an entity that may interact with the system. A use case is a set of interactions between any number of actors and the system under development. Thus, a use case describes a part of the behavior of the system.

An example of a use case diagram is shown in figure 1. The ovals represent use cases, and the puppets represent actors. A line between an actor and a use case represents the actors involvement in one or more of the interactions covered by the use case. The example models a system that supports the processing of mortgage applications. The system can be used to enter the mortgage data, perform a creditworthiness check, draw up an offer, and process an advice.

When two use cases partly share the same behavior, this behavior may be put into a third use case, and the original two use cases may be defined to 'include' the behavior of this new use case. In the description of the original use cases, the point at which to include the behavior described by the third use case is specified. The goal of this construction is to remove redundancy. Thereby, it allows a use case diagram to be better understood, and avoids that functionality is implemented twice. An include relationship is represented by a dashed arrow, labeled ≪include≫.

When one use case is a generalized form of another use case, we can draw a generalization relationship between them. We could, for example, draw a generalization relationship from the use case 'process complaint' to the use case 'process incoming mail'. Generalization relationships can also exist between actors. The actor 'employee', for example, is a generalization of the actor 'administrative worker'. A generalization relationship is represented by a solid line with a hollow arrowhead from the specific to the generic use case or actor.

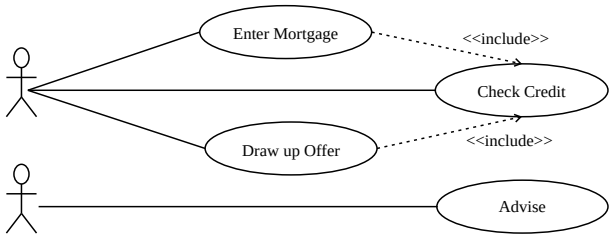

Figure 1. Example use case diagram

Table 1. Example of a use case's interactions

1.	enter name and address
2.	enter mortgage details
3.	carry out the use case: 'check credit'
4.	send initial acceptance to client
4.	send rejection to client

We may describe an extension to the behavior of a use case in terms of another use case. To do this, we define an extension relation between the two use cases. If at a certain point (the extension point) in the execution of the base use case, a specified condition is met, then the extension use case is carried out. An extend relationship is represented by a dashed arrow, labeled ≪extend≫.

Each use case can be described in detail, by describing the interactions between the actors and the system, and the order in which they occur. Many description techniques exist to detail a use case. We could, for example, use state machines [16], or activity diagrams [9]. In this paper we use numbered lists to model a use case's interactions and their order. If two numbers occur twice, this models a choice between two possible interactions. The use case 'enter mortgage' in figure 1 could, for example, be detailed by the numbered list in table 1.

2.2. Metamodel of a Use Case Diagram

The metamodel a use case diagram is shown in figure 2. It is a simplified version of the metamodel that can be found in the UML specification [16].

3. Business Process Modeling

This section introduces business process modeling. Subsection 3.1 gives an overview of business process modeling. Subsection 3.2 gives a metamodel of business process modeling.

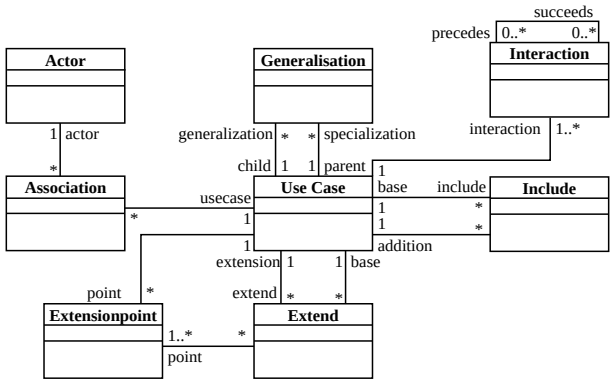

Figure 2. Use case metamodel

3.1. Business Process Modeling Overview

Many techniques for business process modeling exist. In an earlier study [18], we studied 18 of the most referenced techniques. From this study we were able to draw up a generalized view on business process modeling, which we will use in this section. Although all business process modeling techniques have means to model aspects other than behavior, such as organizational structure, and data objects, we will focus on modeling behavior. The reason for this is that use case modeling does not have any concepts to model aspects other than behavior. Therefore these aspects can not be mapped, and are thus irrelevant in this paper.

A business process model describes the tasks that have to be carried out, and the order in which these tasks have to be carried out. A task is the smallest unit of work that makes sense to a user. Each task is assigned to a role. A role is a group of entities that have the same rights and obligations with respect to performing a task or a group of tasks. A role may be assigned to any number of tasks, and an entity may act in any number of roles.

In this paper, we use UML activity diagrams to model business processes. Activity diagrams have been shown to be useful for business process modeling in [2, 7]. An example of an activity diagram that models a business process for mortgage processing is shown in figure 3. The rounded rectangles represent tasks, and the arrows represent transitions between tasks. The names in the columns represent roles, and a task in the column of a role represents that this task is assigned to that role. The bullet, and the bulls eye, represent the beginning and the end of the business process respectively. The abbreviations aut., sup., and man. in the task descriptions represent that a task is performed automatically by a software system, supported by a software system, or performed manually. The example describes the procedure that is followed when a mortgage application is received, until the application is processed and has resulted in an offer.

We can use a guard to model a condition that must be met before starting the following task. A guard could be, for example, 'legal act received'. A guard is graphically represented as a label between square brackets on a transition.

We can use a branch to model a choice on the task that has to be started after the current task has finished. A branch has decision criteria attached to it that define under which condition, which task is started. Decision criteria could be, for example, 'if amount ≤ 50, start task A, otherwise start task B'. Decision criteria cannot overlap. It is for example illegal to specify a choice with decision criteria 'if amount ≤ 50, start task A. If amount ≥ 40, start task B'. A branch is represented by a diamond with one incoming arrow and at least two outgoing arrows. The outgoing arrows have guards attached to them that represent the decision criteria.

We can use a fork to model that two or more tasks are started in parallel (i.e. are carried out in random order) after the current task has finished. When two or more tasks are started in parallel, we may want to wait for all tasks to be finished before going on to the next task. This can be achieved by using a join. A join represents that all tasks that are attached to its source have to be finished before the task at its end can be started. Note that a join does not necessarily have to join the same tasks that where initiated by the fork. When we receive an article for a conference, for example, we may want to start three tasks in parallel, to forward the article to three international experts. After the tasks for forwarding, there may be tasks for reminding the experts of the deadline and for receiving the reviews. These tasks are all carried out in parallel reviews. However, we will want to wait until all three reviews are received, before sending the comment to the person who submitted the paper. A fork is represented by an arrow to a thick line, from which arrows leave to the tasks that have to be started. A join is represented by a set of incoming arrows to a thick line, from which an arrow leaves to the task that has to be started next. Without loss of generality, we chose to disregard the fork and join construct in this paper to simplify the mapping. The fork and join construct can be added to the transformation procedure in future work.

3.2. Metamodel of a Business Process Model

The metamodel of business process modeling is shown in figure 4. It has been constructed by generalizing the metamodels of 18 tools studied in [18]. As in the previous section, we focus on the behavioral aspect of business process modeling.

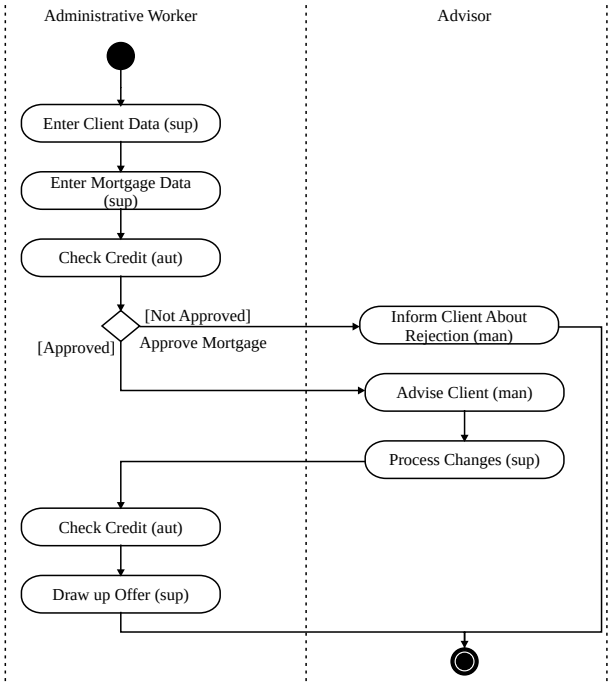

Figure 3. Example business process model

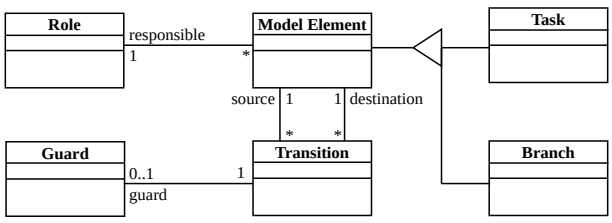

Figure 4. Business process metamodel

4. Mapping

In this section, we define a mapping from business process modeling concepts and their relations, to use case modeling concepts and their relations.

When we say that two concepts or relations can be mapped, this means that the extension (as opposed to the intension) of the source concept or relation is also a valid extension of the target concept or relation. A mapping from the concept Person to the concept Building would, for example, not be possible. The reason for this is that the extension of the concept Person, which contains Mr. Jameson, is not a valid extension of the concept Building.

The extension of a concept or a relation is defined by its intension. The intension of a concept or relation is defined by the definition of this concept or relation, and the additional semantic constraints in which it participates. Therefore, we can assess the validity of a mapping, by checking if the definitions, and additional semantic constraints of the concepts or relations that are mapped, match.

The approach to define the mapping is to first create an initial mapping, based on the definitions of the concepts and relations. Thus, the initial mapping is based on the intension of the concepts and relations. Second, we verify the correctness of the initial mapping, by validating a formal specification of the mapping. The formal specification of the mapping specifies the extension of the concepts and relations. Thus the validation of the mapping is based on the extension of the concepts and relations. In our case the formal specification of the mapping consists of a Z specification of the mapping itself, and a Z specification that we derived from the metamodels of the modeling techniques. We validate the specification by checking if the semantic constraints that can be derived from the original semantic constraints and the mapping, represent desirable properties.

We used this approach to mapping modeling techniques before in [6, 14]. A more detailed description of the approach is given in [14].

4.1. Assumptions

Because a use case diagram describes the behavior of the system under development, we must decide what part of a business process we are going to automate before we can draw a use case diagram for it. For each task there are three possibilities: automated, supported or manual. A task is automated when the system performs it without the intervention of an external entity. An example of an automated task is the automated generation of a letter. A task is supported when an external entity performs the task with the help of the system. An example of a supported task is the task of deciding whether a client can have a mortgage, while obtaining the client's data from the company database. In

this case, the actual task is performed by an entity that is external to the system (an employee), while this entity is supported by the system (the company database). A task is manual when it is fully performed by an external entity.

For now, we will assume that each task is supported. We will show later, how we can modify the mapping to cater for automated, and manual tasks.

Also, we will assume that a task in a business process model corresponds to an interaction of an entity with the system under development. This can be done safely when assuming that tasks and interactions are described at the same level of detail.

4.2. The initial mapping

We map a business process role to a use case actor. According to its definition, an actor defines a coherent set of parts[1] that users of the system can play when interacting [16]. Every time a specific user interacts with the system, it is playing such a part. When mapping business process roles to actors, it must be enforced that an actor defines a *coherent* set of parts that users of the system can play when interacting. Since the UML semantics does not put any constraints on what exactly is considered to be a coherent set of parts, we assume that any coherency will do. Furthermore, because a business process role is defined as a group of entities that have the same rights and obligations with respect to performing a task or group of tasks, we can safely assume that a business process role defines a coherent set of parts that users can play when performing tasks.

According to its definition, an actor can be assigned to any number of interactions. When mapping a business process role to an actor, and assuming that a task is equivalent to an interaction, this is still possible. However, an interaction can only be assigned to a single actor, because a task can only be assigned to a single role. This disallows interactions between two actors to be modeled. This is a limitation of the proposed mapping, we have to accept when using the mapping.

A use case can either specify a complete system, or any model element that displays behavior, like a subsystem, or a class in the model of a system [16]. It has even been proposed to use use cases to model business processes [10, 11, 16]. Therefore we can choose freely what business process modeling concept we map to the use concept, as long as it describes behavior. The business process metamodel contains two concepts that describe behavior: task and procedure. We can therefore map either of these concepts to a use case.

When we map a task to a use case, each use case would only model a single interaction, because we assumed that a task is equal to a single interaction. However, according to the UML semantics, a use case must describe a 'complete sequence'. This means that the use case specifies *all* the interactions that have to be carried out to bring the system in a state in which the use case can be performed again. Thus limiting the use case to one interaction would be too constraining. Therefore, mapping a task to a use case is not valid.

When mapping a procedure to a use case, the resulting use cases would become very complex. This can be seen from the number of actors that would then be involved in one use case (which equals the number of roles involved in a procedure, and in the performed case studies was 4 on average). The number of alternative paths that have to be defined for the use case would also be high. Therefore, this is not a practical mapping.

Since no mapping can be made from a business process modeling concept to the use case concept, we introduce the concept of step [13], and map a step to a use case. A step is a sequence of tasks that can be performed without interruption by the same role. 'Send offer', for example can be a step. 'Send offer and process reply' cannot be a step, because time passes between sending the offer and receiving a reply. Because a step is a sequence of tasks that can be performed without interruption by the same role, it is a logical unit of work to the users, and therefore is experienced by the users as a complete sequence. Also, the complexity of the use case is limited, because the number of actors attached to it is by definition one, and the number of alternative paths is limited.

The introduction of the step concept, requires that the step concept is also is introduced in the meta model of business process modeling. We will take this into account in the next section.

We map the association between a step and a role to an association between the corresponding actor and use case. Because all the tasks in a step can be performed by exactly one role, the association between step and role is equivalent to the association between one of the tasks in a step and a role.

We map transitions between tasks in steps to ordering between interactions in use cases. We map a guard on a transition to the description of a constraint on an interaction.

Use cases have a basic path, which is the most common path, and exceptions to this path that define different behavior [10]. The same holds for business processes. We use the differentiation between a basic path, and alternative paths, to define a mapping for branches. We map the basic path of a branch in a procedure to (a part of) the basic path in a use case, and we map each alternative path of a branch in a procedure to an alternative path in a use case. Alternatively, we may choose to map the alternative paths of the branch to an extending use case. To be able to do this however, the basic

[1] The original term was 'role'. However, the term 'role' may be confused with a role in a business process. Therefore we use the term 'part'.

Table 2. Mapping from business process to use case concepts

Business Process Concept	Use Case Concept
Role	Actor
Step	Use Case
Association between Role and Step	Association between Actor and Use Case
Task	Interaction
Task in a Step	Interaction in a Use Case
Transition between Tasks in the same Step	Ordering between Interactions in the same Use Case
Guard on Transition	Constraint on Interaction
Alternative Path through a Branch	Alternative Path Description of a Use Case, or Extending Use Case

path of the branch may not contain tasks. The reason for this is that an extending use case only describes an *extension* to the original behavior of its base use case. It does not describe a *limitation* of the original behavior of its base use case. However, when an alternative path is chosen in a step, then the behavior of the basic path, from the moment this alternative path is chosen to the moment we return from the alternative path, must not be carried out. This describes a limitation of the behavior of a step that can not be expressed by an extend relation in the use case diagram.

A summary of the mapping from business process modeling concepts and use case modeling concepts is shown in Table 2.

The mapping described above is based on the assumption that each task is supported, and not automated or manual.

When a task is manual, it will be described as such in the use case of which it has become a part. However, when this results in a use case that is entirely manual, the use case no longer models interactions between the system and its users, and therefore, by definition, is no longer a use case. In this situation, the modeler may consider making the use case part of the use case that precedes or succeeds it in the business process model, or discarding the use case altogether.

When a task is automated, it will be described as such in the use case of which it has become a part. When this results in a use case that is entirely automated, again, the use case is, by definition, no longer a use case. Thus the options that apply to a situation in which a use case has become completely manual also apply in this case.

A more difficult situation arises when a use case is only fully automated or manual when certain decision criteria hold. In this case, the path of the use case that handles the situation that is not fully automated or manual can be described in the use case. The options that apply to the situation in which an entire use case has become automated or manual, apply again to the path that handles the situation that is fully automated or manual.

4.3. Validation of the Initial Mapping

We give a Z specification of the metamodels and the mapping below. The Z specification is derived from the metamodels by postulating a set for each class in the metamodel, and postulating a relation between two sets for each association between two classes in the metamodel. The semantic constraints are specified in the predicate part of the schemas that represent the metamodels. Since cardinality constraints are often recurring semantic constraints, we specified shorthands for them in a generic schema.

$$
\begin{array}{|l}
\hline
[X, Y] \\
\hline
tot : (X \leftrightarrow Y) \\
fun : (X \leftrightarrow Y) \\
sur : (X \leftrightarrow Y) \\
inj : (X \leftrightarrow Y) \\
\hline
tot(r) \Leftrightarrow \forall x : X \bullet \exists y : Y \bullet x \mapsto y \in r \\
fun(r) \Leftrightarrow \forall x : X;\ y_1, y_2 : Y \bullet \\
\qquad x \mapsto y_1 \in r \land x \mapsto y_2 \in r \Rightarrow y_1 = y_2 \\
sur(r) \Leftrightarrow \forall y : Y \bullet \exists x : X \bullet x \mapsto y \in r \\
inj(r) \Leftrightarrow \forall x_1, x_2 : X;\ y : Y \bullet \\
\qquad x_1 \mapsto y \in r \land x_2 \mapsto y \in r \Rightarrow x_1 = x_2 \\
\hline
\end{array}
$$

$[Object]$

$$Task, Branch, Step : \mathbb{P}\, Object$$
$$Role, Transition : \mathbb{P}\, Object$$

BusinessProcess

$$member : Step \leftrightarrow Task \cup Branch$$
$$responsible : Task \cup Step \cup Branch \leftrightarrow Role$$
$$source : Transition \leftrightarrow Task \cup Step \cup Branch$$
$$destination : Transition \leftrightarrow Task \cup Step \cup Branch$$

$$tot(responsible) \land fun(responsible)$$
$$tot(source) \land fun(source)$$
$$tot(destination) \land fun(destination)$$
$$tot(member) \land sur(member)$$
$$\forall\, t : Task \cup Branch;\ r : Role \bullet \exists\, s : Step \bullet$$
$$\quad s \mapsto t \in member \land s \mapsto r \in responsible \Leftrightarrow$$
$$\quad t \mapsto r \in responsible$$
$$\forall\, t : Transition \bullet \exists\, s : Step \bullet t \mapsto s \in source \Rightarrow$$
$$\quad (\neg\ \exists\, ta : Task \bullet t \mapsto ta \in destination) \land$$
$$\quad (\neg\ \exists\, b : Branch \bullet t \mapsto b \in destination)$$
$$\forall\, t : Transition \bullet \exists\, s : Step \bullet$$
$$\quad t \mapsto s \in destination \Rightarrow$$
$$\quad (\neg\ \exists\, ta : Task \bullet t \mapsto ta \in source) \land$$
$$\quad (\neg\ \exists\, b : Branch \bullet t \mapsto b \in source)$$
$$\forall\, b : Branch \bullet \exists\, t : Transition \bullet$$
$$\quad t \mapsto b \in destination$$
$$\forall\, b : Branch \bullet \#(source \rhd \{b\}) \geq 2$$
$$\forall\, s_1, s_2 : Step;\ t : Transition \bullet$$
$$\quad t \mapsto s_1 \in source \land t \mapsto s_2 \in destination \Leftrightarrow$$
$$\quad (\exists\, ta_1, ta_2 : Task \cup Branch;\ t' : Transition \bullet$$
$$\quad t' \mapsto ta_1 \in source \land t' \mapsto ta_2 \in destination$$
$$\quad \land$$
$$\quad s_1 \mapsto ta_1 \in member \land s_2 \mapsto ta_2 \in member)$$

The predicate part of the BusinessProcess schema expresses, apart from the cardinality constraints, the following properties successively:

1. if a task or branch is a member of a step, and this step has a certain responsible role, then the task or branch also has this responsible role. This property also applies vice versa;
2. if a transition originates from a step, then this transition cannot point to a task or branch;
3. if a transition points to a step, then this transition cannot originate from a task or branch;
4. a branch must have an incoming transition;
5. a branch must have at least two outgoing transitions;
6. if two steps are connected by a transition, then another transition must exist. This transition must have a branch or task in the fist step as its source, and a branch or task in second step as its destination. This property also applies vice versa.

$$Interaction, UseCase, Actor : \mathbb{P}\, Object$$
$$Association, Extend, Point : \mathbb{P}\, Object$$

UseCase

$$actor : Association \leftrightarrow Actor$$
$$usecase : Association \leftrightarrow UseCase$$
$$interaction : UseCase \leftrightarrow Interaction$$
$$succeeds : Interaction \leftrightarrow Interaction$$
$$extension : Extend \leftrightarrow UseCase$$
$$base : Extend \leftrightarrow UseCase$$
$$point_{EE} : Extend \leftrightarrow Point$$
$$point_{UE} : UseCase \leftrightarrow Point$$

$$tot(actor) \land fun(actor)$$
$$tot(usecase) \land fun(usecase)$$
$$tot(extension) \land fun(extension)$$
$$tot(base) \land fun(base)$$
$$tot(point_{EE})$$
$$tot(interaction) \land sur(interaction)$$
$$inj(interaction)$$
$$\forall\, u : UseCase;\ p : Point \bullet \exists\, e : Extend \bullet$$
$$\quad e \mapsto u \in base \land e \mapsto p \in point_{EE} \Leftrightarrow$$
$$\quad p \mapsto u \in point_{UE}$$
$$\forall\, i_1, i_2 : Interaction \bullet \exists\, u : UseCase \bullet$$
$$\quad u \mapsto i_1 \in interaction \land u \mapsto i_2 \in interaction$$
$$\quad \Rightarrow$$
$$\quad i_1 \mapsto i_2 \in succeeds^* \lor i_2 \mapsto i_1 \in succeeds^*$$

The predicate part of the UseCase schema expresses, apart from the cardinality constraints, the following properties successively:

1. A use case that acts as the base of an extension relation, must refer to the extension point of this extension relation. Also, if a use case refers to an extension point, then it must act as the base of the extension relation that refers to this extension point;
2. if two interactions are in the same use case, then one must succeed the other, either directly or indirectly.

Mapping

BusinessProcess

UseCaseDiagram

$$
\begin{array}{l}
s \mapsto r \in \mathit{responsible} \Leftrightarrow \\
\quad \exists\, a : \mathit{Association} \bullet \\
\quad a \mapsto r \in \mathit{actor} \land a \mapsto s \in \mathit{usecase} \\
\forall\, u_1, u_2 : \mathit{UseCase} \bullet (\exists\, e : \mathit{Extend} \bullet \\
\quad e \mapsto u_1 \in \mathit{base} \land e \mapsto u_2 \in \mathit{extension}) \Rightarrow \\
\quad (\exists\, t : \mathit{Transition};\ ta_1, ta_2 : \mathit{Task} \cup \mathit{Branch} \bullet \\
\quad u_1 \mapsto ta_1 \in \mathit{member} \land u_2 \mapsto ta_2 \in \mathit{member} \\
\quad \land \\
\quad t \mapsto ta_1 \in \mathit{source} \land t \mapsto ta_2 \in \mathit{destination}) \\
\forall\, u_1, u_2 : \mathit{UseCase} \bullet (\exists\, i : \mathit{Include} \bullet \\
\quad i \mapsto u_1 \in \mathit{base} \land i \mapsto u_2 \in \mathit{addition}) \Rightarrow \\
\quad (\exists\, t : \mathit{Transition};\ ta_1, ta_2 : \mathit{Task} \cup \mathit{Branch} \bullet \\
\quad u_1 \mapsto ta_1 \in \mathit{member} \land u_2 \mapsto ta_2 \in \mathit{member} \\
\quad \land \\
\quad t \mapsto ta_1 \in \mathit{source} \land t \mapsto ta_2 \in \mathit{destination}) \\
\forall\, i_1, i_2 : \mathit{Interaction} \bullet i_1 \mapsto i_2 \in \mathit{succeeds} \Rightarrow \\
\quad (\exists\, t : \mathit{Transition} \bullet \\
\quad t \mapsto i_1 \in \mathit{source} \land t \mapsto i_2 \in \mathit{destination}) \\
\quad \lor \\
\quad i_1 \mapsto i_2 \in \\
\quad (\mathit{source}^{\sim} \fatsemi ((\mathit{destination} \rhd \mathit{Branch}) \fatsemi \\
\quad (\mathit{Branch} \lhd \mathit{source}^{\sim})) \fatsemi \mathit{destination}) \\
u \mapsto i \in \mathit{interaction} \Leftrightarrow u \mapsto i \in \mathit{member} \\
t \mapsto t_1 \in \mathit{source} \land t \mapsto t_2 \in \mathit{destination} \land \\
\quad s \mapsto t_1 \in \mathit{member} \land s \mapsto t_2 \in \mathit{member} \Rightarrow \\
\quad t_2 \mapsto t_1 \in \mathit{succeeds}
\end{array}
$$

The predicate part of the Mapping schema expresses the following properties successively:

1. if a relation exists between a step and a role, then an association exits between the corresponding use case and actor. This property also applies vice versa;
2. if two use cases are connected by an extension relation, then a transition must exist in the business process model. This transition must have a task or branch as its source that represents an interaction in the base use case. Also, this transition must have a task or branch as its destination that represents an interaction in the extension use case;
3. if two use cases are connected by an include relation, then a transition must exist in the business process model. This transition must have a task or branch as its source that represents an interaction in the base use case. Also, this transition must have a task or branch as its destination that represents an interaction in the included use case;
4. if an interaction succeeds another interaction, then the corresponding tasks must either be connected directly by a transition, or indirectly via a number of branches;
5. if an interaction belongs to a use case, then the task corresponding to the interaction must belong to the step corresponding to the use case. This property also applies vice versa;
6. if there is a transition from a task to a task in the same step, then the interaction that corresponds to the second task must succeed the interaction that corresponds to the fist task.

We specified the mapping, by only mapping the associations in the metamodels. We can do this when we only consider associations, and set elements that participate in an association. For example, we only consider the actor 'administrative worker', if it has an association to one of the use cases.

We evaluate the mapping by deriving additional semantic constraints from the semantic constraints specified in the schemas above. If one of the derived semantic constraints represents an undesirable property of the mapping, the mapping has to be reconsidered.

The first derived semantic constraint is derived as follows:

$$
\begin{array}{l}
s_1 \mapsto t \in \mathit{member} \land s_2 \mapsto t \in \mathit{member} \\
\qquad \Rightarrow \quad \text{[rule of Mapping]} \\
s_1 \mapsto t \in \mathit{interaction} \land s_2 \mapsto t \in \mathit{interaction} \\
\qquad \Rightarrow \quad \text{[inj(interaction)]} \\
s_1 = s_2
\end{array}
$$

This constraint specifies that a task cannot occur in more than one step.

The second derived semantic constraint is derived as follows:

$$
\begin{array}{l}
u_1 \mapsto p \in \mathit{point}_{UE} \\
\qquad \Rightarrow \quad \text{[rule of UseCase]} \\
\exists\, e : \mathit{Extend} \bullet \\
e \mapsto u_1 \in \mathit{base} \land e \mapsto p \in \mathit{point}_{EE} \\
\qquad \Rightarrow \quad \text{[rule of } \land\text{]} \\
\exists\, e : \mathit{Extend} \bullet e \mapsto u_1 \in \mathit{base} \\
\qquad \Rightarrow \quad \text{[tot(extension)]} \\
\exists\, e : \mathit{Extend};\ u_2 : \mathit{UseCase} \bullet \\
e \mapsto u_1 \in \mathit{base} \land e \mapsto u_2 \in \mathit{extension} \\
\qquad \Rightarrow \quad \text{[rule of Mapping]} \\
\exists\, t : \mathit{Transition};\ u_2 : \mathit{UseCase} \bullet \\
\exists\, ta_1, ta_2 : \mathit{Task} \cup \mathit{Branch} \bullet
\end{array}
$$

$$u_1 \mapsto ta_1 \in \mathit{member} \wedge u_2 \mapsto ta_2 \in \mathit{member} \wedge$$
$$t \mapsto ta_1 \in \mathit{source} \wedge t \mapsto ta_2 \in \mathit{destination}$$

This constraint specifies that, if we choose to transform a Branch into an Extend relation, then a transition must exist. This transition must have a Task as its source that is a member of the Step that represents the base Use Case. It must have a Task as its destination that is a member of the Step that represents the extension Use Case. In other words: there is no point in specifying a Use Case as an extension to another Use Case, if this Use Case is never executed.

The third derived semantic constraint is derived as follows:

$$\exists\, a_1, a_2 : \mathit{Association} \bullet$$
$$a_1 \mapsto ac_1 \in \mathit{actor} \wedge a_1 \mapsto u \in \mathit{usecase} \wedge$$
$$a_2 \mapsto ac_2 \in \mathit{actor} \wedge a_2 \mapsto u \in \mathit{usecase}$$
$$\Rightarrow \quad \text{[rule of Mapping]}$$
$$u \mapsto ac_1 \in \mathit{responsible} \wedge u \mapsto ac_2 \in \mathit{responsible}$$
$$\Rightarrow \quad \text{[fun(responsible)]}$$
$$ac_1 = ac_2$$

This constraint specifies that a Use Case can be associated with at most one Actor.

The fourth derived semantic constraint is derived as follows:

$$u_1 \mapsto i_1 \in \mathit{interaction} \wedge u_2 \mapsto i_2 \in \mathit{interaction} \wedge$$
$$u_1 \neq u_2$$
$$\Rightarrow \quad \text{[rule of Mapping]}$$
$$u_1 \mapsto i_1 \in \mathit{member} \wedge u_2 \mapsto i_2 \in \mathit{member} \wedge u_1 \neq u_2$$

This constraint specifies that if Interactions are members of two different Use Cases, then the Tasks that represent the Interactions were in two different Steps. A trivial constraint, but it turned out to be the largest source of mistakes in the case studies. Therefore we thought it was useful to specify this constraint here.

5. The Transformation Procedure

From the mapping defined in the previous section, we derived a procedure for transforming a business process model into a use case diagram.

First, we create an actor for each role in the business process model.

Second, if this is not done already, we create steps around the tasks in the business process model. To create steps, we take the first task in the procedure model, and determine the tasks that: (1) can be reached from this task by transitions, (2) are assigned to the same role, and (3) do not have time passing between them. These tasks together form the first step. For each task that is directly connected by a transition to a task in the first step, we repeat the procedure. The procedure to create steps from tasks was studied in [17]. Two steps are connected by a transition, if and only if a transition exists from a task or branch in the first step, to a task or branch in the second step. The guards that exist on the transition that connects these tasks or branches are also guards of the transition that connects the steps.

Third, we create a use case for each step that we identified. We create an association between an actor and a use case when there is a relation between the role that corresponds to the actor, and a task that is in the step that corresponds to the use case.

Fourth, we describe the identified use cases in detail, by describing their interactions. To do this, we first determine the most common path of tasks and transitions through a step. We describe this path as the most common path of the corresponding use case. Next, we determine the alternative paths in a step. We describe these as alternative paths in the corresponding use case, or optionally as extension use cases.

Fifth, we restructure the use case diagram that results from applying the procedure. This can be done according to normal restructuring rules, like, for example, the ones described in [10]. However, when using the procedure that is described in this section, some constructs need special attention. Therefore, we will point out some restructuring actions that are normally necessary when creating use case diagrams from business process models.

From case studies, we found out that tasks are often described multiple times. The reason for this is that business process modeling techniques do not cater for reuse of tasks. However, in use case diagrams reuse is possible. Therefore, we search the resulting use cases for interactions that are defined more than once. We put these interactions in a separate use case, and we draw an include relation from the original use cases to this separate use case.

We also found situations in which tasks in a step are only carried out under certain conditions. This may cause use cases to start halfway their common path, or be aborted halfway their common path. To cater for this situation we can use one of the following three solutions depending on the concrete situation. We can either:

1. include the optional tasks in an alternative path description of the use case;

2. include the optional tasks in an additional use case and use this use case as an extension of the original use case;

3. include the optional tasks in an additional use case, and ignore the control information altogether.

According to its definition a use case delivers a result of value to its user. However, in the case studies we found situations in which users experienced that a use case only delivered a result of value when it was combined with another use case. This situation can be solved by verifying if each use case delivers a result of value to its users, and, if not, by combining use cases.

It may be so that one of the use cases that are combined, is only carried out under a certain condition. A use case is only carried out under a certain condition, when the step that corresponds to it, is only the destination of a transition that has a guard on it. If this is the case for one of the use cases that we combined, then we can choose to treat the steps as two different use cases after all. Optionally, we can relate the use cases with an extension relation.

As an example of the execution of the procedure, consider the use case diagram shown in figure 1. We derived this diagram from the business process model shown in figure 3, by applying the procedure. The roles in the activity model were mapped to actors in the use case diagram. Steps were created around the tasks 'enter client data', 'enter mortgage data', and 'check credit', around the task 'inform client about rejection', around the tasks 'advise client', and 'process changes', and around the tasks 'check credit', and 'draw up offer'. These steps were mapped to use cases in the use case diagram. The associations between the roles and the steps were mapped to associations between the corresponding actors, and use cases. While restructuring, the 'check credit' interaction was put in a separate use case, because it was defined twice. We drew an include relation to the 'check credit' use case from the use cases of which the 'check credit' interaction originally was a part. Also, the use case consisting of the interaction 'inform client about rejection', was eliminated, because it was completely manual.

6. Case Studies

To evaluate the procedure, we applied it in two case studies. The studies took place in the mortgage processing departments of two internationally operating banks. We will discuss the two case studies in the following subsections.

6.1. Case Study: Evaluation of Correctness

In the first case study, we focused on the evaluation of the correctness of the technique. In this case study, we applied the technique in practice, and evaluated the quality of the resulting use cases as compared to use cases that were constructed by performing interviews. From the results we derived some recommendations for the technique.

In this case study, we investigated a total of 6 business processes. From these business processes we derived 42 use cases. For all use cases together, 10 alternative paths were found. When evaluating the use case diagrams, a number of improper use case constructions were identified.

Of 2 use cases some of the interactions were defined more than once. 3 use cases were completely redundant. This type of construct results from the lack of reusability of steps and tasks in business process models. Therefore, when a task or step is carried out twice in a business process, it has to be defined twice. Hence, when applying the procedure it results in use cases being defined twice.

6 use cases had the option to be performed only in part. 2 of these use cases could be stopped halfway the use case description, and 4 could be started halfway the use case description. This situation arises when part of the work only has to be performed under a certain condition, and thus a step is left or started halfway the common path of tasks.

according to the users of the system, 12 use cases only delivered a result of value in combination with another use case. This situation usually occurs when the processing of an application can only continue until certain information is known that has to be obtained from an external entity. When, for example, assessing the creditworthiness of a client, information is requested from various institutions. Although the user has to wait for the response of one institution, before continuing with the next, he experiences checking the creditworthiness of a client as a single task.

The choice to describe use cases belonging to the same business process in one use case diagram suggests that each system is built for only one business process. Also, it suggests that only one system is built for each business process. However, this is rarely the case. A client database, for example, is usually used by all primary business processes. Therefore, once a use case diagram is drawn, there must be evaluated which use cases will be implemented by which system. This decision will be based on what systems already exist to support other business processes. When we decide that a use case is partly going to be implemented by one system, and partly by another system, we may decide to split the use case. Eriksson and Penker describe a procedure for doing this in [8].

From this case study we concluded that the procedure produces use cases that can be used as a specification for software to support business processes. To cater for the constructions defined above, we integrated them into the restructuring step of the procedure.

6.2. Case Study: Evaluation of Benefits

In the second case study, we focused on the evaluation of the benefits of the technique. In this case study, we applied the technique in practice, and evaluated the time it took to create use case diagrams with the proposed procedure, as compared to the time it took to create use case diagrams

without the procedure. From the application of the technique in this case study, we came up with some figures to support the assumption that the technique speeds up the process of constructing use cases.

In this case study, we observed a project. In this project two teams worked separately on the development of business process models, and the development of use case diagrams respectively. The estimated development time for the business processes was 4800 hours, and the estimated development time for the use case diagrams was 8800 hours. After inspecting some of the resulting models, we noticed that the use case diagrams did not contain any information that was not in the business process models. Therefore, it was not necessary to have a separate team working on the development of use case diagrams.

7. Conclusion

In this paper, we introduced a procedure to transform business process models into UML use case diagrams. First, we created a mapping between the metamodels of use case diagrams and business process models. Then we created a procedure that complies to this mapping. We have shown in a case study that the application of the procedure results in use case diagrams that can serve as a basis for further system development. In another case study we have shown that the procedure significantly speeds up the specification of use case diagrams.

The study described in this paper is related to the fields of requirements, and method engineering. Much work is done in both fields to study how models of different types can be generated from each other. In the areas of requirements, and method engineering, research is done to how business processes can be described using use case diagrams [15, 10, 11, 16]. We, however, do the opposite, and use business processes to derive use cases. Eriksson and Penker describe a procedure to derive use cases from business processes that is similar to ours in [8]. However, this procedure is less detailed.

The use case diagrams that result from the application of the procedure have limitations.

First, the use case diagrams specify a typical information system, in the sense that they do not specify control information that is above step level. For example, the use case diagram from figure 1 does not specify whether client data must be entered before mortgage data or not. When we want to specify this type of control information, a service has to be built on top of the information system that is specified by the use case diagrams. In further research we will investigate the relation between the information system and tools that enforce control information, like, for example, workflow engines.

Second, the use case diagrams are as detailed as the business processes from which they have been derived. A use case may, for example, contain the interaction 'enter client data', or the interactions 'search client based on name and date of birth', if the client was not found 'create a new client', and 'enter name, address, and date of birth of the client'. The second set of interactions provides a more precise specification than the first set of interactions. Therefore, depending on the amount of detail specified in the business processes, it may be necessary to perform further interviews with system users to detail the use case diagrams further. The extend to which this is a problem can not be assessed from two case studies, because statistical data must be gathered to assess this.

The approach we used also has as a limitation: it does not provide absolute proof that the procedure is correct. The reason for this is that the formal specification of the procedure that we use as proof merely allows us to validate the procedure. It cannot provide a formal proof of the correctness of the procedure. To assess the correctness of the procedure further, we must apply it in more case studies, and gather statistical data from its application.

Acknowledgements

We would like to thank Luís Ferreira Pires, and Giancarlo Guizzardi for their valuable comment on a previous version of this paper.

References

[1] B. Boehm. *Software Engineering Economics*. Prentice Hall, Englewood Cliffs, 1981.

[2] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, MA, 1999.

[3] A. Cockburn. Structuring use cases with goals. *Journal of Object Oriented Programming*, 10(7):35–40, 1997.

[4] A. Cockburn. Structuring use cases with goals. *Journal of Object Oriented Programming*, 10(8):56–62, 1997.

[5] T. Davenport. *Process Innovation*. Harvard Business School Press, Boston, MA, 1993.

[6] R. Dijkman, L. Ferreira Pires, and S. Joosten. Calculating with concepts: a technique for the development of business process support. In A. Evans, R. France, A. Moreira, and B. Rumpe, editors, *Proceedings of the UML 2001 Workshop on Practical UML Based Rigorous Development Methods: Countering or Integrating the eXstremists*, volume Proceedings 7 of *Lecture Notes in Informatics*, pages 87–98, Bonn, 2001. Gesellschaft für Informatik.

[7] M. Dumas and A. ter Hofstede. UML activity diagrams as a workflow specification language. In M. Gogolla and C. Kobryn, editors, *Proceedings of the UML 2001 Conference on Modeling Languages, Concepts and Tools*, volume 2185 of

Lecture Notes in Computer Science, pages 76–90, Berlin, 2001. Springer.

[8] H.-E. Eriksson and M. Penker. *Business Modeling with UML: Business Patterns at Work*. Wiley, New York, 2000.

[9] M. Fowler and K. Scott. *UML Distilled: Applying the Standard Object Modeling Language*. Addison-Wesley, Reading, MA, 1997.

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

[11] I. Jacobson, M. Ericsson, and A. Jakobson. *The Object Advantage: Business Process Reengineering with Object Oriented Technology*. Addison-Wesley, Reading, MA, 1994.

[12] I. Jakobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, Workingham, 1992.

[13] S. Joosten, G. Aussems, M. Duitshof, R. Huffmeijer, and E. Mulder. *An Emperical Study about the Practice of Workflow Management*. University of Twente, Enschede, 1994.

[14] S. Joosten and S. Purao. A rigorous approach for mapping workflows to object oriented is models. To appear in: Journal of Database Management.

[15] S. Nurcan, G. Grosz, and C. Souveyet. Describing business processes with a guided use case approach. In *Proceedings of the 1998 Conference on Advanced Information Systems Engineering*, volume 1413 of *Lecture Notes in Computer Science*, pages 339–362, Berlin, 1998. Springer.

[16] Object Management Group. OMG unified modeling language specification version 1.4. Internet: http://www.omg.org, 2001.

[17] J. van Beek. Generation workflow: How staffware workflow models can be generated from protos business models. Master's thesis, University of Twente, 2000.

[18] W. van Dommelen, S. Joosten, and M. de Mol. Comparative study to aids for managing business processes (in dutch: Vergelijkend warenonderzoek hulpmiddelen beheersing bedrijfsprocessen. Technical report, Department of Finance, The Hague, 1999.