Transformation of Use Cases to EPC Models

Daniel Lübke

Leibniz Universität Hannover, FG Software Engineering
daniel.luebke@inf.uni-hannover.de

Abstract: Within the requirements phase of many projects, functional requirements are often documented as Use Cases. Within SOA projects, however, these Use Cases are not sufficient since they do not represent a global control flow resembling the one of a business process - an integral aspect of a SOA. Instead they are written from the point of view of a single actor. This makes an additional step necessary: Converting the Use Cases to business processes, which is – if done manually – a tremendous task. To reduce this effort wasted on conversion, this paper proposes a method for generating business processes – expressed as EPC – as from Use Cases. Thereby, the need for extensive business process modelling after gathering the requirements is eliminated.

1 Introduction

Service-oriented Architecture (SOA) [EMPR05] is an emerging architectural style for organising large information systems in enterprises. SOA aims to integrate software systems by exposing functionality through so-called services which can be composed later on. The composition reflects the business processes the enterprise wants to support. Therefore, business process descriptions become an integral part of the requirements SOA projects need. As has been shown [ZM05], business processes in EPC notation can be used to semi-automatically generate compositions modelled using the Business Process Execution Language (BPEL) [ACD+03]. Additionally, they can be annotated to automatically derive user interfaces which can be used in SOA applications [LLSG06].

However, prior to the project's start, business processes might not have been described yet, and if they will certainly lack specifics needed for implementing information systems on top of them: Besides the control flow through an organisation, more information about roles' needs, e.g. information concerning their background, their interaction with the system, is necessary for software development. Especially descriptions of user interaction with the system are normally missing in business process descriptions.

The missing details, which are not documented in any business process, are normally gathered in software projects using software engineering methods - more precisely requirements engineering techniques. A common technique for describing requirements for user interactive systems are Use Cases [Coc05]. Use Cases are freely-written text with only some structure, like tabular templates. They can capture requirements from the point of view of a main actor interacting with the system achieving one goal per Use Case. Therefore, the system is decomposed into many small interaction scenarios. These scenar-

ios together resemble the whole system's functionality. However, requirements captured this way are very detailed but at the same time spread over many Use Cases, making their handling very difficult. UML Use Case Diagrams [Gro04] try to bring back overview by graphically showing Use Cases and respective actors. Use Case Diagrams focus on the relationships between Use Case and Actor and between several Use Cases by their include and extend associations. However, Use Case Diagrams are not designed to illustrate the control flow between Use Cases. As such they are not suited to close the gap to the business processes. Thus, a direct business process-like and control flow centered notation is not directly available.

In order to capture the details needed for describing user interface requirements Use Cases are an appropriate option. However, to derive business processes they must be transformed into a business process model. Therefore, this paper proposes a transformation of a number of small but detailed Use Cases into a business process model. While the transformation is possible to all business process languages, Event-driven Process Chains (EPCs) are used in this paper because they can better express textual conditions using Events than UML activity diagrams with transition conditions. Furthermore, transformation techniques for converting EPC structures to BPEL compositions are available and user interface generation capabilities have been added.

In the next section of this paper Use Cases are introduced in more detail, defined and a meta-model is constructed. Afterwards, the same is done for EPCs. The third section describes the steps for transforming Use Cases to EPC models. Within the fourth section a prototype implementation is shown before an example is presented. Finally, the paper presents some related work and conclusions and an outlook is given.

2 Use Cases

2.1 Definition

According to Cockburn "a use case captures a contract between the stakeholders of a system about its behaviour. The use case describes the system's behavior under various conditions as the system responds to a request from one of the stakeholders, called the primary actor" [Coc05]. Therefore, Use Cases are part of system requirements, defined as a (logical) contract and describe the system interaction with the main actor, i.e. a user. However, this definition does not mention the structure of a Use Case.

Adolph and Bramble state that "Use Cases are simply stories about how people (or other things) use a system to perform some task" [AB02, p. 1]. Moreover, they have the opinion that the semi-formal nature helps structuring the requirements while at the same time not forcing the Use Case writer into too much formalism. But this definition omits a concrete Use Case structure, too.

This vagueness led to misconceptions what a Use Case actually is or looks like. Some people think that a Use Case actually is a UML Use Case Diagram which is definitely wrong. Normally, Use Cases are written in free text. This can be either without any form

or - most commonly - in a tabular form which serves as a template for all relevant Use Case attributes. An example Use Case in tabular form can be seen in figure 1.

Use Case	#3: Thesis Supervisor hands out topic
Primary Actor	Thesis Supervisor
Stakeholders	Thesis Supervisor: wants to hand out topic easily and without much paperwork Student: wants to receive topic quickly Secretary: wants easy to use/read forms for completing registration
Minimal Guarantees	A topic is only handed out once at a time
Successs Guarantees	Student knows topic Supervisor knows all needed adminstrative information of the student
Preconditions	Student has achieved at least 80% of credit points Student has clearance from Academic Examination Office
Triggers	Student wants to sign up for a topic
Main Success Scenario	1. Supervisor checks whether the topic is still available or not 2. Supervisor reserves topic for student 3. System updates list of current thesis 4. Supervisor confirms thesis topic and student's information to the Academic Examination Office 5. System sends confirmation to student, supervisor and Academic Examination Office
Extensions	1a If topic is not available anymore, then EXIT

Figure 1: An Example Use Case

2.2 Usage of Use Cases

Use Cases are often used for specifying functional requirements of a software system concerning the user interaction. Use Cases provide a very powerful mechanism for this kind of usage due to the following reasons:

- **Readability:** Use Cases are written in normal language; graphical notations and figures are only used for clarifying the text. Therefore, all kinds of users can read, understand and comment on the requirements from their point of view. This is a very important property of Use Cases since user feedback and collaboration is very important in the requirements phase of a project.
- **User centric:** Use Cases are written from the point of view of an end-user of a system. Therefore, specifics can be discussed independently and directly with the corresponding person(s), e.g. in interviews, without being distracted by additional details not relevant for the actual person.
- **Template Support:** Tabular Use Cases are in themselves a guidance what requirements must be specified: Necessary and helpful attributes, like preconditions, guarantees and extensions, which are often neglected in textual requirements documents, are clearly visible and attracting the necessary attention.

- **Recognition of Error Conditions:** Use Cases encourage thinking about error and abnormal conditions [AB02, p. 2]. These are often neglected areas within requirements specifications and Use Cases are a good way to close this gap.

However Use Cases can have some drawbacks if not embedded correctly into the project:

- **Over-Specification of Requirements:** Use Cases can be very detailed. This can be beneficial for main functions of a system but can be wasted effort in non-critical functions.
- **Missing Overview and Global Control Flow:** Use Cases themselves are not only very detailed but also from the point of view of a single user. It can therefore be a tremendous task to collect and arrange use cases based on their prerequisites in an easy way which allows navigation and an explicit control flow between the Use Cases.
- **Missing Non-Functional Requirements:** Use Cases only describe the behaviour of a system, i.e. mainly the control flow. Non-functional properties, like performance and security requirements cannot be efficiently dealt with, although there are approaches like Misuse Cases [Ale03] and aspects for integrating them into Use Cases [AC03].

Cockburn lists some template variation for all kinds of projects concerning different required detail levels. This includes a template for business process modelling. Although Use Cases are often associated with object-oriented software development, they do not depend on any architectural style. Thus, they can be beneficial within SOA projects as well, as long as the system interacts with the user.

2.3 Meta Model

While Use Cases are a semi-formal technique which is normally used within word processors or spreadsheets, one can define a meta-model for a specific template.

People concerned with a specific functionality of a system in general can have two interests: They may be the users of the system in which they need to be obviously involved in narrowing down the requirements because they are directly concerned. Such users are called **actors** in a system. Furthermore, an actor can also be an involved computer system, like the one to be developed. All actors directly interact with the system or are the system itself.

Besides the actors other people are interested in the system: For example, the management financing the system and optimising workflows may have completely different interests than the users. Those people are called **stakeholders**. The difference between actors and stakeholders is especially obvious (and extreme) in the case of new systems replacing some of the existent workers. Stakeholders are written down with their interests in Use

Cases as well, as has been shown in the example. Because all human actors have interests in the system they operate (like usability etc.) all human actors are stakeholders for this particular Use Case as well.

The functionality represented by a Use Case is modelled as a so-called **main scenario** which is performed whenever the **trigger** occurs and if all **preconditions** are met. The scenario itself consists of **steps** which are performed by an actor. The actor can do an action which can influence or trigger other actors (like the system) or business objects (like fill out an order). These elements are the **objects** of a step in the scenario.

The main scenario is the success case of a Use Case in which each step is completed successfully. For **error conditions and other exceptions** the scenario can be split up. Each step can therefore have **extensions**. Each extension in itself is a mini-scenario with steps resulting in a recursive structure. If an extension has been performed it can either exit the Use Case or jump back to a step in the upper scenario.

Steps can reference other Use Cases for improving clarity and partitioning the requirements. If a reference to a Use Case is made from the main scenario the Use Case is said to include the other one. If it is referenced within an extension the Use Case is said to be extended by the other one.

Furthermore, certain **guarantees** must be fulfilled by a Use Case even if something goes wrong. These can be transactional guarantees, like “data is consistent all the time” or other things which must be fulfilled at all times and even in case of failures. These are modelled as **minimal guarantees** in contrast to the **success guarantees** (sometimes also called postconditions) which are achieved by a successful execution of a Use Case.

Use Case’s precondition, minimal guarantee and success guarantee are given as a set of expressions. All expressions are formulated as free text and are implicitly joined by a logical and. If two expressions are given as a precondition the precondition is satisfied if both conditions evaluate to true. In the context of this paper, it does not matter in which form (e.g. as predicates) these expressions are given. They will only be compared for equality and not for implication.

Popular templates match this meta-model. They can therefore not express parallel control flow within a Use Case. Instead parallel activities must be modelled and written down in different Use Cases. Some templates denote the actor of a step separately by having a field for each the actor and the action. This shall encourage the usage of active sentences in order to improve clarity. In the following, access to the step’s actor is assumed.

The complete meta-model modelled as a UML diagram is shown in figure 2.

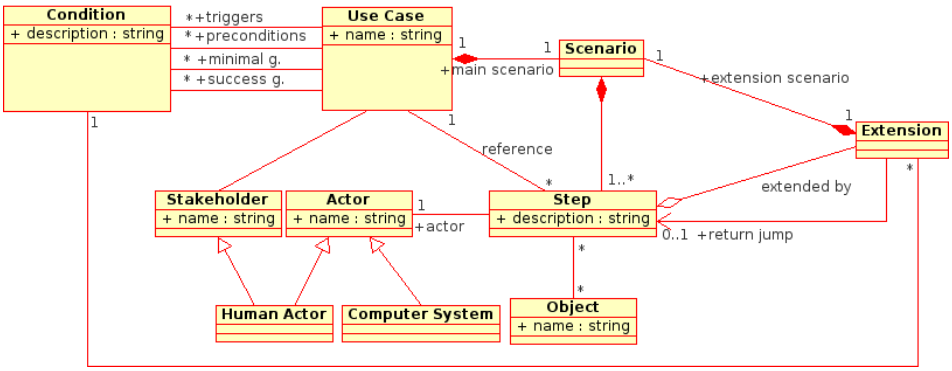


Figure 2: The Use Case meta model

3 Event-driven Process Chains

3.1 Definition

Event-driven Process Chains (EPCs) as defined in [NR02] are a graphical model for representing business processes from an organisational point of view.

While business processes do not contain information about non-functional requirements, they do offer a coarser-grained, organisational view on the functions of a system. This means that business processes actually have strengths (global control flow and organisational perspective) where Use Cases have some of their limitations. While personalised views have been proposed [GRvdA05], such views are not as detailed as Use Cases and are not sufficient for software development. Therefore, EPCs profit as well from integration with Use Cases.

EPCs represent a business process using two main elements: events and functions. **Events** are symbolising an organisational state. They trigger functions which represent an action performed within the organisation. A **function** alters the organisational state and therefore a function results in a new event which can trigger a new function and so on. To symbolize which function results in which event and which event triggers each function these elements are connected using arcs. For using the boolean operators AND, OR and XOR, connectors can be placed in an EPC. They can be used as splits (one incoming arc and many outgoing arcs) or as joins (many incoming arcs and one outgoing arc). Furthermore, modelling many elements like organisational roles, business objects, other operators etc. have been proposed in several EPC extensions.

3.2 Meta Model

EPCs have been formalised by Nüttgens and Rump in [NR02]. There are certain elements an EPC **Process** can consist of: **Events**, **Functions**, **Process Indicators** and **Connectors**. Connectors are divided into **Splits** and **Joins**. Splits may only have one incoming but many outgoing arcs, i.e. associations to other elements while joins may have many incoming but only one outgoing arc. Additionally, functions can be refined by sub-processes.

A simplified meta-model is shown in figure 3. Prominently missing for conserving clarity are the restrictions for the associations, e.g. functions may only be connected to events or connectors which are ultimately followed by events. These restrictions are fully discussed in [NR02].

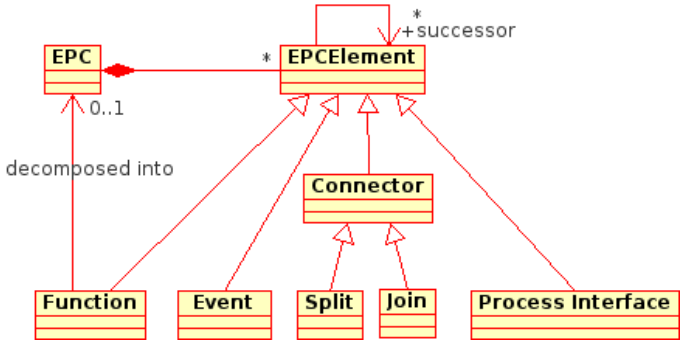


Figure 3: Simplified EPC meta model

4 Transformation Guidelines

The transformation of a set of Use Cases to an EPC repository is done in two steps: First, each Use Case is transformed to an EPC resulting in a set of EPCs. In the second step, the EPCs must be joined to a system of connected EPCs representing the whole process. While the first step is straight forward using some rules presented in section 4.2, there are two strategies for joining the EPCs. These strategies are presented in section 4.3.

4.1 Use Case Constraints

The transformation approach outlined below will work for Use Case models satisfying some constraints. Most important, our approach will not work with Use Cases having cyclic include and extend relationships.

Furthermore, our approach requires pre- and postconditions to equal literally. Since Use

Cases – including the pre- and postconditions – are written in natural language, no automatic evaluation can be made whether conditions can be satisfied by other means. Formulating conditions in a more formal way would run contrary to the Use Case goal of being understandable both by the stakeholders and the developers. Therefore, conditions should be formulated as small free text fragments. Use Case editors, like the Use Case DODE [Chr06], can offer help in consistently editing Use Cases in this regard by offering already used conditions e.g. in combo boxes. Thereby manual checks and typing errors are eliminated. Additionally, checks for literal similarity of conditions can further improve the Use Case model in this regard.

4.2 Use Case Transformation

For transforming Use Cases to EPCs, a set of rules is applied to the set of Use Cases (which are illustrated in figure 4:

1. **Include** and **extend** relationships of Use Cases are removed by copying the scenario and extensions of the referenced Use Case. Thereby, a flat Use Case model without any relationships is derived.

2. **Preconditions** and **triggers** are realised as events since their conditions fulfill the same role as events do. Because all preconditions must be met and the trigger must occur in order to start the Use Case, all events are joined using an AND join in the EPC if this rule results in more than one event. The first step in the main scenario is inserted after the AND join as an EPC function.

3. All steps of the **main scenario** are mapped to a linear control flow in an EPC. Each step is mapped to an EPC function. Functions are connected by using trivial OK-events. The step's actor becomes the role responsible for the created EPC function. Objects normally cannot be easily extracted from the Use Case templates and are therefore not handled by this algorithm.

4. **Success Guarantees** are like the preconditions and triggers conditions concerning a system which are to achieved. They are mapped to EPC events which are inserted after the last function of the main scenario. Since all guarantees must hold after completion, all events (in case of multiple guarantees) are connected using an AND split.

5. **Minimal Guarantees** are discarded. These guarantees normally represent non-functional requirements which cannot be visualised using EPCs. Since they must be valid before, during and after the Use Case they do not change the system at all.

6. **Extensions** are introduced using an XOR connector. After the proceeding function an XOR split is introduced which splits into the OK-event and an start event for each extension. A step with 2 extensions therefore becomes a function followed with an XOR split to 3 paths.

7. All **Extension steps** are handled like steps in the main scenario. **Extensions to extension steps** are handled recursively using these rules.

8. **Jumps** typically occurring from one extension step to a step in the main scenario are realised using XOR joins. A join is introduced before the function representing the step which is the jump target.

Below the algorithm is presented in pseudo code. All possible joins and splits are introduced first. If they are not used anywhere (e.g. by steps) these connectors are removed afterwards. The EPC is supposed to create an element on the fly if one is accessed in the EPC which is non-existent due to forward-jumps from the extensions back into the main scenario in order to make the algorithm simpler:

```
function ConvertUseCaseToEpc(UseCase)
  let EPC = new EPC(UseCase.Name);

  let UseCase = MakeFlatUseCase(UseCase)
  let LastElement = HandleTriggersAndPreconditions(
                      UseCase, EPC)
  let LastElement = HandleScenario(
                      UseCase.Scenario, LastElement)
  HandleGuarantees(UseCase, EPC, LastElement)
  EPC.RemoveUnnecessaryConnectors()

  return EPC
end function
```

To make the algorithm better understandable, it is split up into subroutines. The triggers and preconditions are processed first:

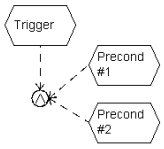
```
function HandleTriggersAndPreconditions(UseCase, EPC)
  let AndJoin = EPC.CreateAndJoin('TriggerAnd')

  for each C in UseCase.Preconditions
                  + UseCase.Triggers
    let Event = EPC.CreateEvent(C.Name)
    Event.ConnectTo(AndJoin)
  next C

  return AndJoin
end function
```

Afterwards, the main scenario is converted. In this step, extensions are handled recursively:

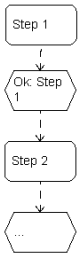
```
function HandleScenario(Scenario, LastElement)
  for each Step in Scenario
    let XorJoin = EPC.CreateXorJoin('Join'+Step.Name)
    LastElement.ConnectTo(XorJoin)
    let Function = EPC.CreateFunction(Step.Name)
    Function.AddRole(Step.Actor)
    XorJoin.ConnectTo(Function)

    let XorSplit = EPC.CreateXorSplit('Split'+Step.Name)

    for each Extension in Step.Extensions
      let Event = EPC.CreateEvent(
                    ExtensionStep.Condition)
      XorSplit.ConnectTo(Event)

      HandleScenario(Extension.Scenario, Event)

      if Extension.Jumps then
        Event.ConnectTo(
          EPC.Element['Join'+ExtensionStep.ReturnJump])
      end if
    next Extension
    let LastElement = EPC.CreateEvent('OK ' + Step.Name)
    XorSplit.ConnectTo(LastElement)
  next Step
end function
```

Finally, the success guarantees are appended to the end of the main scenario. Because each Use Case must achieve at least one user goal, each Use Case has at least one success guarantee:

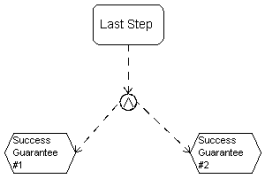
```
function HandleGuarantees(UseCase, EPC, LastElement)
  let AndSplit =
    EPC.CreateAndSplit('SuccessGuaranteesSplit')
  LastElement.ConnectTo(AndSplit)

  for each SG in UseCase.SuccessGuarantees
    let Event = EPC.CreateEvent(SG.Name)
    AndSplit.ConnectTo(Event)
  next SG
end function
```



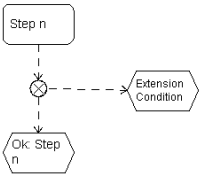
(a) Rule 2: Triggers and Preconditions



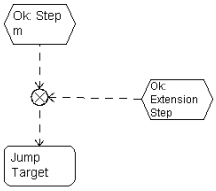
(b) Rule 3 & 7: Steps



(c) Rule 4: Success Guarantees



(d) Rule 6: Extensions



(e) Rule 8: Jumps

Figure 4: Results of Applying Transformation Rules for Use Case attributes to EPCs

4.3 Joining EPCs

After transforming each Use Case into an EPC, the transformation must proceed joining the single EPCs to a comprehensive EPC model. For doing this, two options are available: Either a **single, large EPC model** is built or the processes are joined using **process interfaces**.

The Use Cases are connected by using the preconditions and success guarantees. These are expressed as a set of conditions. Both join strategies work by comparing these conditions literally. A precondition can be satisfied by a literally matching success guarantee.

The first strategy joins the EPCs by looking for identically named events and constructing one large EPC. For each event its occurrence as start and end events of EPCs is looked up. All end events are replaced by an OR join, a single event and an AND split. The OR join is connected with the arcs from the removed end events. Afterwards, all start events are removed and their arcs connected to the AND split. The procedure is illustrated in figure 5.

The result of applying this strategy is a single and large EPC model. If it is not too large, the model clearly shows the organisational control flow. However, it is hard to identify the former Use Case model. Furthermore, the model can quickly become too complex. This makes manual rework necessary, e.g. refactoring the EPC using hierarchical decomposition.

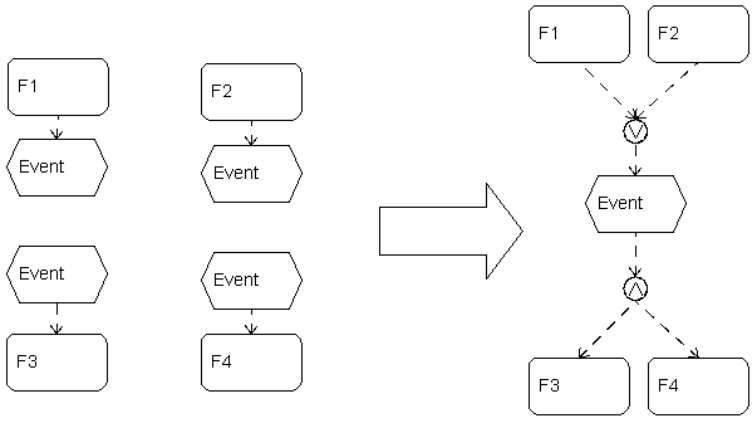


Figure 5: Joining resulting EPCs to one EPC model by Events

The second possibility for joining the EPCs is to leave all the generated EPCs intact by adding process interfaces. Thereby each process remains in an acceptable size. The process interfaces are named after the original Use Cases. This is illustrated in figure 6.

Both strategies can be beneficial in different scenarios: For communicating with isolated stakeholders and users, a small EPC with process interfaces shows only the details necessary for visualizing one Use Case. However, the organisational control flow is not as

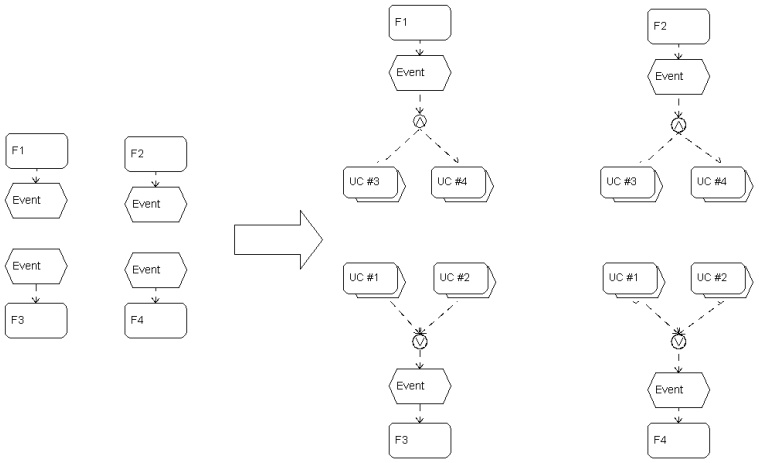


Figure 6: Joining resulting EPCs using Process Interfaces

visible as process and composition designers might like it. The organisational control flow is better visualised by a large model. Since both models are automatically (and therefore with little effort) generated, both can be used for visualising the process to different stakeholders as long as no changes are made which would render the two models inconsistent.

5 Prototype Implementation

Based on the outlined transformation rules, a prototype application has been developed [Die06]. The prototype consists of an XSLT stylesheet and a Java GUI front-end. The stylesheet transforms Use Cases stored within XML documents into EPCs stored in EPML [MN05].

Since there is no standardised XML format for storing Use Cases, an XML schema based on the described meta-model has been developed. It can store multiple Use Cases in different sets accompanied with all relevant project information described in the meta model.

The main logic of the transformation tool is contained in the XSLT stylesheet which implements the rules presented in section 4 and joins the EPC using process interfaces. The stylesheet can be applied to a Use Case project and will generate EPML. However, no "nice" layout of the EPCs is done; this task is left to other tools, in this case the Java front-end. The benefit from this approach is that the stylesheet can be integrated into other applications as well, which will need to lay out the Use Cases differently. For example, direct integration into Use Case editors would be beneficial, which can show the EPC for validation purposes in a small area of the screen.

The Java GUI makes using the XSLT stylesheet easy: After the user has opened a Use

Case file, the Use Cases can be opened in a tabular view as illustrated in the left hand side of the screenshot seen in figure 7. If the user chooses to transform a Use Case project, the tool applies the XSLT stylesheet and generates an EPML file. The EPML file is loaded and layouting mechanisms provided by JGraph and extended with own implementations lay out the EPC file. Afterwards the EPCs can be displayed on the right hand side as can be seen in the screenshot. The tool is able to track elements of an Use Case to elements of an EPC: Whenever the user activates a Use Case element which is visible in the EPC, the corresponding EPC element is highlighted.

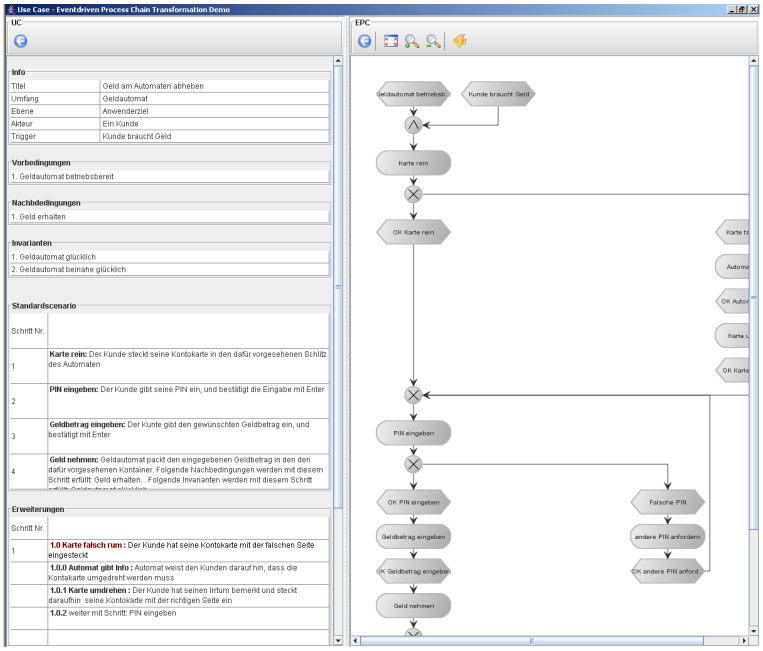


Figure 7: Screenshot of the Prototype Application

6 Example Use Cases

To illustrate the transformation, an example from a fictional university is presented in this section. The to be developed software is a system supporting students' theses registration. The simple registration process is documented in four Use Cases as illustrated in figure 8:

1. **Student applies for Thesis:** This Use Case describes the step a student has to make in order to apply for a thesis. The Use Case is triggered by the student's wish to write a thesis. As a result in case of success, the application is submitted to the Academic Examination Office.

Use Case	#1: Student applies for Thesis		
Primary Actor	Student		
Stakeholders	Student: wants to apply easily Secretary (Academic Examination Office): wants easy to use/read forms for further handling registration		
Minimal Guarantees	none		
Successs Guarantees	Application is submitted		
Preconditions	none		
Triggers	Student wants to write thesis		
Main Success Scenario	1	Student	fills out form with personal data
	2	Student	submits form to Academic Examination Office
Extensions	none		

Use Case	#2: Academic Examination Office approves Thesis		
Primary Actor	Student		
Stakeholders	Secretary (Academic Examination Office): wants easy to use/read forms for further handling registration Manager (Academic Examination Office): wants short handling times		
Minimal Guarantees	Student's data are handled according to regulations		
Successs Guarantees	Student may write Thesis		
Preconditions	none		
Triggers	Application is submitted		
Main Success Scenario	1	Secretary	checks if student has 80% of Credit Points
	2	Secretary	approves application
Extensions	1a If Student has less than 80% of Credit Points then Secretary denies Application		

Use Case	#3: Student selects Topic		
Primary Actor	Student		
Stakeholders	Student: wants to have interesting topic		
Minimal Guarantees	none		
Successs Guarantees	Student has picked a Topic		
Preconditions	none		
Triggers	Student wants to write thesis		
Main Success Scenario	1	Student	looks trough list of available topics
	2	Student	chooses most interesting topic
	3	Student	asks Supervisor to get the topic
Extensions	none		

Use Case	#4: Supervisor approves Topic		
Primary Actor	Supervisor		
Stakeholders	Supervisor: wants no paperwork Secretary (Academic Examination Office): wants easy to use/read forms for further handling registration		
Minimal Guarantees	none		
Successs Guarantees	Student has Topic		
Preconditions	Student may write Thesis Student has picked a Topic		
Triggers	none		
Main Success Scenario	...		
	1	Supervisor	hands out Topic
	...		
Extensions	(left out)		

Figure 8: Use Cases for a Thesis Registration System

2. **Academic Examination Office approves Thesis:** In the next Use Case the Academic Examination Office checks if the student fulfills all prerequisites for writing a thesis. In this case, 80% of Credit Points need to be already earned. If successfully, the application is approved.

3. **Student selects Topic:** If the application is approved, the student has to choose a department and a topic for his or her thesis.

4. **Supervisor approves Topic:** Finally, the supervisor has to approve the topic the student wants to write in.

The Use Cases satisfy the given requirements: For simplification these Use Cases do not have include or extend relationships (which are therefore not cyclic) and all conditions are named consistently.

Therefore, these Use Cases can be transformed using the outlined rules. The first and second Use Cases are connected by the "Application is submitted" event. The second and forth Use Cases are connected by the "Student may write Thesis" event while the third and forth Use Cases are connected by the "Student has picked a Topic" event. The resulting EPC is shown in figure 9.

Use Cases as presented here are easy to write and similar ones have been used internally in a student project. Using the generation capabilities, possible parallel execution paths have been identified which the process participants were not aware of because they were used to the (serialised) process they were part of for years.

7 Application Scenarios

The presented technique for transforming Use Cases to EPC models can be applied in many contexts:

- The original motivation of the transformation was the **Derivation of Business Processes for SOA projects from Software Requirements:** Since SOA projects build upon defined business processes which are often not existent prior to the project's start, they need to be reconstructed from the requirements. If functional requirements have been documented as Use Cases the transformation saves times and effort.
- **Validation of Use Cases:** Generation of business processes can easily show non-matching pre- and post-conditions by visualising the control flow. Thereby naming inconsistencies, missing conditions and missing Use Cases providing needed post-conditions can be detected, which leads to improved Use Cases.
- **Interview Tool for Collecting Business Processes:** Use Cases can be used by business process engineers in order to discover "hidden", unknown and new business processes. For such Use Cases templates are available, e.g. [Coc05, p. 134]. Using Use Case templates in interviews with some stakeholders and merging them afterwards into business processes can be used as an efficient interview technique. Usage of the described transformation technique can accelerate the mining, examination and interpretation of the results. When using the generation capabilities within the interview itself, they can also be beneficial for visualising the process in order to improve and accelerate feedback as has been done with user interface sketches for software requirements [Vol06].

Possibly, there are other scenarios, but these scenarios alone demonstrate that the presented transformation technique can be handy in many situations.

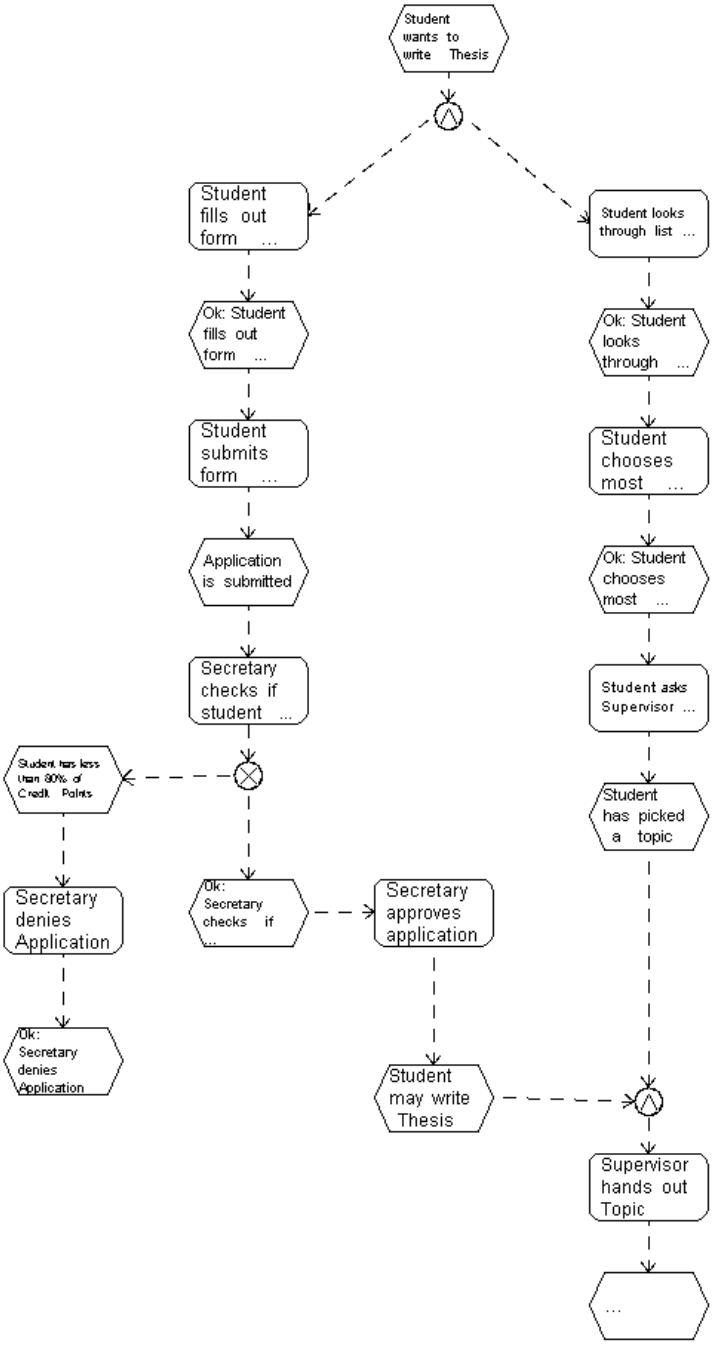


Figure 9: Resulting EPCs from the Use Cases

8 Related Work

Profiting from information contained in Use Cases for other models and development phases has always been a goal in software development projects, which often deal with business processes.

Cockburn himself only mentions the possibility of applying Use Cases for deriving business processes. He offers a template in [Coc05] but no rules or advise how to proceed from there.

The field of model-driven development has tried to combine the concept of Use Cases with its models. Instead of tabular and textual descriptions, UML sequence diagrams or similar models are used [Gro04]. A Use Case is consequently denoted in Use Case diagrams and refined in other models thereby eliminating the textual description. This can pose a problem when communicating with non-technical users. A process for UML-based development of business processes is given in [OWS+03]. Missing from such approaches are capabilities for expressing control flow between Use Cases and therefore the generation of business process models.

Generation of other models is often inspired by the model-driven community and is often based on UML models. The only way to achieve business process generation is to model the control flow between Use Cases in at least one additional model. With the introduction of Use Case Charts and their formalization [Whi06], it is possible to define control flow dependencies between Use Cases and refine them in UML. From such descriptions other models can be generated, e.g. Hierarchical State Machines [WJ06].

Our approach differs from the Use Case Charts by working with the textual description and impose some limitations on these. Instead of explicitly modelling the control flow in other graphical models, the control flow is expressed by the Use Cases' conditions and triggers. Therefore, we can keep the textual description in order to communicate with stakeholders while still being able to generate business processes from these descriptions.

9 Conclusions & Outlook

Within this paper the transformation of Use Cases into EPC models has been shown. This conversion can be automatic although manual editing for improving readability and improving event names may be necessary. This approach can be used to gather requirements for SOA projects with user interaction using well-known techniques which have proofed their usefulness in many projects.

The described transformation rules have been implemented in a prototype application allowing Use Cases to be transferred to EPC models using XSLT.

Besides the initial usage scenario in SOA-based development projects, the usage of EPCs in conjunction with Use Cases allows better overview in projects with a huge number of Use Cases, validation of pre- and post-conditions and interview support for business process modelling.

In contrast to personal business processes as described in [GRvdA05] Use Cases carry more details since they are written with exactly these details in mind and contain information about other stakeholders. Both techniques could be combined if Use Cases are connected to business functions during the generation of the business process. A user could then switch between his or her process and the Use Cases he or she is interested in.

For better integration into development projects it remains an open research question how to enable round-trip engineering. In order to achieve this goal, transformation rules from EPCs to Use Cases are necessary. However, since EPCs can have arbitrary connectors not matching the Use Case meta-model, this conversion is unlikely to be fully automatic. However, being able to do round-trips, it would be possible to map business process changes and related comments, like those gathered using Experience Forums [LS06], not only back to business processes but to Use Cases as well.

The given transformation of Use Cases to EPCs is one step for enabling the integration of proven requirements engineering techniques and their related processes with the field of business processes. Thereby, established requirements techniques can be used efficiently in SOA projects. The given transformation techniques and the prototype application will hopefully serve as a foundation for the open research questions.

References

[AB02] Steve Adolph and Paul Bramble. *Patterns for Effective Use Cases*. Addison-Wesley, 1st edition, August 2002.

[AC03] J. Araujo and P. Coutinho. Identifying Aspectual Use Cases using a Viewpoint-Oriented Requirements Method. In *Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architectural Design, in conjunction with AOSD Conference 2003*, 2003.

[ACD+03] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. *Business Process Execution Language for Web Services Version 1.1*, May 2003.

[Ale03] Ian Alexander. Misuse Cases help to elicit non-functional Requirements. *Computing & Control Engineering Journal*, 14(1):40–45, February 2003.

[Chr06] Christian Chrisp. Konzept und Werkzeug zur erfahrungsbasierten Erstellung von Use Cases. Master's thesis, Leibniz Universität Hannover, October 2006.

[Coc05] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 14th edition, August 2005.

[Die06] Dimitri Diegel. Konzept zur Verknüpfung von Use Cases mit ereignisgesteuerten Prozessketten. Master's thesis, Leibniz Universität Hannover, 2006.

[EMPR05] Bill Eidson, Jonathan Maron, Greg Pavlik, and Rajesh Raheja. SOA and the Future of Application Development. In Jen-Yao Chung, Goerge Feuerlich, and Jim Webber, editors, *Proceedings of the First International Workshop on Design of Service-Oriented Applications (WDSOA'05)*, pages 1–8, 2005.

[Gro04] Object Management Group. Unified Modeling Language: Superstructure. WWW: http://www.omg.org/cgi-bin/doc?formal/05-07-04, 2004.

[GRvdA05] Florian Gottschalk, Michael Rosemann, and Wil M.P. van der Aalst. My own process: Providing dedicated views on EPCs. In Markus Nüttgens and Frank J. Rump, editors, *EPK 2005 - Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, pages 156–175, 2005.

[LLSG06] Daniel Lübke, Tim Lüecke, Kurt Schneider, and Jorge Marx Gómez. Using EPCs for Model-Driven Development of Business Applications. In Franz Lehner, Holger Nösekabel, and Peter Kleinschmidt, editors, *Multikonferenz Wirtschaftsinformatik 2006*, volume 2, pages 265–280. GITO Verlag, 2006.

[LS06] Daniel Lübke and Kurt Schneider. Leveraging Feedback on Processes in SOA Projects. In *Proceedings of the EuroSPI 2006*, 2006.

[MN05] Jan Mendling and Markus Nüttgens. EPC Markup Language (EPML) - An XML-Based Interchange Format for Event-Driven Process Chains (EPC). *Information Systems and e-Business Management (ISeB)*, 4(3):245–263, July 2005.

[NR02] Markus Nüttgens and Frank J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In Jörg Desel and Mathias Weske, editors, *Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen - Promise 2002*, LNI, pages 64–77. Gesellschaft für Informatik, October 2002.

[OWS+03] Bernd Oestereich, Christian Weiss, Claudia Schröder, Tim Weilkiens, and Alexander Lenhard. *Objektorientierte Geschftsprozessmodellierung mit der UML*. d.punkt Verlag, 2003.

[Vol06] Carl Volhardt. Werkzeug zur Untersttzung von Interviews in der Prozessmodellierung. Bachelor's Thesis; WWW: http://www.se.uni-hannover.de/documents/studthesis/BSc/Carl_Volhard-Werkzeug_zur_Unterstuetzung_von_Interviews_in_der_Prozessmodellierung.pdf, 2006.

[Whi06] Jon Whittle. A Formal Semantics of Use Case Charts. Technical Report ISE-TR-06-02, George Mason University, http://www.ise.gmu.edu/techrep, 2006.

[WJ06] Jon Whittle and Praveen K. Jayaraman. Generating Hierarchical State Machines from Use Cases. In Martin Glinz and Robyn Lutz, editors, *Proceedings of the 14th IEEE International Requirements Engineering Conference*, pages 19–28. IEEE Computer Society, 2006.

[ZM05] Jörg Ziemann and Jan Mendling. EPC-based Modelling of BPEL Processes: A Pragmatic Transformation Approach. In *Proceedings of the 7th International Conference "Modern Information Technology in the Innovation Processes of the Industrial Enterprises" (MITIP 2005)*, 2005.