Transforming Process Models to Problem Frames

Stephan Faßbender[*] and Banu Aysolmaz[+]

[*] paluno - The Ruhr Institute for Software Technology, stephan.fassbender@paluno.uni-due.de
[+]Dept. of Computer Science, VU University, Amsterdam, The Netherlands, b.aysolmaz@vu.nl

Abstract. An increase of process awareness within organizations and advances in IT systems led to a development of process-aware information systems (PAIS) in many organizations. UPROM is developed as a unified BPM methodology to conduct business process and user requirements analysis for PAIS in an integrated way. However, due to the purpose, granularity and form of UPROM artifacts, one cannot analyze the software requirements in detail with (semi-)formal methods for properties such as completeness, compliance and quality. In contrast, Problem Frames modeled using the UML4PF tool can be used for such analysis. But using the Problem Frames notation and corresponding methods alone does not cover a direct support for building a PAIS. Hence, in this work we propose to integrate UPROM and UML4PF using model transformation. We use eCompany, a project which is part of an e-government program, as running example.

Keywords: Requirements engineering, Transformation, UPROM, Problem Frames

1 Introduction

With the rise of process awareness and IT systems, more and more organizations develop process-aware information systems (PAIS) to automate their processes [19]. Business process modeling (BPM) is the key instrument to analyze, design and identify the user requirements of a PAIS [16, 20]. However, process functions need to be further analyzed for their behavior, data usage and operations during PAIS execution to identify user requirements in the business domain [7, 16, 17, 20]. Moreover, those requirements need to be represented in a structured form so that they can be systematically exploited for detailed requirements analysis and software development.

UPROM is developed as a unified BPM methodology to conduct business process and user requirements analysis for PAIS in an integrated way [5]. Process models of UPROM are then used to automatically generate artifacts for PAIS development [7]. To provide a seamless link between business process analysis and software development, methods to systematically utilize business process knowledge for PAIS development are necessary. UPROM already provides structured representation of process and requirements knowledge in business domain in the form of models. However, due to their purpose, granularity and form, we cannot analyze them with formal methods for properties, such as completeness and compliance. Moreover, the treatment of software qualities in a structured way is neglected in UPROM right now as qualities are only added as textual notes or high level goal. Hence, UPROM focuses on the functionality right now. To improve this situation, we need to refine these models in such a form that they can be used in the subsequent phases, specifically detailed software requirements analysis, and can be (semi-)formally analyzed.

The Problem Frames method based on the Problem Frames notation, as introduced by Jackson [15], decomposes the overall problem of building the system-to-be into small sub-problems. Using the Problem Frames approach to define software requirements provides various benefits. First of all it allows a thorough analysis and rigid definition of requirements [14]. Moreover, Problem Frames models have a semi-formal structure, which allows an (semi - automated) analysis of different qualities, such as, for example, privacy [9], security [13], and compliance with laws and standards [11, 8], and an (semi - automated) interaction analysis for functional [3] as well as quality [1] requirements. Additionally, several topics such as aspect-orientation [12], variability [2], and requirements reconciliation and optimization [1] can be treated if necessary. Last but not least, Problem Frame models allow a seamless transition to the architecture and design phase [4]. Moreover, the overall system-to-be is decomposed using the requirements, which makes the usage of problem diagrams scalable even for large systems, because the complexity of single problem diagrams is independent of the system size.

The UML4PF tool which facilitates the modeling in the Problem Frames notation is used for all the aforementioned analysis. Typically, problem frame models are developed from scratch for each software project. This requires a considerable amount of effort and makes it difficult to establish the trace links between problem diagrams and business processes. Such a traceability is the key to ensure the completeness of software requirements from a business perspective, specifically when developing a PAIS.

In this paper, we propose a method to utilize business process and user requirements models of UPROM to create Problem Frame models. As a result, the Problem Frame models are directly based on process models, which ensures that the Problem Frame model covers the user requirements of the business domain. In this way, process knowledge elicited in the business domain is systematically transferred to subsequent phases of software development and the definition of process aware requirements is ensured. The integration of Problem Frames and UPROM also lowers the obstacles of using Problem Frames within a company. First of all, it relates a notation which is already well known with Problem Frames. Second, the transformation lowers the effort of using Problem Frames and assures that information which is already available in the form of process descriptions can be actually reused. Therefore, this integration enables a detailed requirements analysis using semi-formal Problem Frame models, but at the same time lowers the obstacles and effort to be taken for using the Problem Frame notation.

In the following section (Sec. 2) we introduce the background on UPROM and Problem Frames, which is necessary to comprehend the rest of the paper. Next, we introduce our case study in Sec. 4. Then, we describe our method (Sec. 3), which consists of two phases. In Sec. 4 we show the results of applying our method to our case study. In the last section (Sec. 6), we conclude our paper and present the future work.

2 Background

UPROM is a unified BPM methodology to conduct business process and user requirements analysis in an integrated way. Event-driven process chain diagram (EPC) is the core of UPROM notation for business process analysis. To conduct user requirement analysis, the functions in EPC diagrams to be automated by a PAIS are identified. If the function is to be (semi-) automated, a functional analysis diagram (FAD) is created as a sub-diagram. An FAD is used to analyze the requirements by identifying the responsibilities, related entities, operations on entities, and constraints applicable during the execution of the related function.

An FAD includes elements of the types function, entity, cluster, application, organizational elements (position, organizational unit, external person), and constraint. An example FAD is shown in Fig. 4. The function element, which is the same element as in the EPC diagram, is in the center of the FAD. Organizational elements are connected to the function named with the involvement types of “carries out”, “approves”, “supports”, “contributes to”, “must be informed on completion“. In this way, the responsibilities to conduct the function in PAIS are identified. Entities or clusters which are required for or during the execution of the function, are connected to the function. The operations executed on these entities are defined by the connection name. The possible operations are uses, views, creates, changes, reads, deletes, and lists. Each entity is also connected to the application on which it resides. If there are further constraints that restrict how the system operates, they are modeled as constraints and connected to the related application. During the development of FADs, conceptual level entity definitions are discovered. The general aggregation and generalization relations between those entities are modeled in Entity Relationship (ER) diagrams.

The UPROM tool is developed to support the method. By using the tool, one can automatically generate textual user requirements, functional size estimation and process documentation from the EPCs and FADs developed.

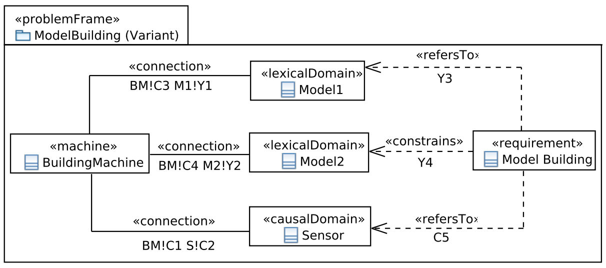


Fig. 1. Problem Frame Model Building

Problem Frames The objective of requirements engineering is to construct a *machine* (i.e., system-to-be) that controls the behavior of the environment (in which it is integrated) in accordance with the requirements. For this purpose, Jackson proposes to use the so called problem frame notation and approach [15]. The first step towards understanding and defining the problem to be solved by the machine is to understand the context of the machine. The context of the machine is given by the environment in which the problem to be solved is located, and in which the machine will be integrated to solve the problem. The environment is defined by means of domains and interfaces between these domains and the machine. Note that we use a UML-based adaption of problem frames, which is implemented as a specific UML profile for problem frames (UML4PF) as proposed by Hatebur and Heisel [14]. Hence, the graphical representation differs from the original representation proposed by Jackson [15], but the semantics remain the same. To be able to annotate problem diagrams with quality requirements, we extended the problem frames notation [4]. This enables us to complement functional requirements with quality requirements. Jackson distinguishes the domain types *machine* (Class with the stereotype ≪`machine`≫) which is the thing to be built, *biddable domains* (Classes with the stereotype ≪`biddableDomain`≫) that are usually people, *causal domains* (Classes with the stereotype ≪`causalDomain`≫) that comply with some physical laws or the specification is known, and *lexical domains* (Stereotype ≪`lexicalDomain`≫) which represent data. In the Problem Frames notation, *interfaces* connect domains and they contain *shared phenomena*. Shared phenomena may, for example, be events, operation calls or messages. They are observable by at least two domains, but controlled by only one domain, as indicated by "!". In Fig. 1 the notation $BM!C1$ (between the machine domain *BuldingMachine* and the causal domain *Sensor*) means that the phenomena *C1* are controlled by the machine *BuildingMachine*. All other domains which are connected by the interface the phenomenon is part of can observe it. In our case, $BM!C1$ is observed by the domain *Sensor*. The information about the machine and its environment is modeled in a so called *context diagram*. An example is given in Fig. 3.

Problem frames are a means to describe and classify software development problems. A problem frame is a kind of pattern representing a class of software problems. It is described by a frame diagram, which consists of *domains*, *interfaces* between them, and a *requirement*. Fig. 1 shows the problem frame for a model building variant. This variant contains the machine *BuildingMachine* which is the software or system which shall later fulfill the described requirement *Model Building*. The causal domain *Sensor* represents information about the real world from which the model is built. The lexical domain *Model1* provides the information necessary for the model building. The lexical domain *Model2* shall then reflect the result of the model building.

Requirements analysis with problem frames provides decomposition of the overall problem into sub-problems, which are represented by *problem diagrams*. A problem diagram is an instance of a *problem frame*. When we state a requirement we want to change something in the environment of the machine. Therefore, each requirement talks about and *constrains* at least one domain. Thus, these domains have to be influenced or changed to fulfill the requirement. For example, the domain *Model 2* (Class *Model 2* with stereotype ≪`lexicalDomain`≫) is constrained by the requirement *Model Building* (Class *Model Building* with stereotype ≪`requirement`≫) as shown in Fig. 1 (Dependency with stereotype ≪`constrains`≫ between class *Model Building* and class *Model2*). A requirement may also *refer to* several other domains in the environment of the machine which provide necessary information for fulfilling the requirement. The requirement *Model Building* refers to the domains *Sensor* (Class *Sensor* with stereotype ≪`causalDomain`≫) and *Model1* (Class *Model 1* with stereotype ≪`lexicalDomain`≫) (Dependency with stereotype ≪`refersTo`≫ between class *Model Building* and classes *Model1* and *Sensor*).

3 Method

The method to transform a UPROM model into a Problem Frames model consists of two phases. First, the EPC process model is turned into a context diagram. Then, each FAD bound to a function in the EPC is turned into one or more problem diagram(s).

Create a Context Diagram from an EPC An EPC, which describes the business process which shall be supported by the system-to-be, is transformed into one or more context diagrams through a series of steps which are described in the following. The expected output is an initial *Problem Frame Model* which includes *machines, causal domains, biddable domains, lexical domains, phenomena* and *context diagrams*.

Create Machines: In the first step we investigate the *applications* in the *EPC*. If this application is to be developed, we add a *machine* to our *problem frame model*. Decision rules for separating machines and external applications are described in [5].

Aggregate Machines: In many cases there are several *applications* to be developed, thus several machines, as part of an *EPC*. Now, we aggregate those to one *aggregated machine*. It might happen that there are several independent systems-to-be and therefore several *aggregated machines*. For each of them, we create a separate *context diagram*.

Create Causal Domains: For those *applications* which are not to be developed, we add *causal domains* to the *context diagram(s)*. We will refer to such applications as external or existing applications.

Create Biddable Domains: In this step, we consider elements which are of the types *position, group, organizational unit* or *external person*. Those elements have in common that their behavior is not predictable and they can be influenced only to some extent. Hence, we turn them into *biddable domains* and add them to the *diagram(s)*.

Create Lexical Domains: In this step, we transform the *EPC* elements that represent information (*document, list, log* or *files*) to *lexical domains*. We then check if ERDs contain any information to aggregate them. If so, we add this joined element as lexical domain and create a mapping diagram for relation between that and its parts. We add all lexical domains which are not part of another lexical domain to the *context diagrams*.

Create phenomena: Only those functions that are connected to an application to be developed are used to create *phenomena*. *Phenomena* do not exist on their own but in relation to domains which control and observe them. Hence, we also have to take into account the elements of the *EPC* connected to the functions. We distinguish four different cases for transformation depending on the type of elements connected to the functions. Table 1 shows a transformation table for one of the cases[1]. The first part of the table defines the input required for the transformation. The first mandatory input is *part of the EPC* which defines elements and relations that have to be present in the EPC at hand to enable the transformation. Note that some elements can be exchanged: e.g. the actor can be modeled in different ways. The *Questions* part defines the questions to determine the correct option for transformation and is optional. The second main part defines the output. The output can differ as there are different options for the transformation. An option is described by the answers given (☑ stands for the answer yes, ⚡ stands for the answer no, and ⍰ stands for no answer or an indifferent answer). Underneath, the resulting output is given in the UML4PF notation. In case of a fully automated transformation, the default option is used, rather than expecting answers to the questions. Note that the complete transformation covers a combination of the four cases and different options. Hence, some phenomena might be created several times. In case a phenomenon already exists, it is not created a second time.

Create Problem Diagrams The input for the process of *creating problem diagrams* are the *FADs* developed in the business analysis phase. For each FAD we create a problem diagram including the according *requirement*. For each *application* which does not already exist and is part of the corresponding FAD, we add a new *machine* to the problem diagram. Each *machine* created this way has to be *mapped* to the corresponding

[1] All transformation tables can be found in a technical report available under http://www.uml4pf.org/publications/TR20150306_UPROM2UML4PF.pdf

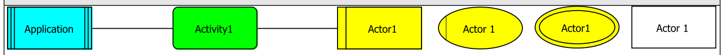
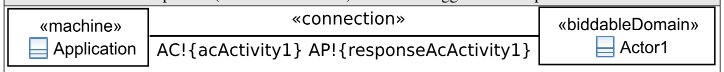
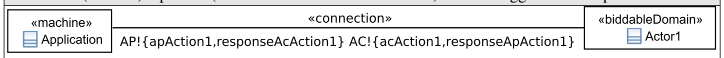
Case 1: Application, Function, Position | Group | Organizational Unit | External Person		
Input		
Part of the EPC		
		
Questions (for semi-automated transformation)		
(1)	Can the *Actor1* trigger the *Activity1*?	☐
(1.1)	Gets the *Actor1* a response by the *Application* when he/she triggers the *Activity1*?	☐
(2)	Can the *Application* trigger the *Activity1*?	☐
(2.1)	Gets the *Application* a response by the *Actor1* when it triggers the *Activity1*?	☐
Output		
Option 2 (**1**☑ **1.1**☑ **2**☒**2.1**☒): External Trigger with Response		
		
(Default) Option 5 (**1**☑/ ☐**1.1**☑/ ☐ **2**☑/ ☐**2.1**☑/ ☐): Multi-Trigger with Response		
 «machine» Application «connection» AP!{apAction1,responseAcAction1} AC!{acAction1,responseApAction1} «biddableDomain» Actor1		

Table 1. (**Context Diagram**)Create Phenomena: Case 1 (Option 2 and 5 out of 5 Options)

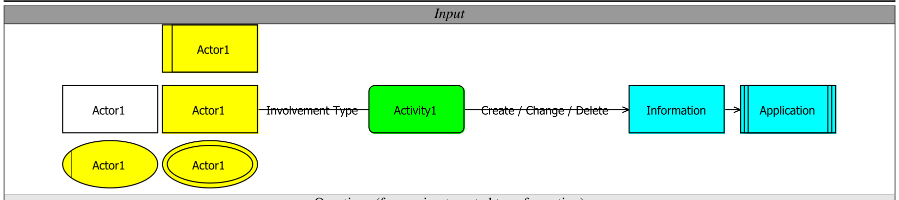
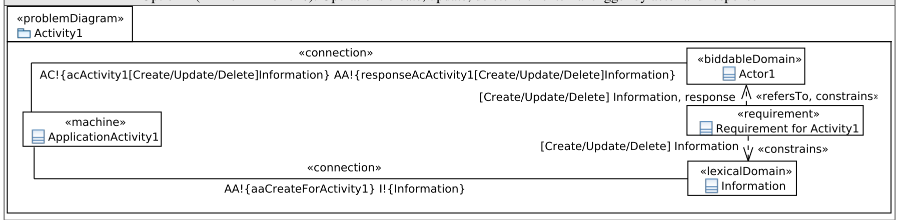
machine that is part of the context diagram or to one of the already known sub-machines which were aggregated to the machine in the context diagram.

For all *entities* in the FAD, we *add a lexical domain* to the problem diagram. If we add a not already known *lexical domain* this way, we search for a *mapping* of the corresponding entity to existing entities in the ERDs. If we find such a mapping, we also model the mapping in the problem frame model. In case we cannot find such a mapping, we have to create one or we have to add the new lexical domain to the related context diagram. In the same manner we *add a biddable domain* for each *position, group, organizational unit, internal person, or external person* and *map* these *biddable domains* whenever necessary. We also *add causal domains* for the *existing applications* and *map* them if necessary.

Up to this point we added machines and domains using the entities in FAD. Next, we need to *add the connections (associations), phenomena and dependencies* between them. We distinguish five cases with different options for transformation. An example transformation table for problem diagrams is given in (Table 2)[1]. In case of a fully automated transformation, always the default option is used, while for semi-automated transformation answers to the questions is used to identify the transformation option. Note that the involvement types of the actors, if present, might indicate certain answers to the questions. Another aspect is that the transformation might not generate valid or complete problem diagrams in the first place. This has to be corrected in a later step.

After adding the phenomena, we also *create a textual* description of the *requirement*. Frequently, textual requirements are needed in PAIS projects for domain expert reviews, contract preparation and project management purposes. In UPROM, textual requirement sentences are generated automatically using FADs [6]. These textual requirements might contain more information than represented by the transformed problem diagram. Hence, they are of use for further refinements of the problem diagrams.

After creating and adding the textual requirement to the problem diagram for the FAD at hand, the transformation itself is finished. Now the resulting problem diagram needs to be analyzed further. First, we have to *adjust the problem diagram to be valid*. As mentioned, the combination of different transformation options might not result in

Case 1: Application and operations *create, update, delete*		
Input		
		
Questions (for semi-automated transformation)		
(1)	Can the *Actor1* trigger the operation *create* | *update* | *delete* on *Information* for *Activity1*?	☐
(1.1)	Gets the *Actor1* a response by the *Application* when he/she triggers the operation *create* | *update* | *delete* on *Information* for the *Activity1*?	☐
(2)	Can the *Application* trigger the operation *create* | *update* | *delete* on *Information* for *Activity1*?	☐
(2.1)	Gets the *Actor1* a notification by the *Application* when it triggers the operation *create* | *update* | *delete* on *Information* for *Activity1*?	☐
Output		
Option 4 (**1**☑ **1.1**☑ **2**☒**2.1**☒): Operations create, update, delete with external trigger by actor and response		
		

«problemDiagram»
Activity1
«connection»
AC!{acActivity1[Create/Update/Delete]Information} AA!{responseAcActivity1[Create/Update/Delete]Information}
«biddableDomain»
Actor1
[Create/Update/Delete] Information, response
«refersTo, constrains»
«machine»
ApplicationActivity1
«requirement»
Requirement for Activity1
[Create/Update/Delete] Information
«constrains»
«connection»
AA!{aaCreateForActivity1} I!{Information}
«lexicalDomain»
Information

Table 2. (**Problem Diagram**) Create Phenomena: Case 1 (Option 4 out of 5 Options)

a valid problem diagram. For example, it can happen that a requirement only refers to domains but no domain is constrained. Such invalid problem diagrams point out missing information, which we now have to add. Even if valid, the problem diagrams which are created from FAD tend to be rather big. The reason is that a business activity might combine different system functions. Hence, it might be possible to *decompose a problem diagram* into smaller sub-diagrams. Note that the last two steps can only be applied if doing a semi-automated transformation.

As the information about desired quality requirements is only modeled in an unstructured and high level way in UPROM, the information about quality requirements is missing up to this point. But as the functionality as described by the UPROM models is now turned into a Problem Frames model, for example, the UML4PF extension PresSuRE can be used to turn a high level goal (such as "System shall be secure") into related security requirements. In the same manner all other UML4PF extensions can be used to add quality requirements, as the extensions only require a detailed model of the functionality and some high level quality goals such as performance, compliance, and so forth.

4 Application

In this section, we briefly introduce our case study. Afterward, the application of our method for creating a context diagram and problem diagrams is explained on our case.

Case Study The Company Central Registration project (eCompany) was initiated as part of an e-government program to develop an online workflow management system to automate life cycle processes of companies such as such as citizen application for company establishment, management of new establishments and updates by officers. The initial phase of the project included the analysis of business processes and user requirements and preparation of the technical contract. In this phase, three analysts from the integrator, three from the subcontractor and two domain experts cooperated. UPROM was used for process and requirements analysis. 15 EPCs and 82 FADs were created. 363 generated textual requirement statements were used in the technical contract.

A part of the process for establishing a company is shown in Fig. 2. The FAD for the last function *Define company name* of this EPC is shown in Fig. 4. The *Company Establishment Applicant carries out* this function. To accomplish the function, the *Company Establishment Application* has to *create* the *company name* and the *name alternatives* ensured not to match the *list* of existing *company names*, and the *name control fee*, which is calculated based on the number of *name alternatives* identified by the company establishment applicant. The company establishment application sends (*creates*) the name control fee to an external application called *finance office web service*. To identify the later payment it *uses* the *application number*. In the end, the company establishment application *changes* the *application status*. Restrictions on the function to be considered during its execution, which cannot be expressed directly in the FAD, are attached as constraint elements in dark green.

Create Context Diagram In the following, we will explain how to create the context diagram for our case as shown in Fig. 3. In the first step, we *create the machines*. The EPC contains two applications we need to consider: The *Company Establishment Application* and the *Company Establishment Approval* application. We *aggregate the machines* found to the *eCompanyApplication* machine. Next, we *create the causal domains*. In case of the EPC alone we do not have to model a causal domain as the external applications, such as *Finance Office Web Service*, only relate to external processes. For the causal domains we also have to consider the FAD related to the functions in the EPC at hand. The external application finance office web service is also part of the FADs for the functions “Define company name” and “Control original docs”. Hence, we add the causal domain *FinanceOfficeWebService* to our context diagram. Next, we *create the biddable domains*. According to the EPC, we create the biddable domains *CompanyRegistrar* and *CompanyEstablishmentApplicant*.

Afterward, we also create the lexical domains *LandRegister*, *StatementOfEstablishment*, *LetterOfWarranty*, *ForeignIdentityCard*, and *CompanyStatute*. As described in Sec. 3, we also examine the ERD and create according mapping diagrams.

Last, we *create the phenomena*. We exemplify the creation of phenomena using the function “Define company name”. We conduct a semi-automated transformation. Table 3 shows the transformation case for the function which matches the first case. The function is triggered by the company establishment applicant who gets a response, while the company establishment application remains passive. Note that in the resulting output the company establishment application is replaced by the eCompanyApplication, which aggregates the different applications which have to be developed to support the EPC at hand. The function “Define company name” matches also further cases which are not shown for sake of brevity but explained in the technical report[2]. The resulting complete context diagram is shown in Fig. 3.

Create Problem Diagrams The FAD which we use to exemplify the creation of a problem diagram is the one bound to the function “Define company name” as shown in Fig. 4. We create a problem diagram for the FAD which contains the *requirement*

[2] http://www.uml4pf.org/publications/TR20150306_UPROM2UML4PF.pdf

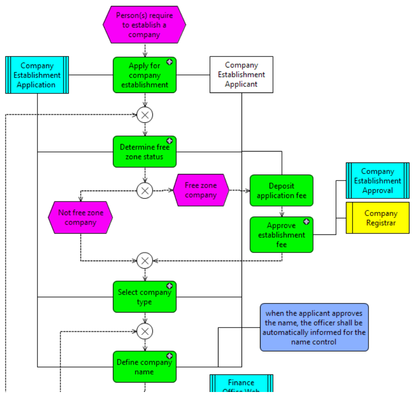
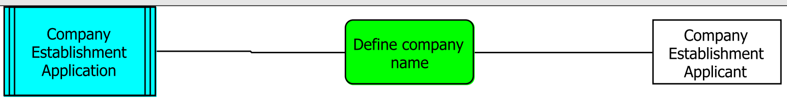
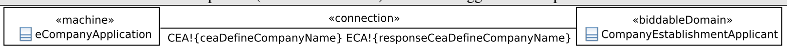

Person(s) require to establish a company
Company Establishment Application
Apply for company establishment
Company Establishment Applicant
Determine free zone status
Free zone company
Not free zone company
Deposit application fee
Company Establishment Approval
Company Registrar
Approve establishment fee
Select company type
when the applicant approves the name, the officer shall be automatically informed for the name control
Define company name
Finance Office Web

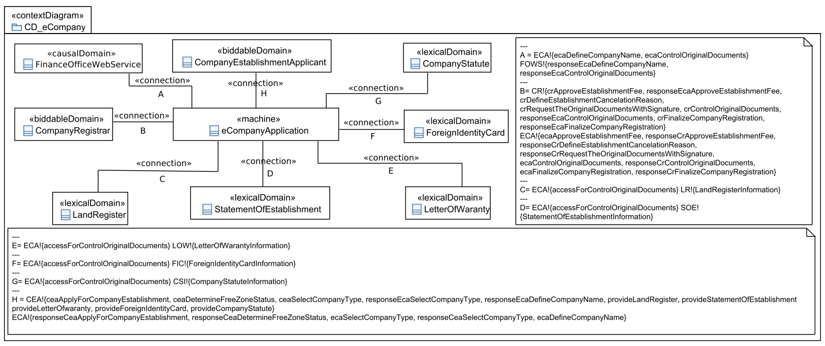
Fig. 2. Part of the EPC for the process "Establish Company"

Case 1: Application, Function, External Person		
Input		
Part of the EPC		
		
Questions (for semi-automated transformation)		
(1)	Can the *Company Establishment Applicant* trigger the *Define company name*?	☑
(1.1)	Gets the *Company Establishment Applicant* a response by the *Company Establishment Application* when he/she triggers the *Define company name*?	☑
(2)	Can the *Company Establishment Application* trigger the *Define company name*?	☒
(2.1)	Gets the *Company Establishment Application* a response by the *Company Establishment Applicant* when it triggers the *Define company name*?	☒
Output		
Option 2 (**1**☑ **1.1**☑ **2**☒**2.1**☒): External Trigger with Response		
		

Company Establishment Application
Define company name
Company Establishment Applicant

«machine»
eCompanyApplication
«connection»
CEA!{ceaDefineCompanyName} ECA!{responseCeaDefineCompanyName}
«biddableDomain»
CompanyEstablishmentApplicant

Table 3. (**Context Diagram**) Create Phenomena: Case 1 for "Define company name"



A = ECA!{ecaDefineCompanyName, ecaControlOriginalDocuments}
FOWS!{responseEcaDefineCompanyName,
responseEcaControlOriginalDocuments}

B= CR!{crApproveEstablishmentFee, responseEcaApproveEstablishmentFee,
crDefineEstablishmentCancelationReason,
crRequestTheOriginalDocumentsWithSignature, crControlOriginalDocuments,
responseEcaControlOriginalDocuments, crFinalizeCompanyRegistration,
responseEcaFinalizeCompanyRegistration}
ECA!{ecaApproveEstablishmentFee, responseCrApproveEstablishmentFee,
responseCrDefineEstablishmentCancelationReason,
responseCrRequestTheOriginalDocumentsWithSignature,
ecaControlOriginalDocuments, responseCrControlOriginalDocuments,
ecaFinalizeCompanyRegistration, responseCrFinalizeCompanyRegistration}

C= ECA!{accessForControlOriginalDocuments} LR!{LandRegisterInformation}

D= ECA!{accessForControlOriginalDocuments} SOE!
{StatementOfEstablishmentInformation}

E= ECA!{accessForControlOriginalDocuments} LOW!{LetterOfWarantyInformation}

F= ECA!{accessForControlOriginalDocuments} FIC!{ForeignIdentityCardInformation}

G= ECA!{accessForControlOriginalDocuments} CSI!{CompanyStatuteInformation}

H = CEA!{ceaApplyForCompanyEstablishment, ceaDetermineFreeZoneStatus, ceaSelectCompanyType, responseEcaSelectCompanyType, responseEcaDefineCompanyName, provideLandRegister, provideStatementOfEstablishment
provideLetterOfwaranty, provideForeignIdentityCard, provideCompanyStatute}
ECA!{responseCeaApplyForCompanyEstablishment, responseCeaDetermineFreeZoneStatus, ecaSelectCompanyType, responseCeaSelectCompanyType, ecaDefineCompanyName}

Fig. 3. Context Diagram for the eCompany System

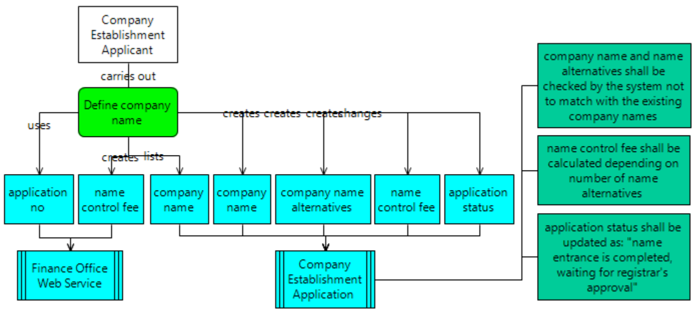
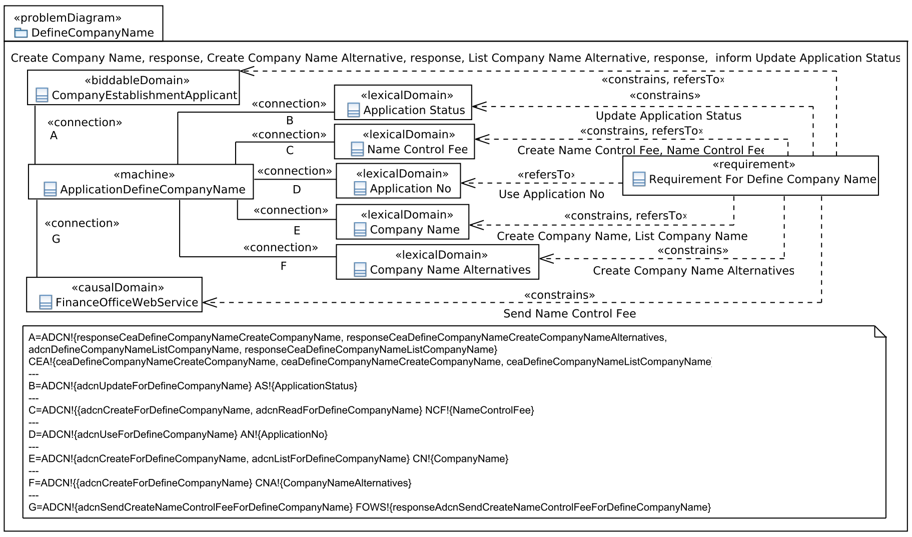

Company Establishment Applicant
carries out
Define company name
uses
creates
lists
creates creates createschanges
application no
name control fee
company name
company name
company name alternatives
name control fee
application status
Finance Office Web Service
Company Establishment Application
company name and name alternatives shall be checked by the system not to match with the existing company names
name control fee shall be calculated depending on number of name alternatives
application status shall be updated as: "name entrance is completed, waiting for registrar's approval"

Fig. 4. FAD for the activity "Define company name"



```
A=ADCN!{responseCeaDefineCompanyNameCreateCompanyName, responseCeaDefineCompanyNameCreateCompanyNameAlternatives,
adcnDefineCompanyNameListCompanyName, responseCeaDefineCompanyNameListCompanyName}
CEA!{ceaDefineCompanyNameCreateCompanyName, ceaDefineCompanyNameCreateCompanyName, ceaDefineCompanyNameListCompanyName
---
B=ADCN!{adcnUpdateForDefineCompanyName} AS!{ApplicationStatus}
---
C=ADCN!{{adcnCreateForDefineCompanyName, adcnReadForDefineCompanyName} NCF!{NameControlFee}
---
D=ADCN!{adcnUseForDefineCompanyName} AN!{ApplicationNo}
---
E=ADCN!{adcnCreateForDefineCompanyName, adcnListForDefineCompanyName} CN!{CompanyName}
---
F=ADCN!{{adcnCreateForDefineCompanyName} CNA!{CompanyNameAlternatives}
---
G=ADCN!{adcnSendCreateNameControlFeeForDefineCompanyName} FOWS!{responseAdcnSendCreateNameControlFeeForDefineCompanyName}
```

Fig. 5. Problem Diagram for "Define company name"

"Requirement For Define Company Name". We *add the application ApplicationDefineCompanyName* as machine domain to the problem diagram. This *machine* we created is *mapped* to the CompanyEstablishmentApplication machine which itself is mapped to the eCompanyAppSlication machine in the context diagram. We *add the lexical domains Application Status, Name Control Fee, Application No, Company Name and Company Name Alternatives* to the problem diagram as the corresponding entities are part of the FAD at hand. The *biddable domain CompanyEstablishmentApplicant* is also *added* to the problem diagram. Last, we *add a causal domain* for the external application *Finance Office Web Service*.

Up to this point, we prepared the elements of a problem diagram reflecting an FAD. Next, we need to add the *connections (associations), phenomena and dependencies* between them. Will will only discuss the transformation for the creation of the *company name* by the *Company Establishment Applicant* using the *Company Establishment Application* in detail. The creation is triggered by the company establishment applicant who gets a response by the company establishment application. Hence, we use *option 4* of *case 1* for the transformation (see Table 2). We get the phenomena for triggering the company name creation and the according response between the biddable domain *CompanyEstablishmentApplicant* and the machine *ApplicationDefineCompanyName*. Furthermore, we get the phenomena for the actual creation of the company name between the machine and the lexical domain *Company Name*. The *Requirement For Define Company Name* constrains the company name, and refers to and constrains the company establishment applicant. The FAD “Define company name” matches also six further cases and options which are not shown for the sake of brevity but are explained in the technical report[2]. After adding the phenomena, we also *create a textual* description of the *requirement*. The resulting problem diagram is shown in Fig. 5.

Next, we would have to *adjust the problem diagram to be valid*. Moreover, as the problem diagram is quite big, we might *decompose a problem diagram* into smaller sub-diagrams. Those two steps are currently under research.

5 Related Work

BPM is frequently used for requirements analysis of PAISs. However, process models are not expressive enough for requirements engineering [16]. Yet, there exist a few studies that integrate process and requirements modeling [6] or derive requirements from process models [17]. Examples are goal modeling approaches utilized in early phases [18] and data-centric approaches [20]. The only study integrating problem frames and process modeling is tne by Cox et al., who link process models to problem frames via Role-Activity Diagrams [10]. However, this study does not systematically handle derivation of requirements from process models.

6 Conclusion

In this paper we have presented a method to integrate the UPROM method with the Problem Frames method via a transformation approach. The motivation is the need to systematically transfer process knowledge to software requirements analysis and further phases. Currently, the business and requirements analysis phases are quite separated, which makes it hard to ensure that a system developed is really able to support the business processes. The integration of UPROM and UML4PF ensures conformance to business processes and enables the use of all the methods offered by UML4PF for various analysis and the seamless transition to following software engineering phases.

Our contributions in this paper are: **1)** A guided method for creating a context diagram and problem diagrams based on process models is presented. **2)** This method lays the foundation for tool support enabling the (semi-) automated transformation from process to Problem Frame models. **3)** In case of semi-automated transformation, additional information is elicited that enriches the Problem Frame diagrams in a lightweight way. **4)** The generated problem diagrams are suitable for further analysis, such as completeness and quality. **5)** Tracing from business processes to software development artifacts is enabled. **6)** The obstacles and effort for using Problem Frames may be lowered.

Currently, we are finishing the development of a tool which supports the (semi-) automated transformations[3] and we are preparing a validation within an ongoing project.

[3] The tool and used models are available under http://www.uml4pf.org/ext-uprom/index.html.

The aim of the validation will be to explore if the generated documentation is useful and usable in real-life settings and if the tool supports the application of the method.

References

1. A. Alebrahim, C. Choppy, S. Faßbender, and M. Heisel. Optimizing functional and quality requirements according to stakeholders' goals. In *System Quality and Software Architecture (SQSA)*, pages 75–120. Elsevier, 2014.
2. A. Alebrahim, S. Faßbender, M. Filipczyk, M. Goedicke, M. Heisel, and M. Konersmann. Towards a computer-aided problem-oriented variability requirements engineering method. In *ASDENCA workshop - CAiSE 2014, Proceedings*, pages 136–147. Springer, 2014.
3. A. Alebrahim, S. Faßbender, M. Heisel, and R. Meis. Problem-based requirements interaction analysis. In *REFSQ'2014 Proceedings*, pages 200–215. Springer, 2014.
4. A. Alebrahim, D. Hatebur, and M. Heisel. Towards systematic integration of quality requirements into software architecture. In *Proc. ECSA'11*, pages 17–25. Springer, 2011.
5. B. Aysolmaz. *UPROM: A Unified Business Process Modeling Methodology*. Phd, Middle East Technical University, 2014.
6. B. Aysolmaz and O. Demirörs. Deriving User Requirements from Business Process Models for Automation: A Case Study. In *REBPM Workshop, 2014*, pages 19–28. IEEE, 2014.
7. B. Aysolmaz and O. Demirörs. Modeling Business Processes to Generate Artifacts for Software Development : A Methodology. In *MISE workshop, 2014*, Hydarabad, India, 2014.
8. K. Beckers, I. Côté, S. Faßbender, M. Heisel, and S. Hofbauer. A pattern-based method for establishing a cloud-specific information security management system. *RE Journal*, 2013.
9. K. Beckers, S. Faßbender, M. Heisel, and R. Meis. A problem-based approach for computer aided privacy threat identification. In *APF '12,* pages 1–16. Springer, 2013.
10. K. Cox, K. T. Phalp, S. J. Bleistein, and J. M. Verner. Deriving requirements from process models via the problem frames approach. *Information and Software Technology*, 47(5):319–337, 2005.
11. S. Faßbender and M. Heisel. A computer aided process from problems to laws in requirements engineering. In *Software Technologies*, pages 215–234. Springer, 2014.
12. S. Faßbender, M. Heisel, and R. Meis. Aspect-oriented requirements engineering with problem frames. In *ICSOFT-PT 2014, Proceedings*, 2014.
13. S. Faßbender, M. Heisel, and R. Meis. Functional requirements under security pressure. In *ICSOFT-PT 2014, Proceedings*, 2014.
14. D. Hatebur and M. Heisel. Making pattern- and model-based software development more rigorous. In *Proceddings ICFEM'11*, pages 253–269. Springer, 2010.
15. M. Jackson. *Problem Frames. Analyzing and structuring software development problems*. Addison-Wesley, 2001.
16. C. Monsalve, A. April, and A. Abran. On the expressiveness of business process modeling notations for software requirements elicitation. In *IECON 2012*, pages 3132–3137, 2012.
17. J. Nicolás and A. Toval. On the generation of requirements specifications from software engineering models: A systematic literature review. *Information and Software Technology*, 51(9):1291–1307, 2009.
18. A. Pourshahid, D. Amyot, L. Peyton, S. Ghanavati, P. Chen, M. Weiss, and A. J. Forster. Toward an Integrated User Requirements Notation Framework and Tool for Business Process Management. In *e-Technologies, 2008 International MCETECH Conference on*, 2008.
19. J. Sinur and J. B. Hill. Magic Quadrant for Business Process Management Suites. 2010.
20. J. Vara, M. Fortuna, J. Sánchez, C. Werner, and M. Borges. A Requirements Engineering Approach for Data Modelling of Process-Aware Information Systems. In *Business Information Systems SE - 12*, pages 133–144. Springer, 2009.