A TRANSFORMATION APPROACH FROM eEPC TO S-BPM MODELS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

BAŞAK ÇAKAR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

JANUARY 2014

A TRANSFORMATION APPROACH FROM eEPC TO S-BPM MODELS

Submitted by **Başak Çakar** in partial fulfillment of the requirements for the degree of **Master of Science in Information Systems, Middle East Technical University** by,

Prof. Dr. Nazife Baykal
Director, **Informatics Institute** _____

Prof. Dr. Yasemin Yardımcı Çetin
Head of Department, **Information Systems** _____

Prof. Dr. Onur Demirörs
Supervisor, **Information Systems, METU** _____

Examining Committee Members:

Prof. Dr. Semih Bilgen
Electrical & Electronics Engineering, METU _____

Prof. Dr. Onur Demirörs
Information Systems, METU _____

Dr. Ali Arifoğlu
Information Systems, METU _____

Assoc. Prof. Dr. Altan Koçyiğit
Information Systems, METU _____

Dr. Barış Özkan
Information Systems Engineering, Atılım University _____

Date: 21.01.2014

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name and Surname : Başak ÇAKAR
Signature :

ABSTRACT

A TRANSFORMATION APPROACH FROM eEPC TO S-BPM MODELS

ÇAKAR, Başak
M.S., Department of Information Systems
Supervisor: Prof. Dr. Onur DEMİRÖRS

January 2014, 87 pages

Business process models are vital assets of organizations. The organizations prefer to use one of the many modeling methods and notations according to its features like tool support, size of user base, ease of use. During the last decade bottom up process modeling approaches such as S-BPM started to become popular among organizations. Many organizations have large process model assets modeled in a top down fashion. As a result, for most organizations to adopt bottom up process modeling approaches the existence of transformation algorithms is critical. In this work, model transformation is proposed as a method to migrate from eEPC to S-BPM. Direct mapping rules are defined to transform models and the application of these rules is demonstrated by on a real world case studies.

Keywords: Process Modeling, eEPC, S-BPM, Model Transformation

ÖZ

eEPC MODELLERİNDEN S-BPM MODELLERİNE BİR DÖNÜŞÜM YAKLAŞIMI

ÇAKAR, Başak
Yüksek Lisans, Bilişim Sistemleri Bölümü
Tez Yöneticisi: Prof. Dr. Onur DEMİRÖRS

Ocak 2014, 87 sayfa

İş süreci modelleri kuruluşların hayati varlıklarıdır. Kuruluşlar, araç desteği, kullanıcı tabanın büyüklüğü, kullanım kolaylığı gibi özelliklerine bakarak birçok modelleme yönteminden birini kullanmayı tercih ederler. Son yıllarda kuruluşlar arasında S-BPM gibi bu süreçleri tabandan yukarı doğru modelleyen yaklaşımlar popüler olmaya başlamıştır. Fakat kuruluşların elinde yukarıdan tabana doğru modellenmiş birçok iş süreci bulunmaktadır. Bu nedenle, birçok kuruluş için önceden modellenmiş süreçleri tabandan yukarı süreç modelleme yaklaşımlarına dönüştüren algoritmalar kritik bir öneme sahiptir. Bu çalışmada, model dönüşümü eEPC`den S-BPM`e göç için bir yöntem olarak önerilmiştir. Modellerin dönüştürülmesi için doğrudan eşleştirme kuralları tanımlanmış ve bu kuralların uygulamaları durum çalışmaları ile verilmiştir.

Anahtar Kelimeler: Süreç Modellemesi, eEPC, S-BPM, Model Dönüşümü

To My Family

ACKNOWLEDGEMENTS

I am deeply grateful to my supervisor Prof. Dr. Onur DEMİRÖRS who has guided me throughout this research with their invaluable suggestions, criticisms and encouragement.

My most sincere appreciation goes to Banu Aysolmaz. She never hesitated to provide support whenever I needed it. Our extensive discussions and their comments were invaluable.

Many thanks go to Murat Salmanoğlu for participating in case study and providing valuable comments.

I am also thankful to my family for their patience and support during this process.

Last but not least, I would like to thank my love for his love, trust, understanding and every kind of support throughout this study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ARIS	:	Architecture of Integrated Information Systems
BDD	:	Business Driven Development
BPEL	:	Business Process Execution Language
BPM	:	Business Process Management
BPMN	:	Business Process Modeling Notation
CPN	:	Colored Petri Nets
CSD	:	Communication Structure Diagram
EMF	:	Eclipse Modeling Framework
eEPC	:	Extended Event-driven Process Chain
EPC	:	Event-driven Process Chain
ERD	:	Entity Relationship Diagram
FAD	:	Functional Analysis Diagram
FTD	:	Function Tree Diagram
GMF	:	Graphical Modeling Framework
IT	:	Information Technology
oAW	:	OpenArchitectureWare
OC	:	Organizational Chart
oEPC	:	Object-oriented Event-driven Process Chain
SBD	:	Subject Behavior Diagram
S-BPM	:	Subject-oriented Business Process Management
SID	:	Subject Interaction Diagram
SMRG	:	Software Management Resource Group
TCPN	:	Timed Colored Petri Nets
UPROM	:	Unified Process Modeling Tool
YAWL	:	Yet Another Workflow Language

CHAPTER 1

INTRODUCTION

Business process management (BPM) becomes more crucial for organizations to maintain competitive advantage recently. BPM is the discipline of defining and outlining business practices, processes, information flows, data stores and systems [**1**]. It supports design, administration, configuration, enactment, and analysis of business processes. It also provides organizations to improve the performance of business processes in a short time and to respond changes in the market rapidly. Therefore; BPM increases customer satisfaction with quick responses and reduces business and service cost. Business process models are the main artefacts of BPM. Business process models describe logical order of activities and dependencies in organization [**2**]. Business process modeling is an important part of understanding and restructuring the activities and information of enterprise systems to achieve organization's business goals. Business process models are used to analyze process efficiency and quality by business analysts and managers. Additionally they are used to analyze system requirements and to design system architecture. Thereby models helps to narrow gaps between business processes (organization) and IT systems (technology). There are many modeling languages to visualize system specifications and process execution. In the frame of this study, we particularly focus on eEPC and S-BPM languages.

EPC is a business process modeling technique developed by Scheer et al. at the Institute for Information Systems in Germany, in 1990s [**3**] [**4**]. EPC represents business process as an ordered graph which shows chronological sequence and logical interdependencies between elements. In order to model more complex business processes, EPC notation is extended with additional elements from the organization and data view, which is called eEPC (extended Event-driven Process Chain). It is relatively simple notation to model business processes and highly accepted by the practitioners from diverse areas for business process re-engineering, management and documentation. eEPC is one of the most frequently used modeling notations for top down process modeling approaches. Top down approach focuses on overall business process and business strategy of organizations supported by that process. In eEPC sequence of activities, relationships between actors and data flow of process are modeled in the first step. In other words, the big picture up-front is given in eEPC models.

S-BPM (Subject-oriented Business Process Management) is a paradigm that is developed by Albert Fleischmann [**5**] to describe and execute business processes from the perspective of subjects. S-BPM gets inspired from natural languages and the structure of S-BPM is similar to sentence structure of natural languages. According to S-BPM, subjects are active elements in a business process. Therefore, they should be the starting point of the activities (like natural language sentences) [**6**]. S-BPM diagrams can be directly derived from process descriptions in natural language representation. Subjects execute business processes by exchanging messages with each others. Interactions between subjects are shown in the Subject Interaction Diagram (SID). SIDs visualize subjects and data flow (exchange messages) among them. Internal activities of subjects are shown in Subject Behavior Diagram (SBD). S-BPM uses top down approach in determining communication between subjects and uses bottom-up approach in determining internal behavior of subjects.

In the industry EPC is widely accepted during the last decade by means of ARIS toolset. A number of organizations represented their processes using eEPC. However; there is a gap between business and information technology systems in the eEPC [**7**]. S-BPM helps organization to close that gap. S-BPM enables to create dynamic business applications and to integrate them into the existing systems seamlessly. This also provides organizations, which use S-BPM as modeling language, competitive advantage. In addition to this, S-BPM provides a better representation for human interaction patterns and its notation is simple and easy to understand (only a few symbols). As an alternative to EPC, S-BPM gaining ground with IT support. S-BPM modeling language is based on process algebra with a clear formal semantics and this allows automatic code generation. Extensive usage of EPCs forces other modeling methods to accept EPCs as an input and transfer EPC information into their own needs [**8**]. Migration of legacy eEPC models requires considerable effort and substantial costs. Furthermore, it's a labor intensive work which increases the usage of personal resources and costs dramatically.

In this work mapping rules are defined to transform eEPC models to S-BPM models and automatic transformation is realized. Model transformation is adopted as a main method to provide automation. A model transformation takes a source model and transforms it into a target model by using predefined transformation definition (rules) [**9**]. The transformation definition is executed on concrete models by a transformation engine. In order to automate eEPC to S-BPM transformation a plug-in in UPROM is developed. UPROM stands for Unified Process Modeling Tool which is developed by Bilgi Grubu and SMRG Research Group.

1.1 Motivation

In the literature there are numerous studies on transformations between modeling notations. Since S-BPM is a new modeling paradigm, a very small part of them studied on S-BPM. Studies which compares S-BPM notation to others states that S-BPM usage increases inevitably because of its advantages. Proposed solutions related to S-BPM transformation are generally focuses on the generation of SIDs instead of SBDs. None of these works provides a concrete and explicit method to transform

eEPC models to S-BPM models. Therefore; the main motivation of this thesis is provide a guideline to generate SBDs from eEPC models.

In contrast to eEPC, S-BPM is a bottom up business process modeling paradigm. Business processes are constructed from the base upwards. Actions of subjects are defined firstly and they are linked together to form processes and procedures. Changes in business strategy of organizations lead frequent changes in core business processes of organizations. Bottom up approaches provide to handle these changes more rapidly and smoothly. In addition to this, S-BPM helps organizations to distribute responsibilities to accomplish a process to different units, groups or positions in the organization. Besides that, S-BPM is simple and understandable for software developers and stakeholders thus they can easily involve in modeling process and give feedbacks. All of them make S-BPM more preferable for organizations. However; many organizations have large process model assets modeled in a top down fashion. Therefore, a transformation tool is necessary for most organizations to adopt bottom up process modeling approaches.

There is only one tool for S-BPM language called Metasonic Suite. It is a commercial tool and does not comprise all SBD notations such as macro class and choice operator. This obligates modelers to model business processes by using limited number of S-BPM elements. Thus providing a more comprehensive modeling tool for S-BPM language is another motivation of this study. Since our tool is based on open source bflow* toolbox, it also enables organizations to minimize tool costs. Modelers can easily model business processes by using graphical user interface of S-BPM editor.

By defining mapping rules, manual transformation process is simplified for modelers. In addition an automatic transformation is also supported. Currently there are no tools to transform eEPC models to SBDs. It is also implemented in UPROM as a new feature. Automatic transformation provides modelers to adapt previously modeled business processes to S-BPM paradigm confidingly in a short time with minimum effort.

1.2 Proposed Solution

In order to solve problems in S-BPM modeling and eEPC to S-BPM transformation, in the following main outcomes of the proposed solution is given.

- *S-BPM Editor:* It is added to UPROM as a plug-in. It provides process modeling ability in S-BPM to UPROM. S-BPM editor provides modelers to construct process models graphically. Visual markers for core elements of SBD are satisfied by the editor. It also provides continues verification during modeling. Continuous verification feature provides modelers to recognize problems in the model during modeling time and prevents to develop IT systems wrongly.

- *Validation rules for SBDs:* Syntax and semantic rules are defined for SBDs. They include constraints on inter-elements relations and element sequence. Besides that, generated models should be verified to ensure that the model

does not contain errors, the validity of those models are automatically checked by the developed plug-in with those rules.

- *Guideline for eEPC to S-BPM transformation:* This works provides a guideline for manual transformation. Mapping rules are described for most commonly used eEPC elements. While defining mapping rules, different patterns which are the different combinations of elements are also taken into consideration to maintain semantic meaning.

- *Transformation Algorithm:* In the scope of this study, a transformation approach is also provided and the algorithm of transformation is given in detail. This provides an opportunity to other researchers to improve the algorithm for future studies. Transformation algorithm mainly focuses on the separation of eEPC model into SBDs according to subjects who accomplish the process and shows how to transform eEPC models in SBDs.

- *Transformation Engine:* As a proof of concept, transformation approach and mapping rules are implemented as a plug-in in UPROM. This implementation gives an idea, how defined transformation can be implemented and realized.

1.3 Organization of the Thesis

The remainder of the thesis is structured into seven chapters.

Chapter 2 summarizes the literature related to advantages of S-BPM and transformations between different business modeling languages.

Chapter 3 explains Business Process Management, Business Process Modeling, eEPC and S-BPM concepts.

Chapter 4 UPROM and S-BPM plug-in are explained. Metamodel of SBD, graphical user interface of S-BPM Editor and validation rules implemented in the editor is given.

Chapter 5 describes the proposed method in detail. The model transformation approach and mapping rules defined for transformation are described. Applied transformation algorithm is also explained in this chapter by the help of flow charts and description of them.

Chapter 6 presents the application of the eEPC to S-BPM transformation on a case involving multiple business processes in a public institute. Questions of the study, data collection and analysis strategies are explained. The conduct of the case is briefly described, automatic and manual transformations of selected processes and comparisons of transformations are given. Strengths and weaknesses of proposed solution are discussed and the outcomes of the case study are analyzed also in this chapter.

Chapter 7 presents the conclusions reached and summarizes the contribution and significance of this research.

CHAPTER 2

RELATED WORK

Business Process management, business process modeling and modeling notations are studied by several researchers in the literature. There are numerous studies on transformations between modeling notations. Since S-BPM is a new modeling paradigm, a very small part of them studied on S-BPM. However; there are also studies which compares S-BPM notation to others and conclude that S-BPM usage increases inevitably because of its advantages. Therefore, automatic transformation to S-BPM becomes critical. This chapter summarizes the literature related to advantages of S-BPM and transformations between different business modeling languages. In section 2.1, contributions of S-BPM approach to business process management are given. In second section 2.2, various transformation studies between business process modeling languages in the literature are explained.

2.1 Advantages of S-BPM

S-BPM is a bottom up business process modeling approach which is used to describe and execute business processes from the perspective of subjects. The structure of S-BPM is similar to sentence structure of natural languages. Thereby, S-BPM models are simple and understandable for software developers and stakeholders and they can easily involve in modeling process. The details of S-BPM approach is given in Chapter 3.

There are numerous notations for business process modeling such as UML Activity Diagrams, Business Process Modeling Notation (BPMN), Event-driven Process Chains and Petri nets widely used in the industry. A new paradigm, S-BPM introduced and it has contributed a lot in business process modeling. Those contributions make S-BPM more preferable by modelers.

In [**10**], Aguilar-Savén compares different modeling languages in terms of message exchange, communication partner's role, process flow and timing, visualization of none sequential process steps, understandability and clear structure of models in order to find the most suitable language for a specific project. According to this study S-BPM is very successful in visualizing message exchange between subjects.

Behavior of the communication partners is also well defined in S-BPM and it has a comprehensive notation.

In [**11**], Fleischmann et al. state that modeling business processes with respect to subjects has many advantages. Firstly S-BPM notation has few basic elements for modeling. Thus, learning and applying this approach is easier and quicker than other languages. Secondly, in S-BPM models are constituted from subjects, predicates and objects like natural languages. This makes models more simple and understandable for software developers and stakeholders can easily involve in modeling process. S-BPM provides executable models and in this way it bridges the gap between business models and IT systems. Finally, S-BPM models integrate functional and data-driven processes technologies. In addition to advantages mentioned in [**11**], Rodenhagen et al. [**12**] compare different modeling languages with regard to the usage of multiple instances. Multiple instances are not supported by EPC; on the other hand S-BPM provides simple notations to visualize multi subjects and repetitive subject behavior.

In [**7**], Singer et al. explain which features of S-BPM make it a valuable alternative for competitive advantage. According to Singer et al. S-BPM is valuable because it provides 'IT support', 'an integrated message orientation', 'a behavior oriented modeling approach', 'a puristic set of graphical symbols', 'natural language based process modeling' and 'process models with strictly formal definition'.

2.2 Business Model Transformations

In literature, there are most of studies on business model transformations that support by different motivations. Those are verification, bridging the gap between business models and IT systems, increasing understandability and necessity of following new modeling techniques. Generally unidirectional transformations are defined and a subset of source models elements is used.

2.2.1 Transformations from BPMN

In [**13**], Dijkman et al. check the semantic correctness of BPMN models by transforming them into Petri nets. Since Petri nets have more efficient analysis techniques, defining semantics of BPMN as mapping is preferred. Mapping rules from BPMN to Petri nets are described in detail for large subset of BPMN element. Rules are mainly focus on functional features and control flows (the order of activities and events) and message flows. This study omits the non-functional features such as groups and associations and organizational features such as lanes and pools. Rules for well-formed BPMN process are also defined, which are restrictions for control flows, start events and end events. They guarantee that all nodes are connected. van der Werf et al. [**14**] also use BPMN to Petri net transformation for verification and validation of BPMN models and define their own mapping rules. Mapping rules defined in those studies are completely different. Since there are not direct mapping between BPMN elements and Petri net elements.

In [**15**] and [**16**], transformation rules from BPMN to UML Activity diagram are defined without losing semantic meaning. The motivations are supporting Business-

Driven Development (BDD) and closing the gap between business process modeling and its realization. According to BDD, IT solutions should satisfy business requirements. UML is chosen because it is well accepted implementation standard in the industry and there are tools which generate the source code of UML models automatically. Automatic transformation from BPMN to UML Activity diagrams reduces time and resource usage for implementation. In [**15**], transformation rules are explained under six groups. They are direct transformation rules (one-to-one mapping), complex transformation rules, data transformation, transformation rules for loops, gateways and stereotypes. Bảo [**15**]combines the presentation power of BPMN with the implementation power of UML and gives a proposal for transformation. In [**16**], defined transformation rules are realized by ATL transformation language. The study focuses on explaining the implementation details in contrast to [**15**]. As a result of those studies, generated UML models should be checked by business modelers because it is not guaranteed that all elements in BPMN are transformed into UML Activity diagram without losing any information. However, in any case automatic transformation reduces the need for manual work.

2.2.2 Transformations from EPC/eEPC

eEPC is an top down business process modeling approach and it is widely used for modeling, analyzing and redesigning business processes by organizations. Business processes are visualized as an ordered graph in eEPC. eEPC diagrams show chronological sequence and logical interdependencies between functions and events. eEPC notation is explained in Chapter 3 in detail.

In [**17**] Hoyer et al. transform organizations' internal private processes to public processes. Internal private processes are modeled by eEPC modeling language and they include technique details about organization's internal business processes. However; public process view also required for external business partners. BPMN modeling language is selected for public process models. Since BPMN models are more easily understood by non-technical people. Transformation is performed semi-automatically. In the first part of the transformation original eEPC model is simplified and information hiding is applied. In this stage, trivial events are eliminated and events which initiate connectors are deleted. Organization units, positions and groups are removed. Only functions which send or receive message take into consideration, others are dropped and process interfaces are also dropped to get rid of hierarchical structure. In the second part mapping rules for eEPC to BPMN transformation are defined. As a result of this work one directional mapping is given. It is inferred that event mapping requires user interaction in order to save semantic meaning.

In [**18**], Tscheschner describes a direct mapping technique to convert eEPC to BPMN and defines transformation rules to map eEPC elements to BPMN elements. The main motivation of this work is that BPMN becomes more popular in the industry and automatic transformation is required to transform tons of business processes modeled by eEPC for rapid adaptation. Rules are defined for core EPC elements and extended EPC elements. Additional semantic rules for sending and receiving events are also defined. This work is realized as a plug-in in the Oryx-Editor. However;

eEPC and BPMN differ in their semantics and formalization. Therefore, a complete mapping (structural and semantic) is almost not achievable by solely using direct mapping for each and every component. In order to get complete one, elements of core EPC definition and a subset of eEPC elements are used for mapping. In [**19**] Levina investigates that whether the significant change in information content or not in eEPC to BPMN transformation. It gives a generic mathematical strategy for information context measurement. The study in [**18**] is used as an example. Information loss occurs during transformation however measurement results show that information content of the model is not change significantly. Additionally, Levina concludes that the size of the eEPC model do not increase the information loss.

In [**20**] Korherr et al. states that eEPC to UML transformation is critical to bridge the gap between business process engineering and software engineering. EPC diagrams are the starting point of software development since they are used to elicit requirements by software developers. They are also used to check the compatibility between the functions of an existing system and requirements of new business processes. In order to provide models to software developers in a well-known notation, EPC models should be converted to UML. This work gives a guideline to transform eEPC models to UML Activity Diagrams. New stereotypes, constraints and tag values are used to extend UML notation and cover more EPC elements. Mapping rules are defined in four categories; Functions and events, Additional process objects (data objects and actors), Flows (control, data and organization) and logical operators. Constraints which an EPC should satisfy are also defined as transformation prerequisites. Consequently; this work supports the business-goal oriented software development and provides software engineers to improve the quality of software system's requirements and design with minimum modeling effort.

In [**21**], Nüttgens et al. introduce an integration approach called by "The Object-oriented Event-driven Process Chain (oEPC)". Relations between EPC and UML diagrams (use case, activity diagram, class diagram and application architecture diagram) and transformation approach for each UML diagram are defined. For each diagram, different subsets of eEPC elements are used for transformation. For activity diagrams; functions and data object are transformed and others are omitted. Use case diagrams are constructed from functions and organizational units. For class diagrams functions and information objects are used. Each application (IT System) is transformed into an application architecture diagram and inner details of components are design from scratch. In this study, mapping between EPC and UML elements is not stated; only structural transformation approach is explained.

In [**22**], Loos et al. gives a different integration approach. Instead of translating EPC models into UML models, new object oriented extensions are defined for eEPC. Motivation of the study is the same as [**20**]. UML diagrams are used for system design and cover all system requirements. However; it is not sufficient to design and model business processes. Therefore integration of UML and business process modeling languages is critical. EPC notation is extended to cover class concept, data encapsulation, message concept, object hierarchies and inheritance. In this work, they define transitions between UML elements and EPC elements for class diagrams,

use case diagrams, statechart diagrams, sequence/collaboration diagrams and activity diagrams separately. This approach is implemented in ARIS Toolset. This study provides to combine process analysis phase and object oriented design and implementation phase.

In [**23**], eEPC models are transformed to Timed Colored Petri nets (TCPN) in order to check correctness of eEPC models. Errors in process design cause errors in developed system as well and error correction process becomes costly. Therefore, identifying and fixing errors in business process modeling phase is vital. All EPC elements can transform into places and transitions of Petri nets. Besides that in timed colored Petri nets additional features (data, time and probabilities) are available to map extended EPC elements. Additionally available CPN Tools support model checking for TCPNs. Therefore, TCPN is the best choice for verification. This study provides eEPC patterns and their corresponding TCPN patterns and describes how to verify the correctness of eEPC with the CPN Tools.

In [**24**] Lohmann et al. provides a survey about transformations of business process models into Petri nets. Since Petri net has formal semantics and it can be verified in a formal way, it is more preferable by academic people. However in the industry, business people prefer to use business languages like BPEL, BPMN, and EPCs. In contrast to academic languages, business languages do not have a proper semantics. Thus the interpretation of models changes according to modeler. In order to verify business process models by using Petri net verification techniques, transformation is required. This work investigates transformation studies from business models (BPEL, EPCs, YAWL, and BPMN) onto Petri nets. Challenges which are encountered in transformation are explained. Those are related to mapping difficulties and semantic problems. As a conclusion Lohmann et al. states that there is no chance to transform all element combinations with saving its semantics. Thus restrictions for source business models should be determined.

2.2.3 Transformations from S-BPM

In [**25**] Sneed states that BPMN is a worldwide standard. On the other hand, S-BPM provides modelers to model distributed system and S-BPM process models can be easily converted into abstract state machines and hence executable code. Thus, the usage of S-BPM increases inevitably in the industry. The goal of this study is providing a mapping method to modelers so as to transfer S-BPM models into BPMN models readily without losing information as much as possible. The method consists of a set of bidirectional mapping rules between subsets of both modeling languages. Since BPMN specification is inadequate for execution in terms of semantics, the method does not support the transformation of executable models. There is remarkable difference between business models and their executions in BPMN.

Transformation consists of two main parts. In the first part rules for atomic structures are defined which are basic modeling constructs. One-to-one mapping for them is not possible since some of the elements in BPMN can only be expressed in S-BPM by multiple elements. Sneed transforms a subset of BPMN elements which includes

manual tasks, service tasks, receive tasks, send tasks, sequence flows, conditional forks and process participants. Main challenges of the first part are transformation of multiple receive tasks and events. The first part of the transformation provides mapping for Subject Behavior Diagrams. In the second part, mapping rules for complex structures are defined. Complex structures are used to visualize the communication view between subjects. In this part pool and participants in BPMN are mapped to subjects and collaboration entities are mapped to S-BPM process entities.

In the conclusion of the study, losses of transformation between both modeling languages are analyzed. The most problematic parts of the BPMN to S-BPM transformation are lane sets, parallel gateways, user tasks, activity callers, events and annotations (group, documentation and text). In S-BPM to BPMN transformation, usage of business objects, multi subjects, roles are difficult to map without losing information. Defined transformation method is implemented as an eclipse plug-in in the Metasonic suite.

CHAPTER 3

BACKGROUND

In this chapter information about related concepts are given in order to increase understandability of the study. In Section 3.1 objectives, phases and key points of Business Process Management are described. Section 3.2 explains Business Process Modeling concepts and modeling notations. Only eEPC and S-BPM notations are in the scope of this study. Therefore, details about eEPC and S-BPM are given in sub-sections 3.2.1 and 3.2.2 respectively.

3.1 Business Process Management

Weske [**1**] defines Business Process Management (BPM) as 'Concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of business processes'. Recently BPM becomes indispensable for organizations to maintain competitive advantage because it provides organizations to improve the performance of business processes. Business processes are set of activities performed to realize organization's business goal in organizational and technical environment [**1**]. Rapid improvement of business processes satisfies changes in the industry, increases customer satisfaction, reduces business and service cost and makes easier to establish new products. Thus, from the business administration point of view BPM is necessity. Most of the business rules are supported by information systems to be performed and BPM helps to narrow gaps between business processes (organization) and IT systems (technology). Therefore, BPM is also critical for computer science communities. In organizational environment analysts investigates process requirements by interacting customers and models them as a set of process activities. Those processes are the abstractions of real world processes. Provided models from process abstractions provide to detect structural deficiencies beforehand. In this way, verified processes are provided to IT developers for technical design and realization. BPM provides organizations to manage changes in those business processes in an effective and efficient way. BPM includes the following steps [**26**]:

- **Identify changes in processes:** In this step, required changes in business processes are identified by managers. Those changes can be the result of

defining or modeling business process wrongly in organization level, being need of more efficient models to improve business performance or the necessity of creating new products.

- **Analyze existing processes:** In this step, modelers identify inefficiencies such as redundant steps, paper-intensive tasks and bottlenecks in the business process. Required changes are measured in terms of time and cost. Besides that results of those analyses are documented to make decision stage easier.
- **Design new processes:** The process is redesigned by determined changes taking into consideration. Redesigned processes are also modeled and documented in order to compare with the old processes.
- **Implement the new processes:** New set of procedures and work rules are defined for redesigned process. Either the process is implemented from scratch or existing systems are enhanced to support redesigned process. Additionally, optimization of the redesigned process is performed by developers.
- **Continues measurement:** Implemented and optimized processes are measured continuously in order to identify the necessity of change. If change is necessary BPM steps are performed from the first step.

In order to apply BPM effectively in an organization, key points given in the following should be considered [**27**].

- Major BPM activities have to be mapped and documented
- Customers needs have to be taken into consideration
- Processes should be relies on IT systems in order to be consistent and repeatable
- Measurement activities for processes should be well-defined to assess the performance
- Process improvements should be incremental and ongoing
- Best practices should be taken into consideration in order to achieve superior competitiveness

3.2 Business Process Modeling

Business process models are the main artefacts of Business Process Management. They visualize a set of activities and states that constitute a business process in their execution order [**28**]. Models include information (data), materials and resources that are used or produced during process execution. Execution constraints and business rules are also defined graphically in process models. They are used to analyze process efficiency and quality by business analysts and managers. Besides that, those models are the basis of process-aware Information Systems' construction [**29**]. They are used to analyze system requirements and to design system architecture.

There are numerous modeling languages to visualize system specifications and process execution. In this study, two of those languages are used for transformation; eEPC and S-BPM. Those business process modeling notations are well known and

established in research and practice. In the following sub-sections brief explanations about those notations are given.

3.2.1 eEPC

eEPC stands for "extended Event-driven Process Chain". EPC is developed by Scheer et al. within the ARIS (Architecture of Integrated Information Systems) framework at the Institute for Information Systems in Germany, in 1990s [**3**] [**4**]. EPC represents business process as an ordered graph which shows chronological sequence and logical interdependencies between elements. Basic elements of EPC are functions and events. By logical connectors business relevant decisions are visualized and complex control flows are modeled. Since its notation is easily understood by business people, it is preferred by them to plan, design, simulate and control their business processes. However EPC notation is inadequate to show data flows, responsibility of actors, the use of IT systems, etc. Therefore, EPC notation is extended with additional elements from the organization and data view, which is called eEPC (extended Event-driven Process Chain). It is relatively simple notation to model business processes and highly accepted by the practitioners from diverse areas for business process re-engineering, management and documentation.

3.2.1.1 eEPC Elements

eEPC models lays out business process work flows and visualize the flow of events and functions, performers of functions, inputs and outputs (products/services) of functions and supporting application systems. The core elements of eEPC notation are events, functions, process paths and logical connectors ("and", "or" and "exclusive or") (Figure 1).

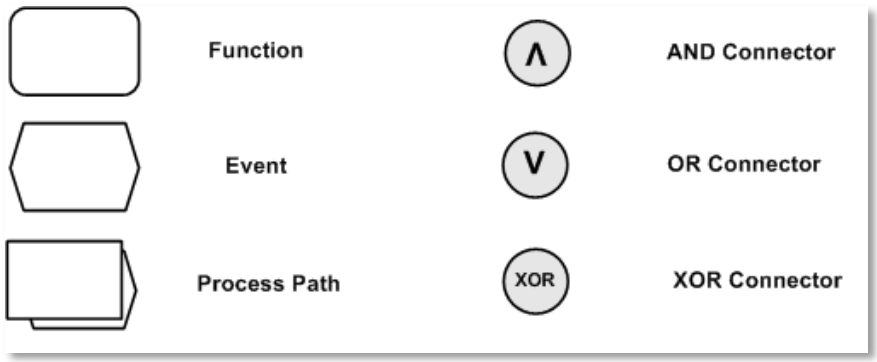

Figure 1. eEPC core modeling elements

- ***Events*** are passive elements, they shows the initial and final state of related function. There are three types of events in the EPC; start event, internal event and end event. Start event shows in what condition the business process starts. Internal events indicate pre-conditions and post-conditions of functions. End events show the result of the business process.

- ***Functions*** are active elements that show tasks or activities need to be executed to support a business goal. A function is triggered by an event and leads to the occurrence of an event when it is performed. In this way, functions describe transformation from the initial state to end state.

- ***Process Paths*** serves as navigation and establish a connection with other processes.

- ***Logical connectors*** show the logical relationships between functions and events in the control flow. They combine functions and events and connect those function-event combinations in order to represent alternative or parallel executions. They are also used to show decision stages and loops in the process. Each connector type can split one control flow into two or more control flows or can concatenate two or more control flows. There are three types of logical connectors; "and", "or" and "exclusive-or". Logical connectors can be categorized according to their usage patterns (Figure 2).

    - *Join_Functions*. This pattern includes a logical connector ("and", "or" and "exclusive or") with two or more incoming control flows coming from a function and one outgoing control flow going to event. "and" connector concatenates and synchronize active control flows (incoming) and activates the result event which occurs after accomplishment of all active functions. "or" connector shows that if one of the functions is accomplished, the following event is fulfilled. "exclusive-or" connector is used if accomplishment of exactly one of the functions is expected to fulfill the following event.

    - *Split_Function*. This pattern includes a logical connector ("and", "or" and "exclusive or") with one incoming control flow coming from a function and two or more outgoing control flows going to event. "and" connector splits post-conditions (output situations) which occur by accomplishment of the previous function and activates outgoing control flows in parallel. "or" connector is used to show that when function is accomplished, at least one outgoing control flows are activated. In other words, at least one post-condition is satisfied. And finally "exclusive-or" connector is used if exactly one of the events is fulfilled after the accomplishment of the function.

    - *Join_Events*. This pattern includes an "and" connector with two or more incoming control flows coming from an event and one outgoing control flow going to a function. "And" connector concatenates preconditions to activate the following function (outgoing control flow). "or" connector is used to show that if at least one precondition is fulfilled, outgoing control flow is activated by "or" connector. "exclusive-or" connector is used if the outgoing control flow is activated after exactly one of the events is fulfilled.

- *Split_Event*. This pattern includes an "and" connector with one incoming control flow coming from an event and two or more outgoing control flows going to a function. "and" connector activates the outgoing control flows in parallel when the precondition(event) is satisfied. It is not used with "or" and "exclusive-or" connectors.

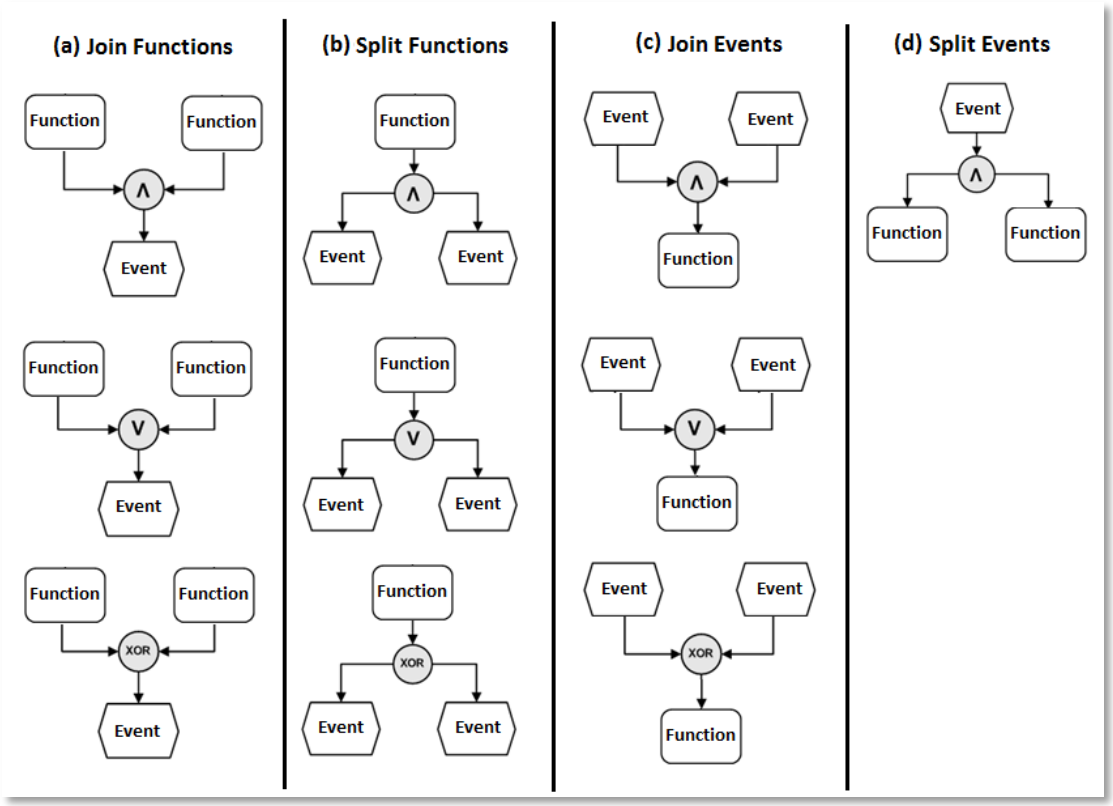

Figure 2. eEPC Logical Connectors' usage patterns

eEPC also includes additional notations for data and organization view (Figure 3). Data view includes information, material and resource objects which are related to functions but they do not have a chronological order on the process workflow. Document, list, log, product and file are data types which are produced as output after the execution of a function or used as input to execute a function. Application, reference and business rules are thought as resource objects which are used as a service.

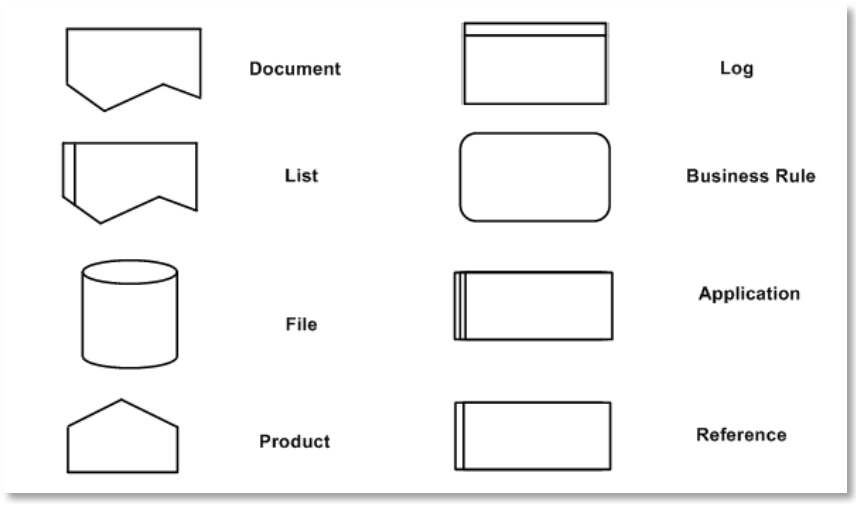

Figure 3. eEPC elements in data view

In organization view, organization unit, group and position elements are used as performers (Figure 4). If a performer is connected to a function, it shows who responsible for a function to execute. Besides that, it shows who send or receive data if a performer is connected to a data object. Organization unit refers the unit within the structure of the organization which is responsible for a specific business goal. Group are people who work together to perform a specific business process in the organization. Position is the smallest unit of an organization and it is assigned to employees.

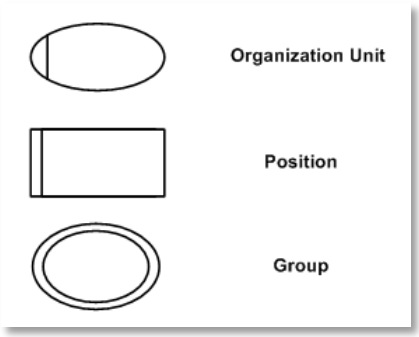

Figure 4. eEPC elements in organization view

In order to connect elements and visualize the flow between those elements, flow notations are used (Figure 5). Control flows show the transition between events, functions and process paths and constructs the business process as a chain. Information flows depicts the data flow between data and function or between data and process path. It can be used bidirectionally. If the source of information flow is a data object, which means that it is used by target function or process path as an input. If the target is a data object, it shows that data is produced by target function or process path as an output. Relation is used for organizational unit assignment, to show applications and references used during execution of connected function and to show constraints of a function.

Figure 5. Flow elements in eEPC

3.2.1.2 eEPC Modeling Rules

W.M.P. van der Aalst gives a formal definition which explains the requirements of an EPC element and defines the core elements as well [**30**]. Kees van Hee et al. define extended-EPC (eEPC) by providing syntax and semantics [**23**]. Those studies provide a foundation for our transformation and validation rules. In order to validate an eEPC diagram the following rules are used [**30**] [**23**]:

- There must be at least one start event,
- There must be at least one end event,
- All elements must be connected,
- All functions or process paths must have exactly one incoming and one outgoing control flow,
- Events cannot be consecutive to each other,
- Split connectors must have one incoming control flow and more than one outgoing control flow,
- Join connectors must have more than one incoming control flow and one outgoing control flow,
- An event cannot be followed by "OR" or "XOR" connector.
- Except start and end events, logical connectors should be used in pairs. Each logical connector block should be opened and closed by the same connector.

3.2.1.3 eEPC Metamodel

In this study, a subset of eEPC elements is covered for transformation. In order to elaborate those elements and their relationships, composed meta-model is depicted in Figure 6. It is based on the formal definition of EPC defined by W.M.P. van der Aalst [**30**] and eEPC Kees van Hee et al. [**23**]. According to our meta-model, a process consists of at least five process elements (start event, function, end event and control flows between them). Process elements can be workflow elements (function, event, process path, control flow, split connector and join connector) or extended elements (data object, resource object, actor, information flow and relation). eEPC workflow elements are consecutive to each other to form a process flow. Core elements (function, event and process path) are connected to each other by control flows. Data objects (document, list, log, product and file) are connected to functions or process paths via an information flow and they are connected to an actor via a relation. Relation also connects functions and process paths to actors and resource objects (application, reference and business rule).

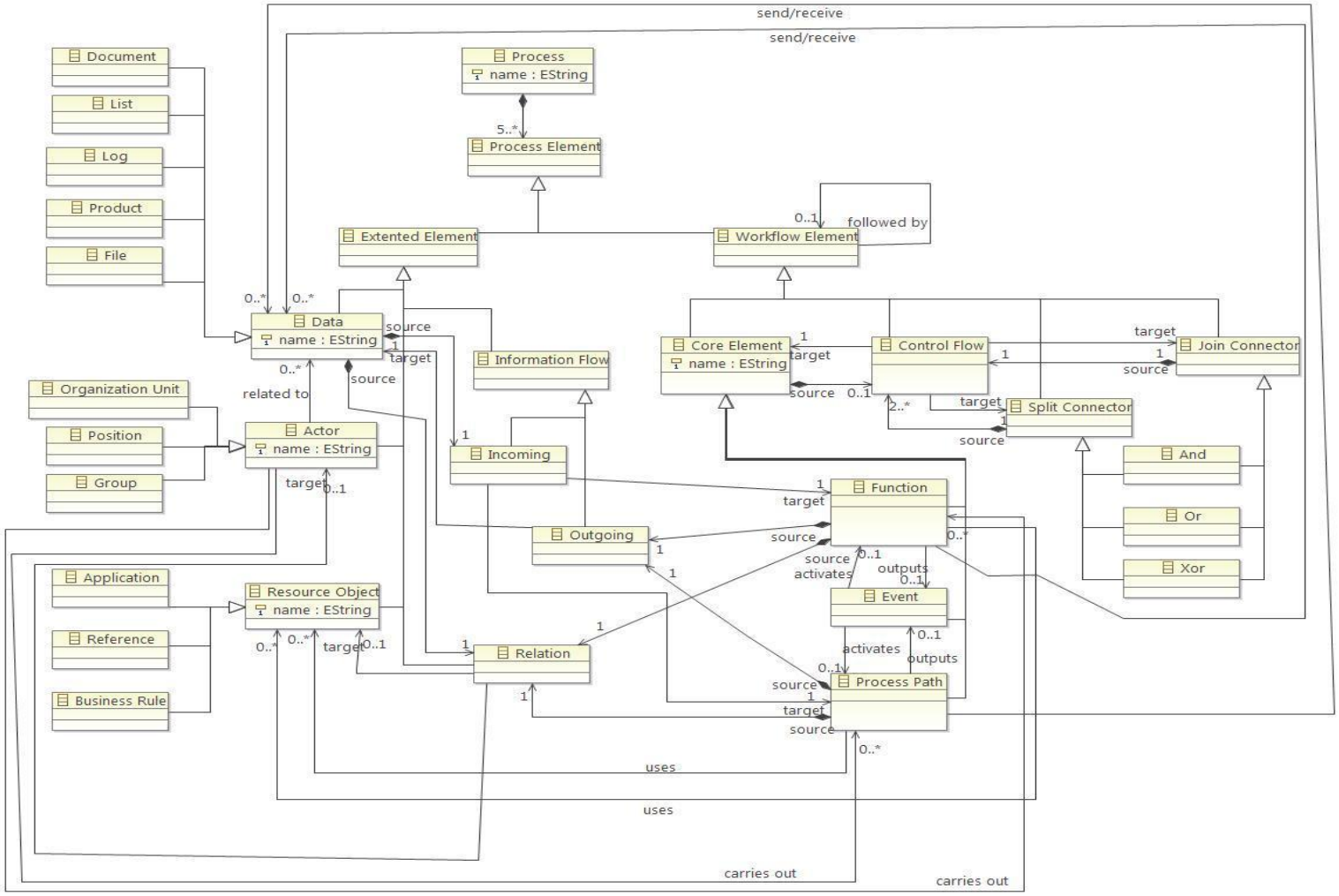

Figure 6. eEPC Metamodel from scratch for covered elements

3.2.2 S-BPM

S-BPM stands for "Subject-oriented Business Process Management". It is a new paradigm that is developed by Albert Fleischmann [**5**] to describe and execute business processes from the perspective of subjects. According to S-BPM, subjects are active elements in a business process. Therefore, they should be the starting point of the activities (like natural language sentences) [**6**]. S-BPM gets inspired from natural languages and the structure of S-BPM is similar to sentence structure of natural languages. The main motivation of this approach is that, task or process descriptions are always initially documented in natural language and they are complemented with diagrams [**31**]. In S-BPM, those diagrams can be directly derived from process descriptions in natural language representation and they show the communication between people and describe the activities of the people involved. Since natural language descriptions are understood by all people immediately, S-BPM models are also understood easily by nontechnical people.

3.2.2.1 S-BPM modeling procedure

S-BPM uses bottom-up business process modeling approach. In this approach; responsibilities of actors in organizations are defined firstly and in detail. Then the message exchange between these subjects is modeled in order to show whole process. S-BPM uses natural language sentence structure while modeling business processes as well. Natural language sentences are formed from Subject, Predicate and Object. Subjects are the starting points for describing a situation or a sequence of events, predicates are actions which are performed by subjects and objects are the targets of actions. While generating S-BPM models from natural language description of processes, following activities are performed [**32**];

- *Identify subjects involved in the process.* Subjects are the result of "Who acts?" question. Unique names of identified subjects with a brief description are documented in this phase.
- *Identify activities of subjects.* Activities are the result of "What does the subject?" question.
- *Identify business objects.* They are the result of "What edits the subject?" question. Business objects can be collections of materials, such as a list of documents, electronic forms, applications being used and data record and data element descriptions, etc. Those objects are attached to messages in Subject Interaction Diagrams (SID) and they are exchanged between subjects.
- *Detail behaviors of individual subjects.* In this phase Subject Behavior Diagrams (SBDs) are formed.

Figure 7 shows the natural language description of Business Trip Application process. The subjects are underlined, predicates are marked by rectangles and objects are marked by rounded rectangles. In this way subjects, activities and business objects are identified.

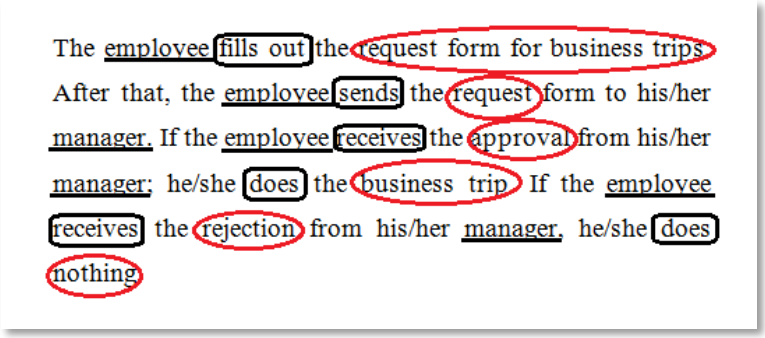

Figure 7. The natural language description of Business Trip Application process (adopted from [**5**])

In SBD, behavior of employee is detailed (Figure 8). All identified predicates are represented by subject states. Sequence of those predicates is also sequence of states in the diagram.

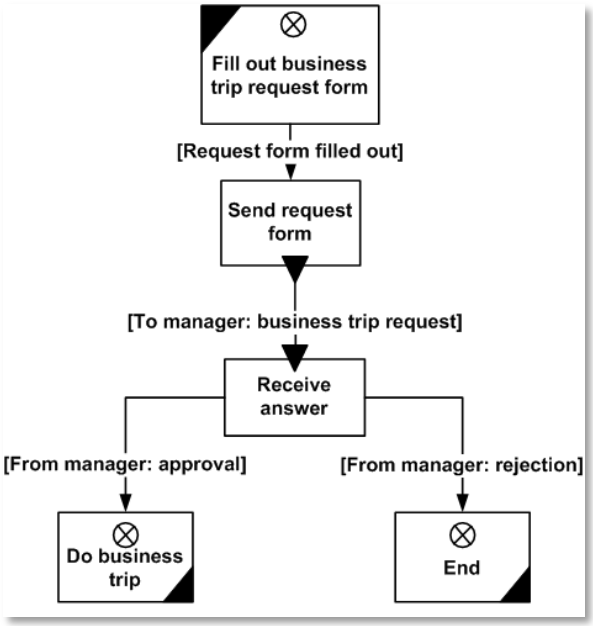

Figure 8. SBD of Business Trip Application process

Business trip application process is performed by employee and manager is a participant. Thus in SID, employee and manager should be represented as subjects. Business objects are transmitted between those subjects. In the second sentence business trip request is sent to manager by employee. In the following sentences it is shown that employee receive approval or rejection information from the manager. SID of the process is given in Figure 9.

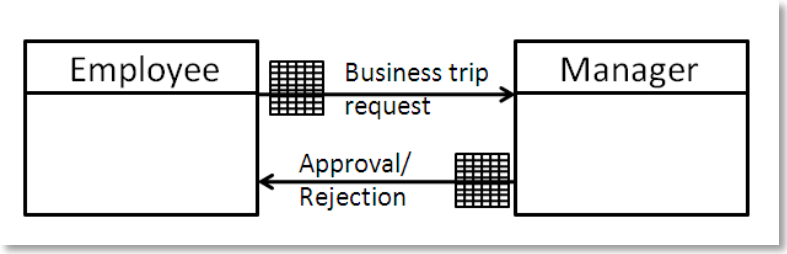

Figure 9. SID of Business Trip Application process

3.2.2.2 S-BPM Notation

While generating S-BPM models from natural language description of processes, sentences are analyzed and subjects, actions of those subjects and business objects are identified and finally Subject Interaction Diagram (SID) and Subject Behavior Diagrams (SBDs) are formed.

SID is also called as "Communication Structure Diagram (CSD)". SIDs show the process performed by more than one actor as a whole. They show subjects and message exchange between those subjects. Figure 10 shows the elements of SID. SIDs consist of three elements; Subject, Message and Business objects. Subject element is used to show actors or participants in business processes. Business objects are attached to message flow and they are used to visualize interactions between subjects during the execution of the process. Business objects are physical or logical "things" which are required to process business transactions.

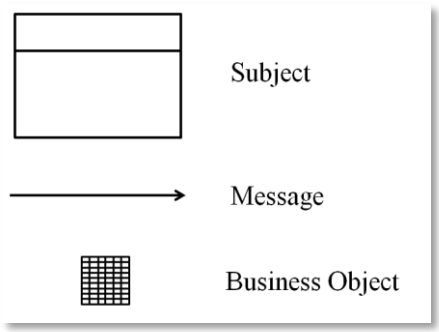

Figure 10. Elements of SID

SBDs show the internal behavior of a subject. SBDs consist of a series of states (send, receive and function state), macro classes and choice operators. Figure 11 shows the elements of SBD. Function, send and receive states can also be start and end states. Start states trigger the process and marked by a triangle in the upper left corner. End states are the last states of the process and when they are performed, the process is terminated. They are marked by a triangle in the lower right corner.

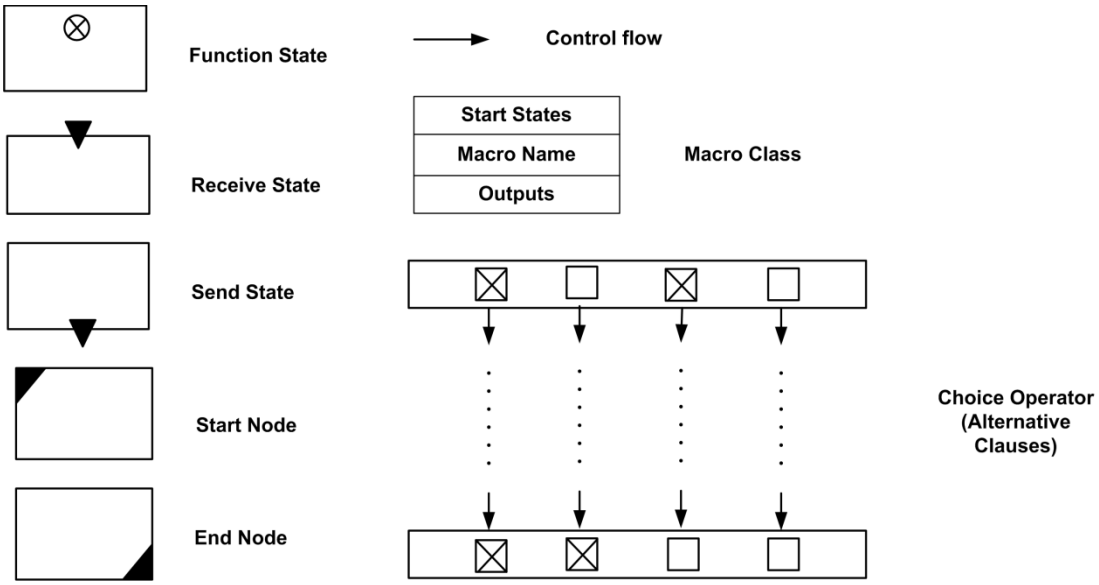

Figure 11. Elements of SBD

- ***Function states (Performing functions)*** represent internal actions of the subject (process owner). They are assigned to services. To reach function states, associated services should be executed. End conditions of the executed service correspond to the exits of the respective internal function state.

- ***Send states (Sending messages)*** are used to show sending messages (with business objects) to other subjects. Information of receiver (subject) and received data are shown in the outgoing transmission.

- ***Receive states (Receiving messages)*** are used to show receiving messages (with business objects) from other subjects. Information of sender (subject) and received data are shown in the outgoing transmission.

- ***Macro Classes*** are used to show sub-processes which are repeated in different SBDs to avoid redundant repetitions and includes behavior sequences. The notation of macro class consists of three parts, in the first part valid start states which activate the sub-process are shown. Name of the macro (sub-process) is shown in the second part. And in the final part the output of the sub-process are shown.

- ***Choice Operators*** provides to model overlapping actions without specifying strict sequence. They consist of a number of parallel paths which are activated simultaneously. Multipath structure starts and ends with a bar which includes beginning and end switches for each path. Set beginning switch means that related alternative path must be started and if it is not set alternative path does not have to be started. Set end switch means that related alternative path must be completed if it is started and if end switch is not set, alternative path does not need to be finished. Functions in the alternative paths may be arbitrarily executed in parallel and overlapping.

CHAPTER 4

S-BPM MODELING TOOL

Growing attention to BPM led to develop business process modeling tools since nineties. They provide to design, control and analyze business processes and support continuous improvement of processes. There are different tools for different modeling languages. For eEPC, ARIS toolset is used widely in the industry. For S-BPM Metasonic Suite is developed. In this study UPROM which is developed by Bilgi Grubu is used. It has different editors for various modeling notations, but it does not support S-BPM. Therefore, UPROM is extended for S-BPM notation and eEPC to S-BPM transformation. In the following UPROM and S-BPM plug-in are explained respectively.

4.1 UPROM

In Turkey, organizations do not perform business modeling activities effectively due to the high cost of commercial modeling tools and the lack of knowledge and expertise. Bilgi Grubu conducts studies for modeling and improving business processes of public institutions and software companies. In order to analyze, model and improve business processes UPROM is developed. It is an integrated business process modeling tool and it is used for modeling activities of Bilgi Grubu. It also provides to generate system requirements from business process models automatically. In this way more effective implementation of IT systems is supported.

UPROM is based on bflow* toolbox which is an open source tool source project contributed by at the University of Hamburg and the University of Applied Sciences Emden/Leer [**33**]. It supports eEPC, Object-oriented EPC and value chain diagrams. It is an Eclipse plug-in and provides graphical business process modeling in the EPC notation. It uses EMF (Eclipse Modeling Framework) and GMF (Eclipse Graphical Modeling Framework) technologies [**34**]. It makes use of the usual features provided by EMF and GMF like storing models as XMI files, collapsing and expanding modeling elements, aligning modeling elements, using the clipboard, etc. Additionally, it provides possibility to add new features easily.

While developing UPROM, new features are added to bflow* toolbox to use FTD (Function Tree Diagram), FAD (Functional Analysis Diagram), OC (Organizational Chart) and ERD (Entity-Relationship Diagram) notations. By using different notations, a system can be modeled from different perspectives and necessary

improvements can be determined easily. Automatic requirement generation is also added by implementing model to text transformation. UPROM uses EPC and FAD diagrams to generate the system requirements document.

4.2 S-BPM Editor

S-BPM editor is added to UPROM as a plug-in in the frame of this work. This plug-in provides to model business processes in S-BPM. A metamodel for S-BPM notation is composed as an ecore file. Graphical representations of ecore elements are realized by the help of GMF. Finally validation rules are defined in check language and continuous verification is satisfied. In the following details of constructed metamodel, graphical representations of SBD elements and validation rules will be given.

4.2.1 Metamodel of SBD

In S-BPM editor, firstly a metamodel for SBD is formed as ecore diagram (Figure 12). This ecore diagram includes all SBD elements which are used in process modeling. A SBD consists of elements and connections between those elements. “Element” and “Connection” classes are extended from “BflowSymbol” class. eEPC elements in UPROM are also extended “BflowSymbol” class. In this way, mapping SBD element attributes to eEPC element attributes becomes easier. Elements and connections include three main attributes that are come from “BflowSymbol” class. These are name, id and description. Name attribute of elements are visualized in the graphical model in contrast to id and description. “Element” class also has lists of in and out connections. This shows that an element can have more than one incoming and outgoing connections. On the other hand, connections can only have one source and one target element. Thus, “Connection” class has “to” and “from” attributes. Types of those attributes are “Element”, “to” holds the source and “from” holds the target of the connection.

Elements that are special for SBD are extended from “Element” class. Those are “MacroClass”, “SubjectState”, “UsedItem”, “AlternativesBar”, “OpenSwitch” and “ClosedSwitch”. “MacroClass” element includes sub-diagram name that holds the path of the sub-diagram, list of start states and outputs as additional attributes. “SubjectState” elements can be “FunctionState”, “SendState” or “ReceiveState”. They can also be start task or final task of the process. In order to hold this information “isStart” and “isEnd” attributes are added to “SubjectState” class. “UsedItem” element is not a core SBD element. It is defined as a new element for transformation to map application and business rules. “AlternativesBar”, “OpenSwitch” and “ClosedSwitch” are used to form choice operator (Alternative clauses). An alternative bar comprises open and closed switches.

In SBD, there are three types of connections; “SendArc”, “ReceiveArc” and “StateArc”. “Relation” is added to connect “UsedItem” element to others. “StateArc” is the basic control flow arc in SBD and “name” attribute is used to show post-conditions. “SendArc” has “receiver”, “data”, “receiverType” and “dataType” and these attributes are also shown in the graphical representation. “receiverType” is an

instance of “SubjectType” which is enumerations with values “Undefined”, “Group”, “OrganizationalUnit” and “Position”. “Receiver” attribute holds the name of the subject who received the data. “Data” attribute holds the name of the data and finally “dataType” holds the type of the data. Type of the data can be undefined, document, file, list, log, product or reference. “ReceiveArc” holds the same information with “SendArc” except receiver information. In “SendArc” name of the sender and sender types are held. They are also shown in the graphical representation.

4.2.2 Graphical User Interface

S-BPM editor provides modelers to construct process models graphically. Visual markers for core elements of SBD are satisfied by the editor. These elements are function state, receive state, send state, macro class, alternative bar, open switch, closed switch, and arcs (receive, send and state arc). There are also new elements which are added for transformation; used item and relation. Modelers can easily add elements by using drag and drop feature. Deletion and update features are also satisfied. When modelers change properties of an element, changes are applied to visual model elements and model is refreshed automatically. Figure 13 shows the graphical user interface of the S-BPM editor. Elements of SBD are shown in the right side of the editor. Properties of the selected element can be changed from “Properties” view.

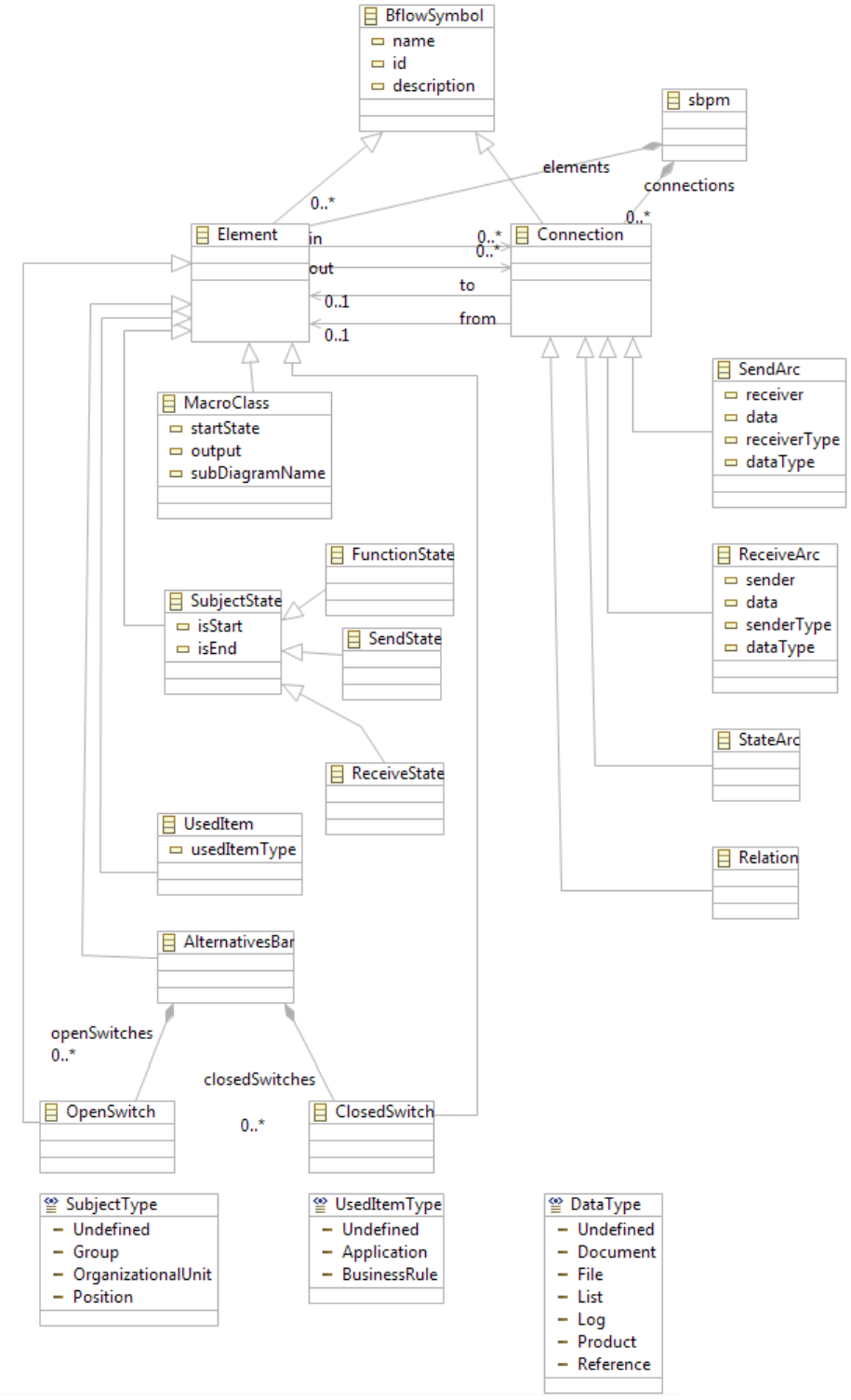

Figure 12. Metamodel of SBD

Editor is designed to avoid modeling errors during modeling time. Therefore, it does not allow every action during modeling. For instance, it does not allow connecting all elements to each other by using any arc types. Types of source and target element for each connection (send, receive and state) are pre-defined and connections can only

be established according to those pre-defined rules. Besides that, after each save action model is validated by pre-defined validation rules and information about errors and warnings are added to the Eclipse problem view. The details of model validation will be given in the next section.

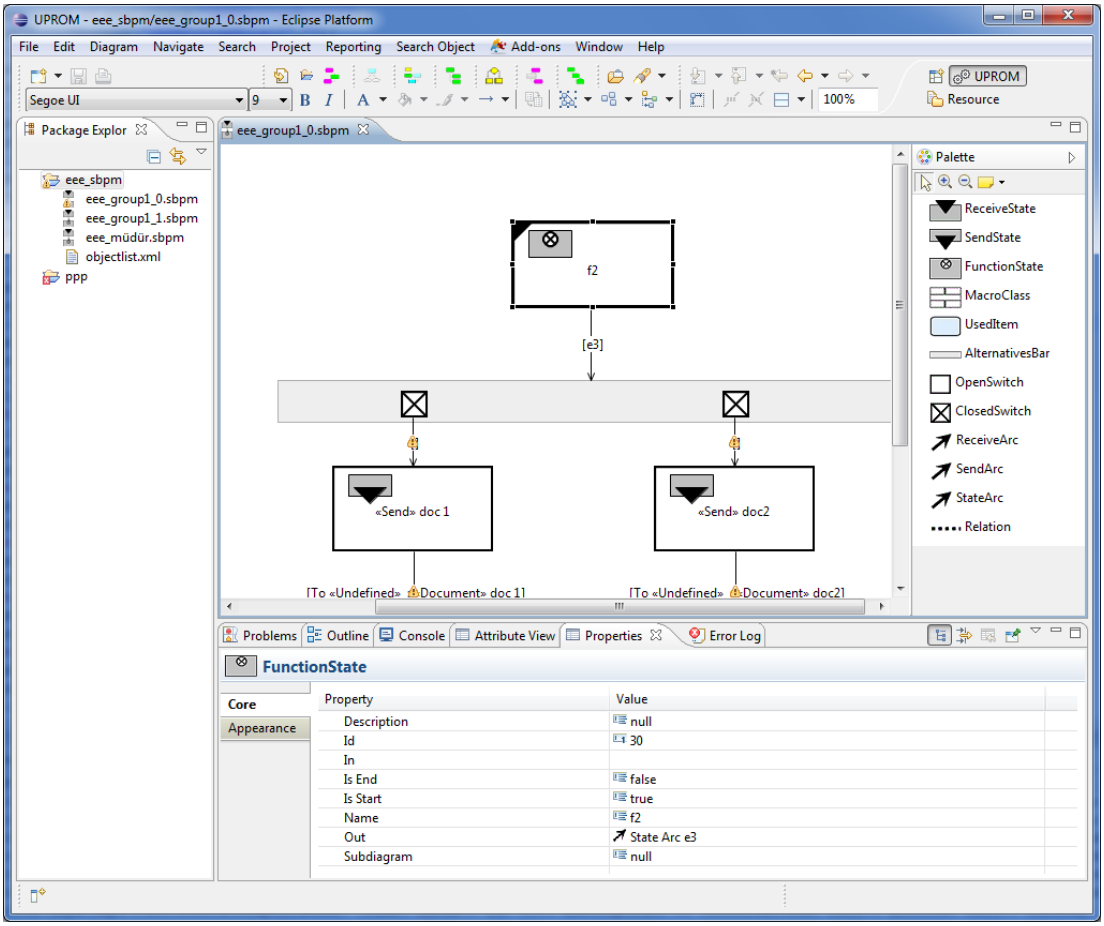

Figure 13. GUI of S-BPM Editor

4.2.3 Validation

Model validation should happen as early as possible in development process. Therefore; syntax and semantic rules should be defined for business process models and automatic validation should be satisfied by modeling tools. Continuous validation for SBDs is satisfied for modelers by UPROM. The modeler gets feedback about possible modeling problems continuously. In S-BPM editor validation rules are detailed and are implemented by Check language as constraints. Check language is a domain specific language and introduced for model validation. It is based on OpenArchitectureWare (oAW) expressions framework [**35**]. The content of the check file and extension file which is used by check file is given in Appendix A. In the following constraints defined in S-BPM editor will be explained.

Constraints for SBDs;

- Function states should not be start and end state at the same time
- Receive states should not be start and end state at the same time
- Send states should not be start and end state at the same time
- Function states should have at least one incoming arc or should be marked as start state
- Receive states should have at least one incoming arc or should be marked as start state
- Send states should have at least one incoming arc or should be marked as start state
- Start states should not have an incoming arc
- End states should not have outgoing arc
- Function states should have at least one outgoing arc or should be marked as end state
- Receive states should have at least one outgoing arc or should be marked as end state
- Macro classes should have at least one outgoing arc and at least one incoming arc
- Alternative bars should have only one incoming or one outgoing arc
- Switches should have only one incoming arc or one outgoing arc
- Alternative bars should have at least two switches (two alternative paths).
- There should be at least one start state in the model.
- There should be at least one end state in the model.
- Elements should have a name.
- Connections should have a name.
- Macro classes should have a name.
- Function states should have only state arcs as outgoing control flows
- Receive states should have only receive arcs as outgoing control flows
- Send states should have only send arcs as outgoing control flows
- Macro classes should have only state arcs as outgoing control flows
- Open switches should have only one state arc as outgoing control flow
- Closed switches should have only one state arc as outgoing control flow
- Alternative bars should have only one state arc as outgoing control flow
- Used items should be connected to an element
- Used items should have only relations as outgoing control flows
- Macro classes should have at least one start state
- Macro classes should have at least one output
- Sources of state arcs should not be Receive States, Send States or Used Items
- Sources and targets of state arcs should not be the same
- Targets of state arcs should not be Used Items
- State arcs should not connect Alternatives bars and switches to each other.
- Sources and targets of send arcs should not be the same
- Sources of send arcs should be Send States
- Targets of send arcs should not be Used Items

- Sources and targets of receive arcs should not be the same
- Sources of receive arc should be Receive State.
- Targets of receive arcs should not be Used Items
- Sources and targets of relations should not be the same
- Sources of relations should be Used Items
- Targets of relations should be Function States or Macro Classes
- Targets of relations should not be Used Items
- Function states should be connected to only one alternative bar or switch as a source.
- Receive states should be connected to only one alternative bar or switch as a source.
- Send states should be connected to only one alternative bar or switch as a source.
- Function states should be connected to only one alternative bar or switch as a target.
- Receive states should be connected to only one alternative bar or switch as a target.
- Send states should be connected to only one alternative bar or switch as a target.

After transformation is completed, those constraints are checked by S-BPM editor and gives feedback to users whether the model is valid or not. In this way, syntax of the generated model is validated.

CHAPTER 5

eEPC to S-BPM TRANSFORMATION

This chapter presents the proposed solution to transform eEPC models to S-BPM models. Section 5.1 explains the model transformation approach. Mapping rules defined for transformation are given in Section 5.2. In Section 5.3 transformation algorithm applied in this study is explained in detail by the help of flow charts.

5.1 Model Transformation

For the automation of eEPC to S-BPM mapping model-to-model (M2M) transformation technology is used. Since different domains have different models, which may or may not conform to same metamodel, transformation from one model to another model is usually needed. M2M transformation increases the reusability since developers use the existing models and make little changes on them. A model transformation which is shown Figure 14 takes a source model and transforms it into a target model by using predefined transformation definition [**9**]. Both models conform to their respective metamodels. A transformation is defined with respect to the metamodels. The transformation definition is executed on concrete models by a transformation engine. In this case, eEPC is the source model and S-BPM is the target model. Metamodels of those models are the extension of bflow metamodel. In order to automate eEPC to S-BPM transformation a plug-in in the existing transformation engine (UPROM) is developed.

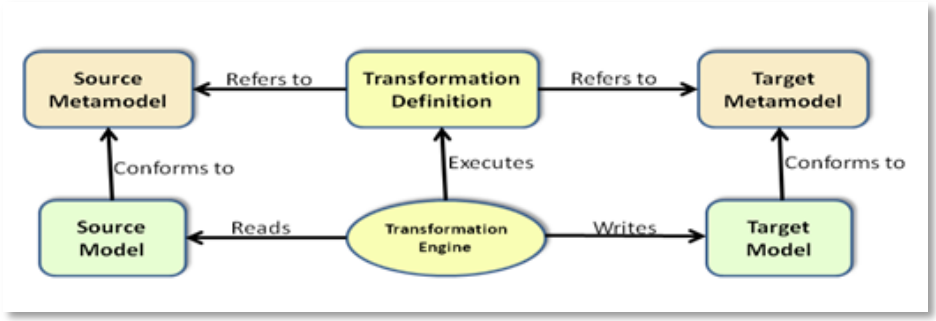

Figure 14. Model Transformation

eEPC and S-BPM differs the most in the aspect of adopted modeling techniques. eEPC uses flow-oriented approach. Due to that, it is generally considered as a kind of flowchart. It visualizes the sequence of tasks which are performed by different actors. On the other hand, S-BPM uses subject-oriented modeling technique which

means that it focuses on subjects (actors) and their relationships. In eEPC, a business process performed by more than one actor can be visualized by one eEPC diagram. However, in S-BPM that business process is visualized by SID in higher level and internal activities of each subject are shown in separate SBDs in lower level. For that reason when an eEPC model is transformed, more than one S-BPM model are generated. The number of generated models depends on consecutive actions performed by different actors. For each consecutive actions performed by the same actor, an SBD is generated for each individual model. Input eEPC diagrams which will be transformed are assumed to be valid according to syntactic and semantic rules defined in [**30**] [**23**]. Transformation can be performed automatically by the new version of UPROM. Validation of eEPC models are also check by UPROM automatically.

5.2 Mapping Rules

SBD generation starts from the root node and follows through the nodes in sequential order. The events without incoming control flow, the events without outgoing control flow and function-event pairs are taken into account firstly. In the conversion of function-event pairs, relations of the function with other elements (data and resource object) are inspected. Matched patterns (defined in following subsections) converted into respecting target patterns. In this section, mapping rules for eEPC elements will be explained in detail.

Functions and Events

Functions and events are the most crucial elements of eEPC. Functions represent tasks or activities which are executed by organization units, groups or positions. Events show the state of the process. They are triggered by functions and they also trigger functions as well. Function-event pairs show the flow of the business process. Events are problematic in transformation because there is not a corresponding element in S-BPM to map. Figure 15 shows the mapping rules defined for start, end and internal events.

In the eEPC diagram there can be events without incoming control flow which means that event is not triggered by a function. They are interpreted as start event and mapped to a dummy start function with **«Start»** annotation (Figure 15.a). Since an eEPC diagram has to start with an event which triggers the business process to start, generated S-BPM diagram starts with this dummy start function. Events which have no outgoing control flow are interpreted as end event of the process since they show in what condition the business process is completed. They are mapped to a dummy start function with **«End»** annotation (Figure 15.b). Generated S-BPM diagram ends with this dummy end function. Internal events are triggered by a function or process path and they also trigger the next function or process path. Events with incoming and outgoing control flows are interpreted as internal events and mapped to a control flow (state arc) with a label that includes event description (Figure 15.c).

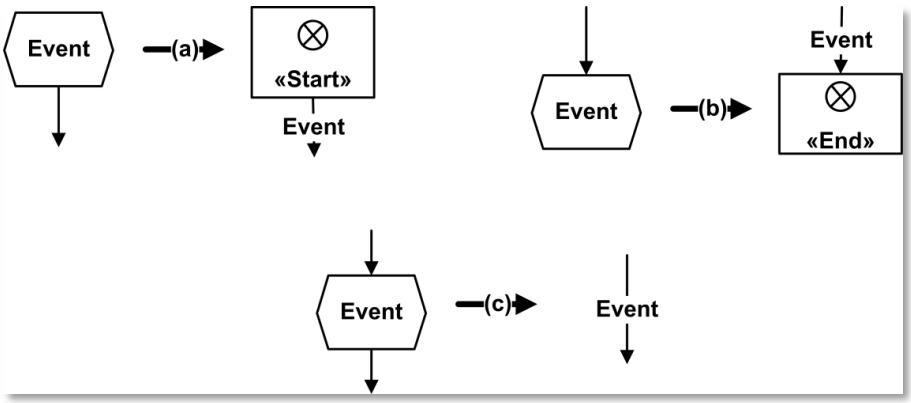

Figure 15. Mapping rules for eEPC events

Internal events and the function or process path which triggers it are transformed together during transformation. Mapping rules for functions is shown in Figure 16. If a function follows by an event, it is transformed as a performing action (function state) element of S-BPM and following event is transformed as text on the outgoing control flow label (Figure 16.a). Functions without following event are mapped to performing action element and a control flow without any label (Figure 16.b).

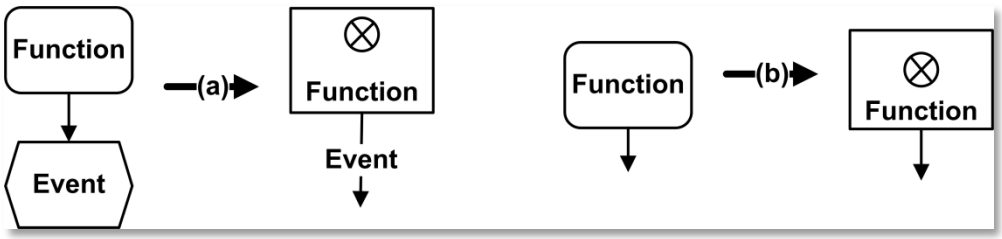

Figure 16. Mapping rules for eEPC functions

Data and Subjects

In the eEPC there are many different types of data and subjects, however corresponding representations of those objects are not available in SBD. Transforming these elements by ignoring these types causes information lost. In order to overcome that, annotations for data and subject types are introduced.

In the eEPC subjects can be connected to functions, process paths or data objects. During transformation subjects which are connected to functions or process paths are used to divide eEPC model into sub-models. These types of subjects are not shown in generated SBD diagrams. However; if they are connected to data, subjects are considered as sender or receiver according to the direction of information flow. Those subjects can be seen on receive arcs or send arcs in SBD diagrams with respective annotation. Table 1 gives the subject type annotations. Transformation of those subjects will be explained in the following sub-section.

Table 1. Annotations for Subjects types

Subject Type	Annotation
Organization Unit	« OrganizationUnit »
Group	«Group»
Position	«Position»

In the EPC a function or process path may use information, material and resource objects as input or may produce them as its output. If those objects are connected to functions with outgoing control flow or incoming control flow, they are considered as sending and receiving messages. Lacking of corresponding S-BPM notations for those objects leads us to use annotations to represent them. Table 2 gives annotations for information, material and resource objects.

Table 2. Annotations for Information, Material and Resource Objects

Information, Material and Resource Objects	Annotation
Document	«Document»
List	«List»
Product	«Product»
File	«File»
Log	«Log»
Application	«Application»
Reference	«Reference»
Business rule	«BusinessRule»

Document, list, product, file and log can be seen as a data object. However; application, reference and business rule are resource objects. Applications are systems and supports functions for execution. References (laws, regulations, standards, guidelines, etc…) are used to provide information to execute related function. Business rules restrict the operations of functions. For resource objects, notation with "Used" keyword and respective annotation is used (Table 2). The description of resource object is also given as a part of this notation. Resource object notation is connected to functions with a dotted line (relation arc) (Refer to Figure 17).

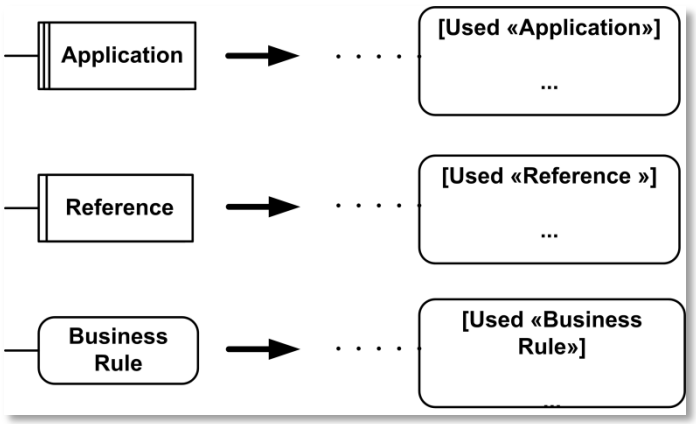

Figure 17. Resource object transformation rules

Receive/Send Data

In the eEPC, there are data objects which are used by functions (input) or produced by functions (output). Direction of the information flow between data object and function (or process path) shows whether it is a receiving or sending data. Data objects do not affect the process flow. In other words a function can receive/send one or more data in any order. In S-BPM notations for data objects (Data and Business Object) are used to visualize message flow between subjects and shared data objects between subjects in SIDs, there is not a graphical representation of data object in SBD. In SBD, data objects are stated on control flow as a label. Textual structure of that label is set according to direction of the information flow. Accordingly, one to one mapping is not an option for data objects. "Receiving Message" and "Sending Message" elements are used during the conversion of data-subject pairs. Since the data objects are not ordered in the process flow, it's assumed that all receiving messages take place before the concerning function and all sending messages come after the function in the flow.

A pattern which consists of data with outgoing information flow in the eEPC diagram is transformed to "Receiving Message" element in S-BPM. Name of the element is given as "«Receive» data name" automatically. Subject and related data which is received from that subject is shown as a textual notation ("[From Subject «related annotation»: Data «related annotation»]") on the outgoing control flow label. If there is only one data received by the following function, one receive state with an outgoing receive arc is generated (Figure 18.a). However; if there is more than one data received by following function, alternative clauses to combine those receiving messages is used (Figure 18.b). For each input data an alternative path is generated. This path consists of a receive state, an outgoing receive arc and two closed switch at the beginning and end. This alternative clause structure gives the meaning of "and" operator. It guarantees that all received states are performed (all data are received) before activating the function. If there is no subject connected to the data object, «undefined» annotation is used for this missing subject. For data with outgoing information flow without any subject relation, the textual structure "[From «undefined»: Data]" on the receive arc is used for "Receiving Message (receive state)" element (Figure 8.c).

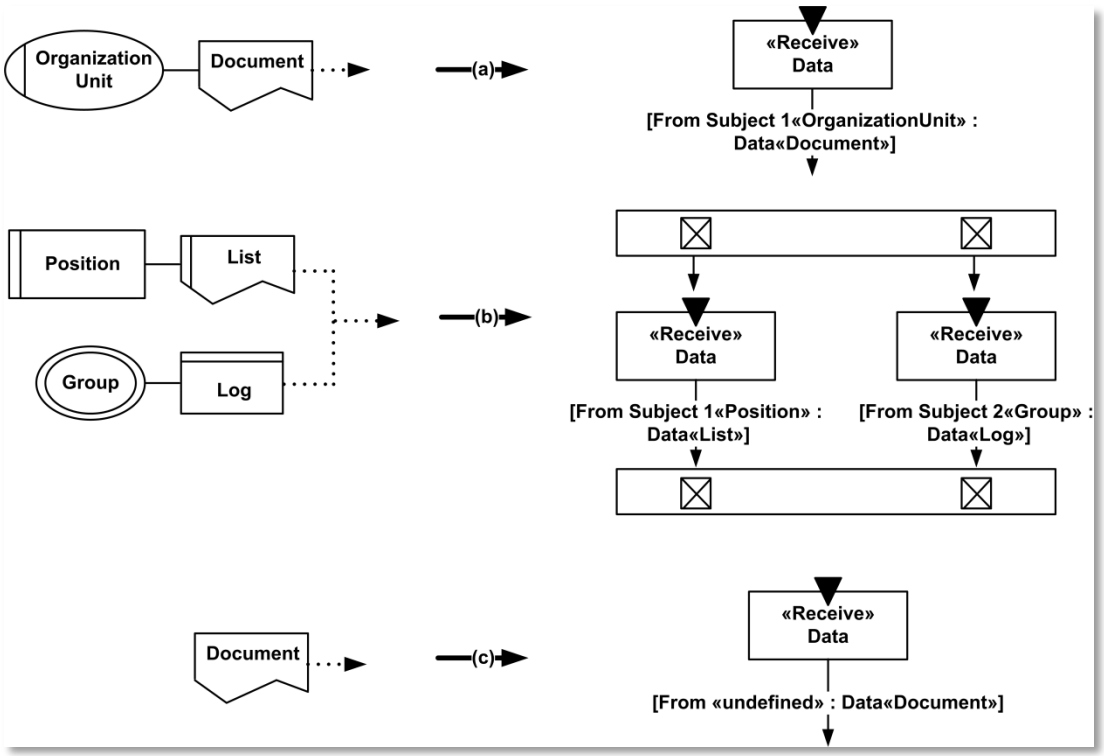

Figure 18. Mapping rules for input data objects

"Sending Message" element is used to transform data with incoming information flow. Name of the element is given as "«Send» data name". Subject and related data which is sent to that subject is shown as a textual notation ("[To Subject «related annotation»: Data «related annotation»]) on the outgoing control flow label. If the function produces only one data, a "Sending Message (send state)" element with an outgoing send arc is generated (Figure 19.a). If more data are produced by the function, alternative clause is used to combine sending messages likewise multiple receiving messages mentioned before (Figure 19.b). In this way, it is guaranteed that all produced data is sent before activating the next function. If there is no subject connected to the data object, «undefined» annotation is used for this missing subject ("[To «undefined»: Data]") (Figure 19.c).

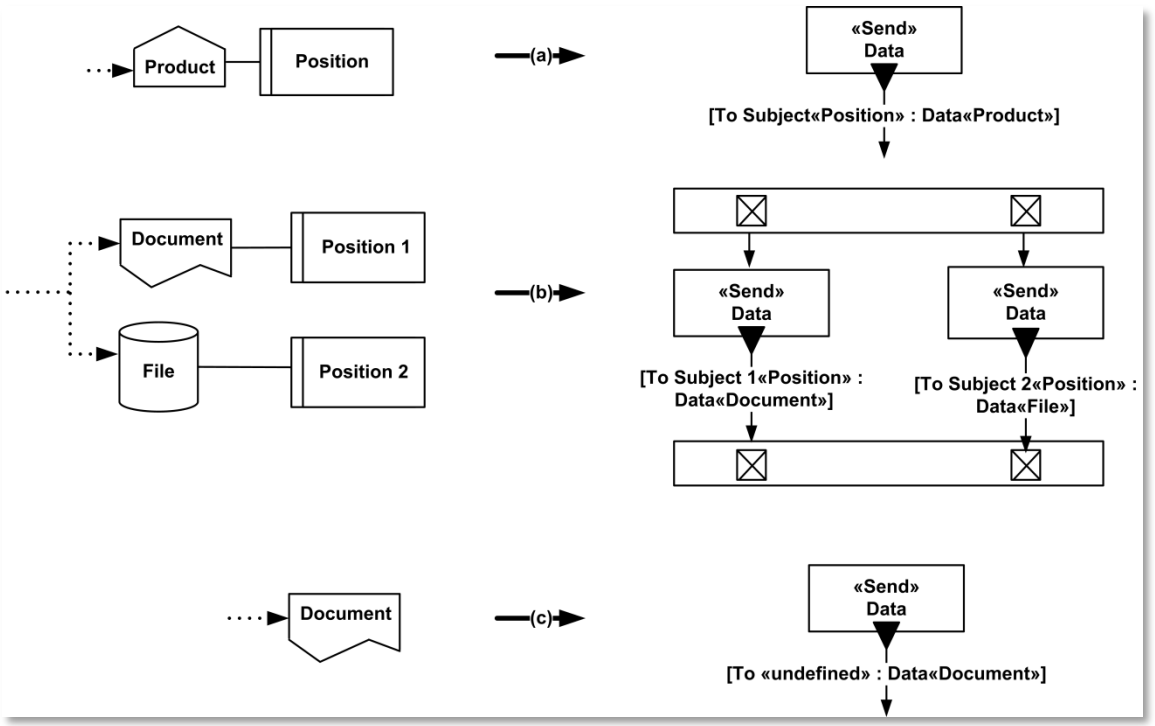

Figure 19. Mapping rules for output data objects

Logical connectors

Logical connectors show the logical relationships between functions and events in the control flow. They are used to split one control flow into two or more control flows or to concatenate two or more control flows. In the eEPC there are three types of logical connectors; “and”, “or” and “exclusive-or”. Since there is not any element in S-BPM with the same behavior to map logical connectors, conversion of them is the most problematic part of the transformation process. Three new functions with “«and»”, “«or»” and “«xor»” annotations are defined in order to use as logical connectors. In S-BPM functions with more than one incoming control flows and more than one outgoing control flows are possible. Therefore, “Performing Action” element with a respective annotation is used to depict logical connectors. “Performing Action” with «and» annotation awaits all incoming controls flows arrive and activates all outgoing control flows. “Performing Action” with «or» annotation awaits at least one incoming controls flow arrive and activates one or more outgoing control flows. “Performing Action” with «xor» annotation awaits exactly one incoming control flow arrive and activates one outgoing control flow. In the following, mapping rules about logical connectors grouped by behavior will be explained.

Join Functions connectors join the incoming control flows (their sources are functions) and activate the result event. “or”, “and” and “exclusive-or” connectors can be used for this purpose. Logical connector which join functions is mapped to performing action elements with «and», «or» or «xor» annotations according to type of the connector. These elements have incoming control flows without any label and

an outgoing control flow with a label that includes the description of the output event of join function (Figure 20).

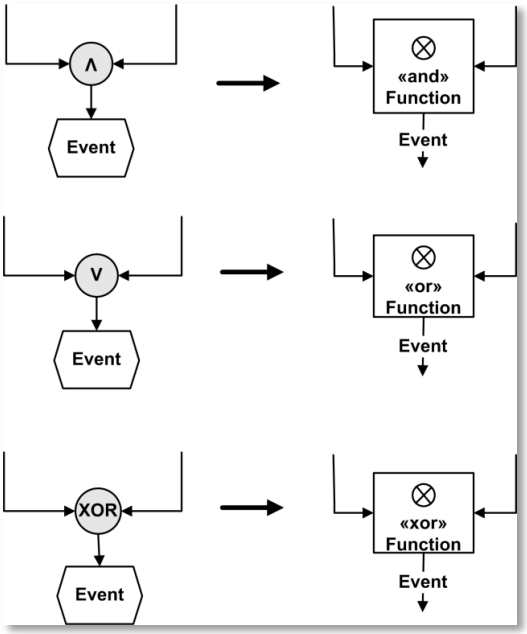

Figure 20. Mapping rules for join functions connectors

Split Function connectors activate two or more post-conditions (events) of a function. “or”, “and” and “exclusive-or” connectors can be used for this purpose. Split Function connectors is mapped to performing action element with «and», «or» or «xor» annotations. These elements have one incoming control flow and outgoing control flows with a label that includes event description (Figure 21).

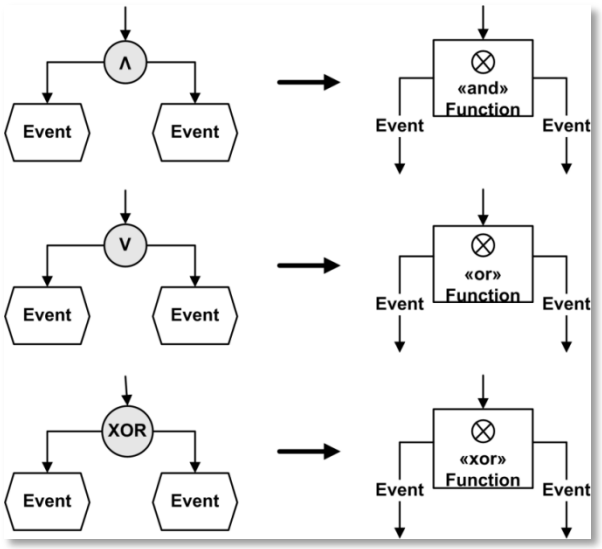

Figure 21. Mapping rules for split function connectors

Join Events connectors concatenate preconditions (events) and activate the following function. “or”, “and” and “exclusive-or” connectors can be used to join events. Join Events connectors are mapped to performing action element with «and», «or» or «xor» annotations. These elements have incoming control flows with a label that includes the description of precondition (event) and an outgoing control flow without any label (Figure 22).

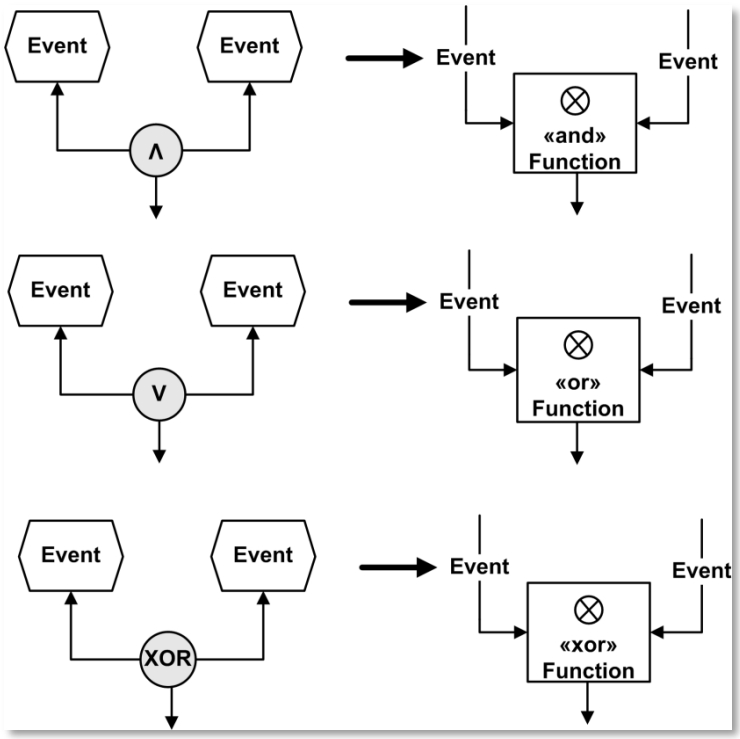

Figure 22. Mapping rules for join events connectors

Split Event connector activates all outgoing control flows when the precondition (event) is satisfied. Only “and” connector is used to split events. In S-BPM, Split Event connector is transformed as performing action element with «and» annotation. It has an incoming control flow with a label that includes the description of precondition (event) and outgoing control flow without any label (Figure 23).

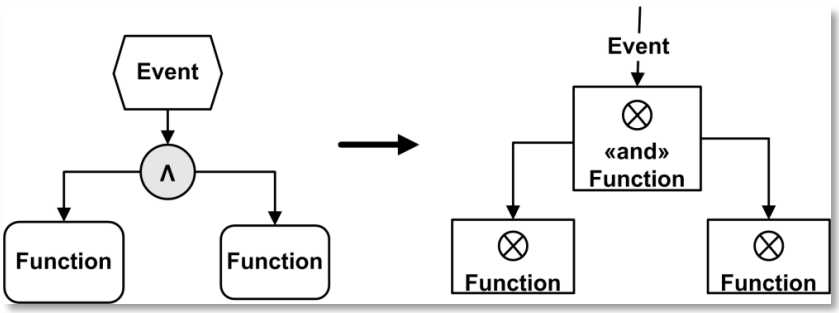

Figure 23. Mapping rule for split event connector

Process Path

Process path navigates the control flow to sub-processes which are modeled in a separate eEPC diagram. Mapping rules defined for function element are also applicable to process path element. In contrast to function element, process path element is mapped to macro class. In the notation of macro class element, valid start states, name of the macro and possible output transitions are shown respectively. In transformation, firstly "subdiagram" property of process path element is read. If the sub-process exists, the given path, the model of the sub-process is read. The start events of the sub-process' model are set as start states of macro class and end events are set as output transitions (Figure 24). If the model of the sub-process is not in the given location, «undefined» annotation is used for missing information.

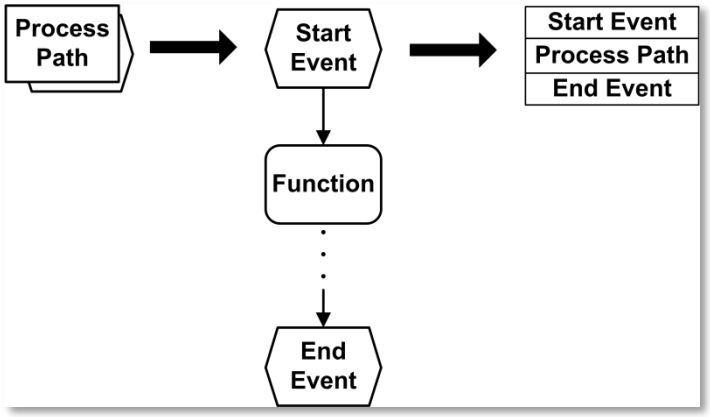

Figure 24. Mapping rule for Process Path element

5.3 Algorithm

In this section complex parts of transformation algorithm are explained by the help of flow charts. Transformation begins from the one of the start state of eEPC model and follows the control flow to the end state. In other words, control flow paths from start event to end event are transformed in order. This procedure is applied for each start event in the eEPC model. There are also sub-paths in models which are the results of split connectors. All encountered sub-paths during transformation are collected in a "paths" list and after transformation of path is completed, sub-paths in that list are transformed in order. Transformation continues until transformation of all paths and sub-paths are completed. If functions or function sets are encountered performed by different actors during transformation of paths, those functions or function sets are collected into "models" list in order to be transformed as a separate model. There can be more than one S-BPM diagram for an actor in eEPC. Since, if functions accomplished by a specific actor are not consecutive, determining the order of those functions is not possible. Therefore, consecutive functions and individual functions accomplished by one actor are transformed separately as a new S-BPM model. This general transformation algorithm is given in Figure 25.

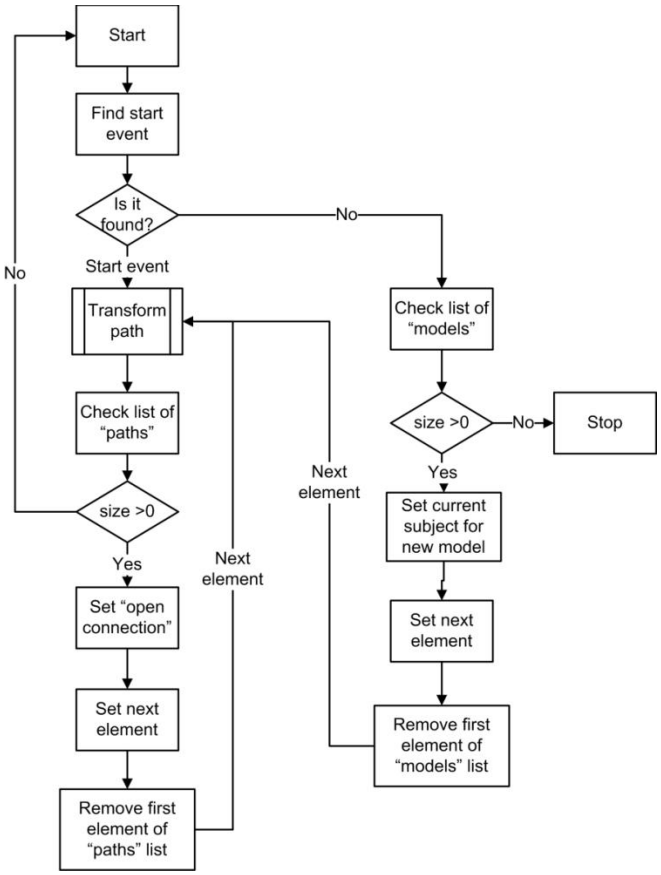

Figure 25. Transformation of eEPC models

The details of "Transformation of path" procedure are given at Figure 26. This procedure takes the start element as an input which can be event, function, process path or logical connector and transform each element in the control flow as long as there is a next element. If element to be translated is event, the type of the event is taken into consideration. Start events are transformed as a function state and a state arc as mentioned previous section. The source of the state arc is set as start event and name of the start function is set as name of the state arc. Created arc is saved as "open connection" in order to connect next transformed element and find the next element to continue transformation. If element is end event, end function state is generated, "open connection" is connected to this function state and transformation of path is terminated. If element is internal event, next element is found and transformation continues from next element. Since internal events are transformed with previous functions connected to them.

If the element to be translated is function or process path, subject who is responsible from that element becomes critical. If subject of the element is different than the subject who is the owner of transforming model at that moment, the element is added to "models" list to be transformed later and the next element is fetched. If not, transformation of the element is performed and the next element is fetched. If the element is a logical connector, "Logical connector transformation" procedure is performed.

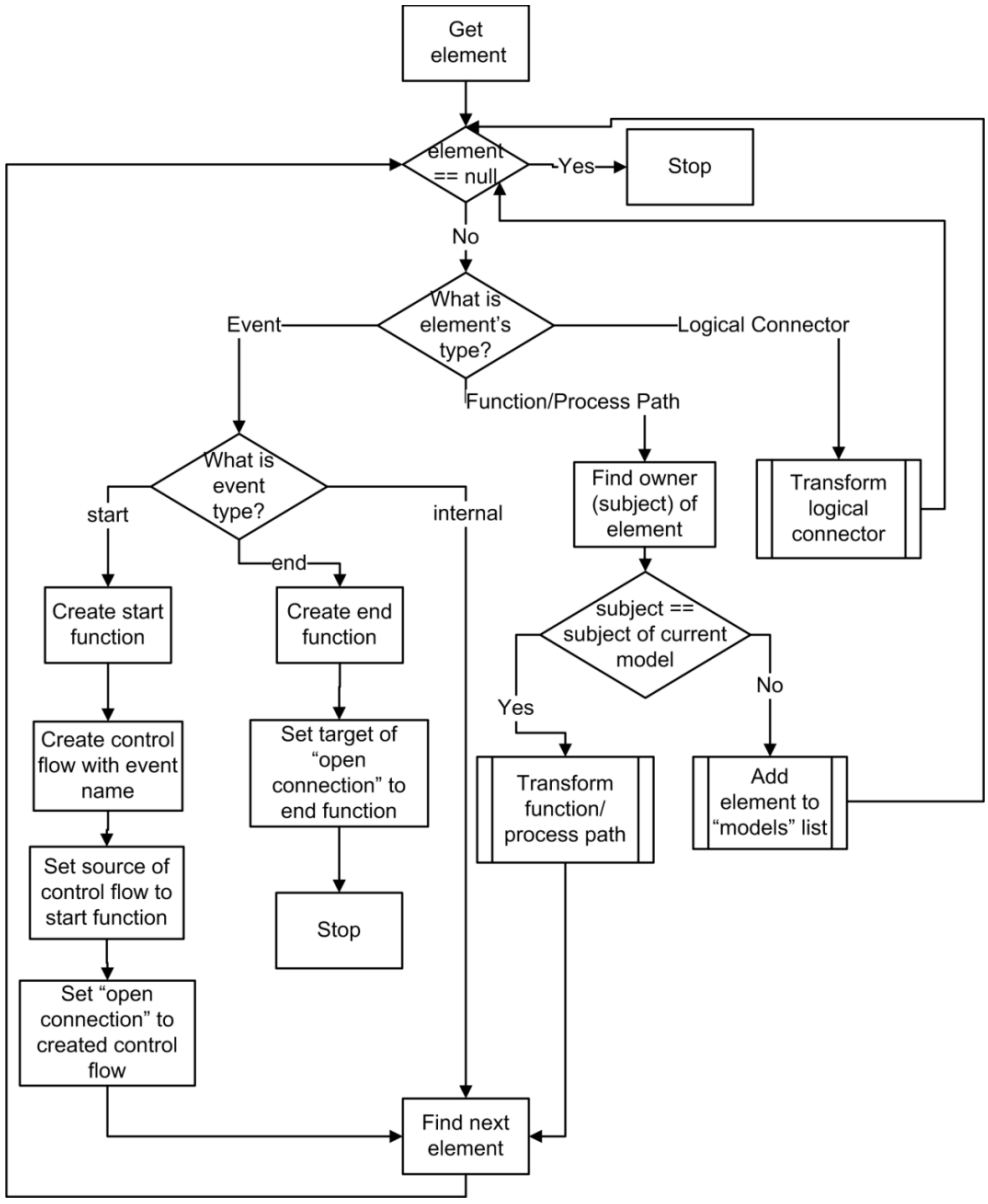

Figure 26. Transformation of path

Functions and process paths are the most complex elements to transform. Since they are connected to information, material and resource objects with information flows or relations. While transforming those elements, all incoming information flows are thought as received data and are transformed first. If the number of incoming information flows are greater than zero, received data part is generated. Receive data part can include one receive state or an alternatives block with more receive states. It also includes an outgoing arc to connect the next transformed element. This arc can be a state arc or a receive arc, it changes according to last element of the received data part. If receive data part is generated, its first element is set as the target of "open connection". If there is not an "open connection", it means that it is a start element of a sub-path. Thus, the "isStart" property of generated function is set to true. If element is a function, a function state is generated and the target of "open connection" is set to this element. If it is a process path, a macro class element is generated and the target of "open connection" is set to this element. While generating

macro class "subdiagram" property is used to get the location of sub-diagram. If the sub-diagram exists in the given location, starts events of that diagram are set as start states and end events are set as outputs of macro class. If not, start states and outputs are marked as undefined. After the element transformed, a state arc with the name of following event is generated and saved as "open connection". The source of this arc is set to the transformed element (function state or macro class). Then sent part is generated likewise received part. "open connection" is connected to the first element of the send part. The outgoing arc is also generated and it is set as "open connection". (Figure 27)

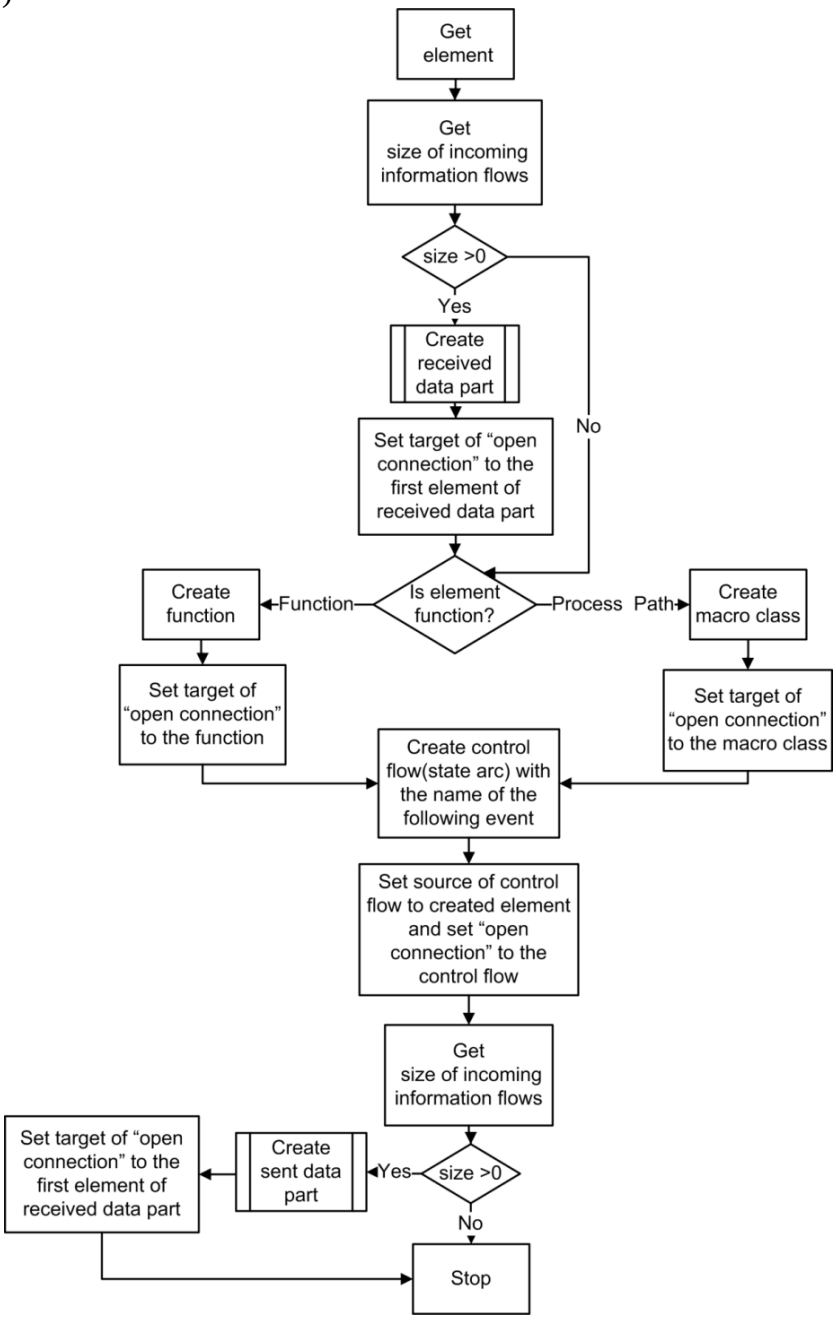

Figure 27. Transformation of Function and Process Path element

In "Logical Connector Transformation" procedure is shown in Figure 28. A "logical connector mapping" list keeps the mapping information of all transformed

connectors. This list keeps eEPC element and generated S-BPM element (function state, connection or closed switch) which corresponds to that eEPC element. While transforming logical connectors, firstly the "logical connector mapping" list is checked to identify whether it is transformed before or not. If it is transformed before, the target of the open connection is set to mapped S-BPM element. This situation can be occurred for join logical connectors. Join connectors can be transformed during the transformations of previous paths. If a pre-transformed join connector is encountered, this means that it is the end of transforming path. Therefore, the transformation is terminated at that point. If the logical connector is not transformed before, the transformation is performed according to its usage. "OR","XOR" and "AND" connectors are transformed as function state. Generated function state is added to "logical connector mapping" list. Target of "open connection" is set to that function state. New state arcs with the name of following events are generated for each outgoing control flow of eEPC logical connector element. Sources of all generated connections are set to the function state. They are added to "paths" list as sub-path in order to be transformed in next iterations except the first one. The first state arc is set as “open connection” to continue the transformation of the current path and the next element which follows the first connection is fetched.

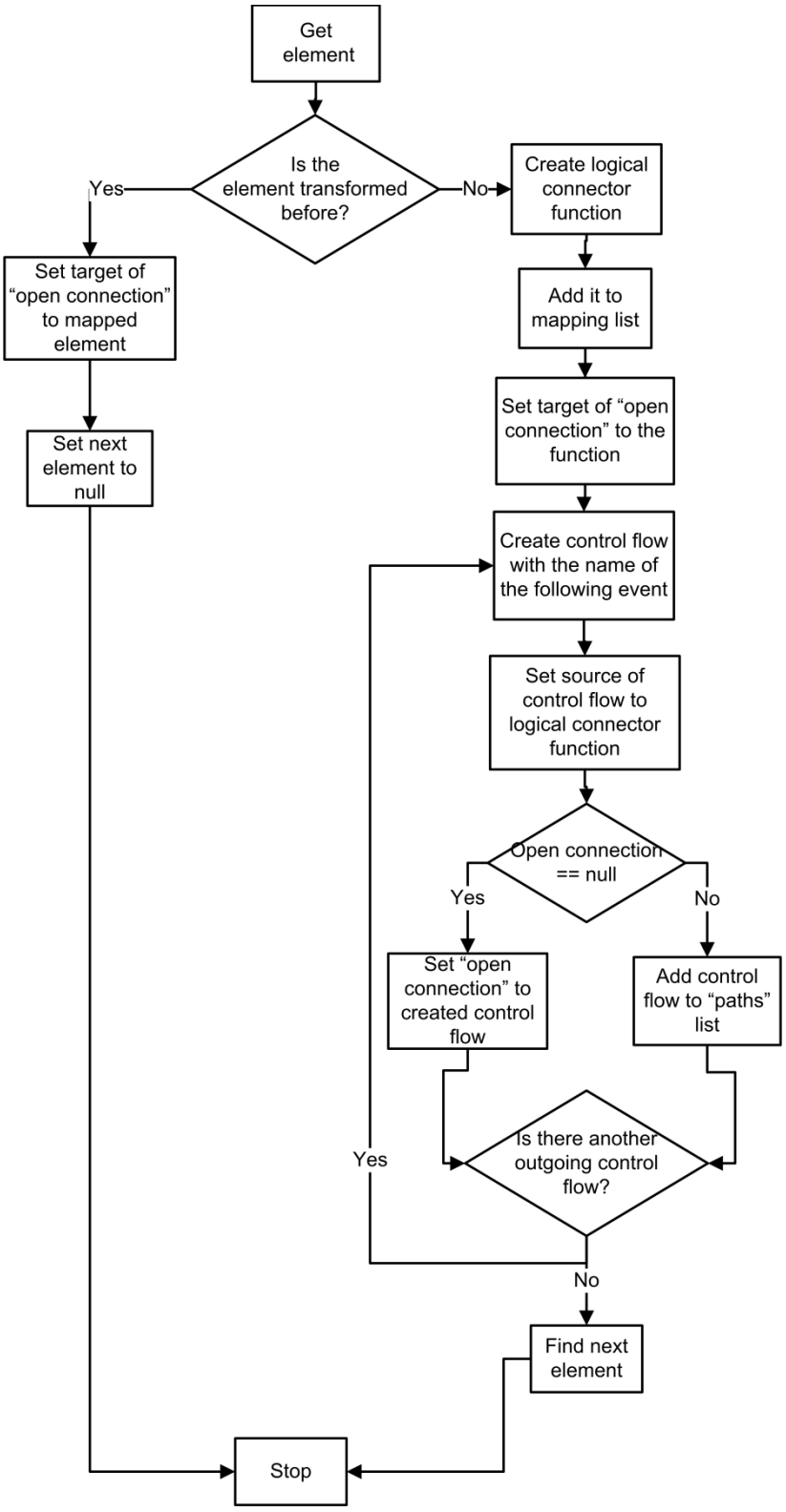

Figure 28. Logical Connector Transformation

While converting a valid eEPC model to S-BPM, the actor who performed the first function is thought as main subject. During the transformation of main subject's functions, function or function set which are connected to another actor are skipped.

In order not to disrupt the flow and the validity of reorganized diagram, the event of the last function which is performed by specified actor and the event of the last function which is performed by another actor are concatenated with an "and" connector. Figure 29 shows this technique in practice. However, during transformation only elements of separate models are saved to transform. Individual eEPC models are not generated.

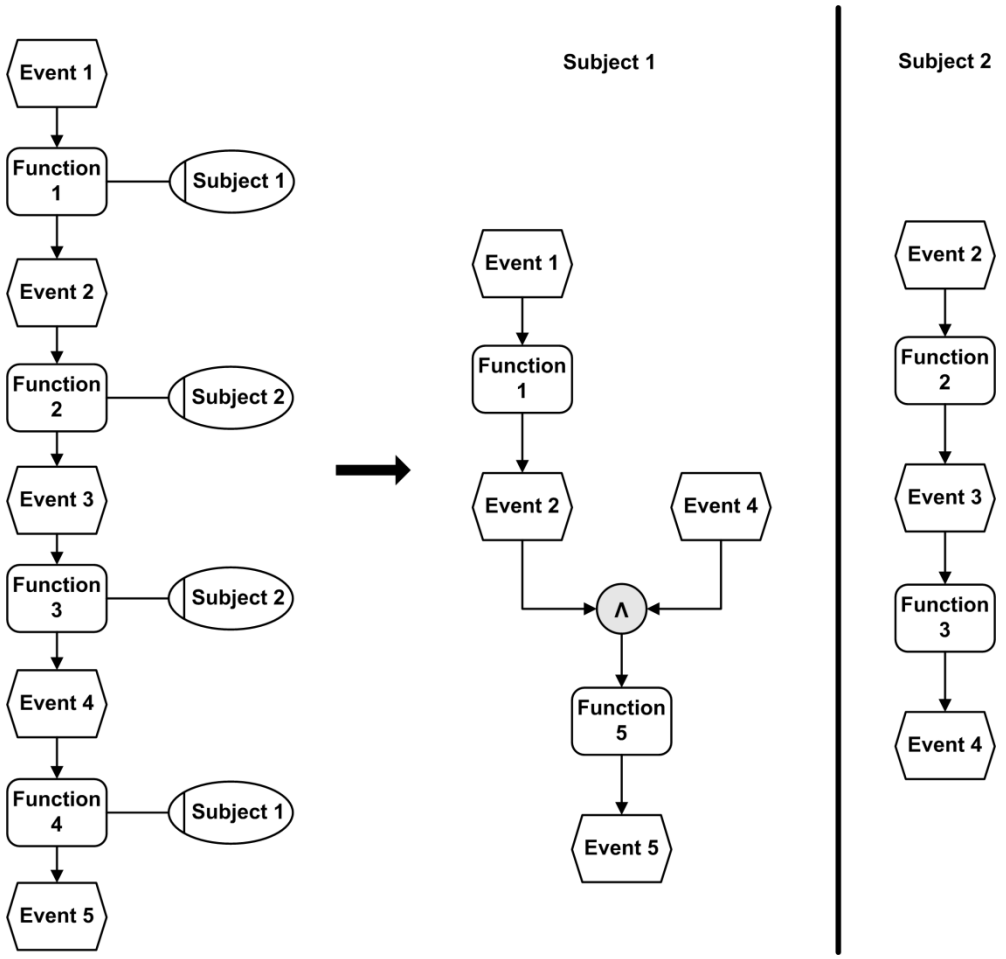

Figure 29. Generated sub-models from one eEPC diagram

Figure 30 shows which steps are performed when a function of a different actor is encountered. This procedure explains under what conditions, elements are added to "models" list and how the transformation continues. This procedure takes an eEPC element as an input. Then it checks that whether the element is added to a pre-defined sub-model or not. For this, element is searched in "models" list. If it exists, it will be transformed in next iterations and so the transformation is terminated. However; if it does not belong to any model, the subject of the element is taken into consideration. If the subject of the element is the same as the subject of previous function or process path, it is added to "models" list as an element of the last added model and the next function is fetched. If the subject is the same as the main subject (the first function's subject of eEPC model), a function state with «AND» annotation is generated to concatenate elements. The target of "open connection" is set to it. Then a start function with a state arc is generated. That state arc includes the name of

the previous element. The target of that arc is also set to the generated function state that includes «AND» annotation. Finally an outgoing state arc for that function state is generated and is set as "open connection". The next element is fetched and the procedure is terminated.

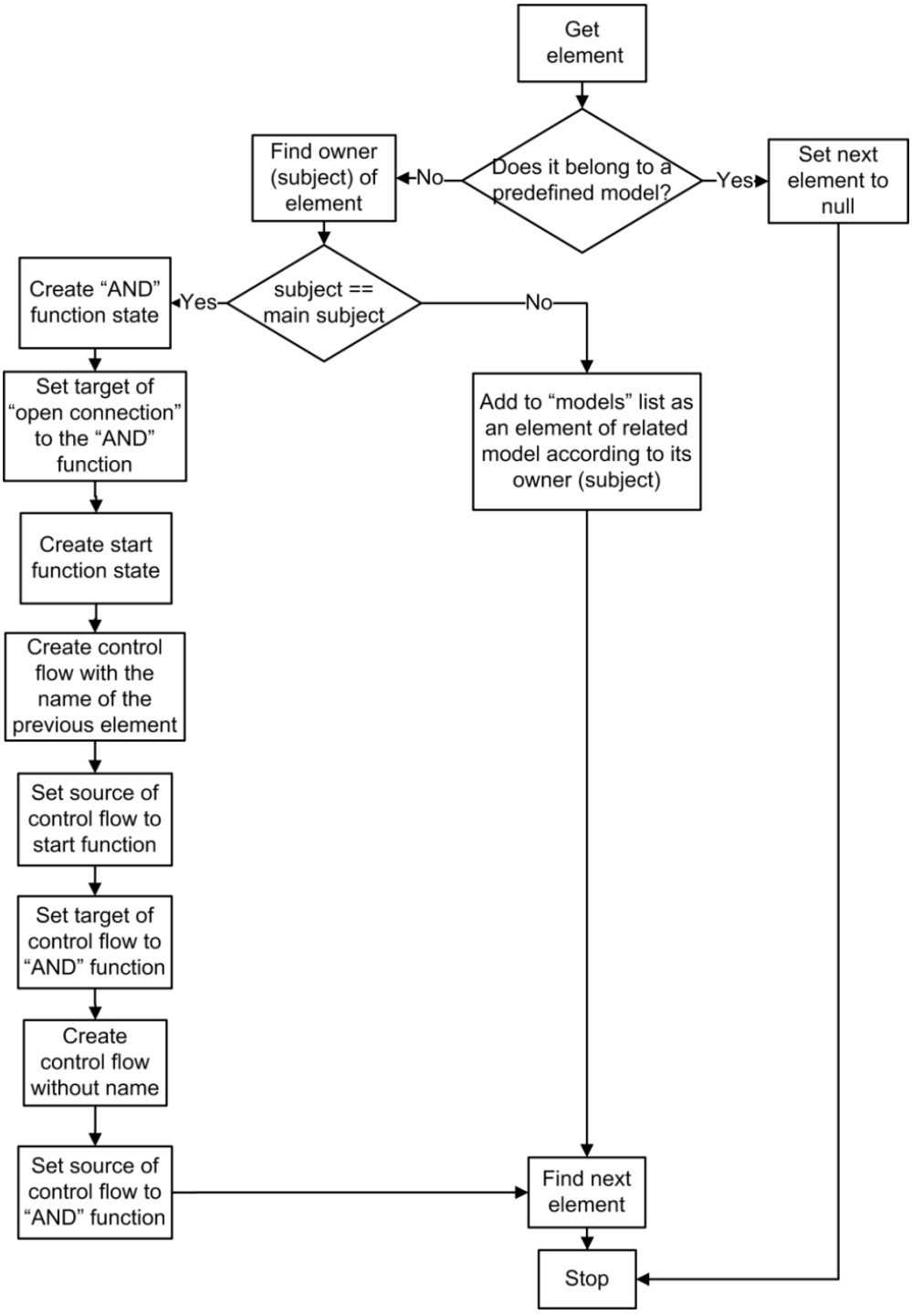

Figure 30. Determine models to be generated

CHAPTER 6

APPLICATION OF THE APPROACH

This chapter presents the application of the eEPC to S-BPM transformation on a case involving multiple business processes in a public institute. Section 6.1 describes the questions of the study, data collection and analysis strategies. In section 6.2 the conduct of the case is briefly described and automatic and manual transformations of selected processes are given. Section 6.3 discusses our findings and improvement possibilities.

6.1 Case Study Design and Questions

Case studies have been a common research strategy in many fields, such as psychology, sociology, political science and information systems [**36**], [**37**]. They can be conduct to show qualitative or quantitative evidences. In order to explore the applicability of eEPC to S-BPM transformation and to uncover improvement opportunities for the transformation approach, a case study is conducted. The case study involves business processes modeled for a public institute. Business process models that are generated by Bilgi Grubu in UPROM with eEPC notation are used as source models. Those processes belong to Ministry of Development of the Republic of Turkey. Bilgi Grubu analyzed the organizational structure and processes of the institute and conducted a project in order to define business processes and to support those processes with IT Systems. Within the scope of the project, main and supporting business processes which belong to development agencies are modeled [**38**].

The case study has the following primary research question:

- "What information is lost during transformation?"

  Automatically generated S-BPM models are analyzed. The source eEPC model and target S-BPM models are compared with each other and what information is lost during the transformation are identified. The answer of this question also reveals the required improvements and shows what type of information are not visualized in S-BPM models.

- "What are the semantic differences between the source and target models?

  In order to answer this question the semantics of the source eEPC model is analyzed. Then the target S-BPM model is checked whether it gives the same

semantic meaning or not. The results show the applicability of the transformation method.

- "What are the differences between automatically and manually generated models?"

  By comparing manually and automatically generated models, the differences are highlighted. The similarities between models show the power of automatic transformation. On the other hand differences help to analyze necessary improvements. The results show that how our automatic transformation reduces effort and spending time of manual transformation.

- "What improvements are needed for the transformation?"

  We were also interested in identifying improvement points and enhancing the transformation approach and tool. The answers of previous research questions reveal the problems of the approach. By using previous analyses we decide what improvements are necessary.

In the first phase of the case study we transform source models (eEPC) into target models (S-BPM) by UPROM automatically and analyze the results of automatic transformation. In the second phase, manually transformed models are compared with automatically transformed models in terms of mapping and semantics. Manual transformations are performed by Murat Salmanoğlu. He is the graduate student in METU Graduate School of Informatics and is working on business process modeling especially in S-BPM.

6.2 Case Study

Three essential processes of Archive Management System are selected for case study. These are archiving process, outgoing document tracking process and incoming document tracking process. While selecting processes, we emphasis on to cover all defined mapping rules as much as possible. In the following, eEPC models of those processes are given with a brief description. Then automatic and manual transformations are performed and the results are analyzed respectively. The validity of output SBDs are checked automatically by UPROM with pre-defined validation rules.

Archiving process

Archiving process is performed by two actors; personnel and archives officer. Personnel is responsible for determining which materials need to be achieved and creating archiving requests for them. Archives officer is responsible for filing the determined material and puts away the material to file defined in standard filing plan. Then he/she creates an inventory record for material and terminates the process. In eEPC diagram (Figure 31), it is shown that the process is started by personnel and completed by archives officer.

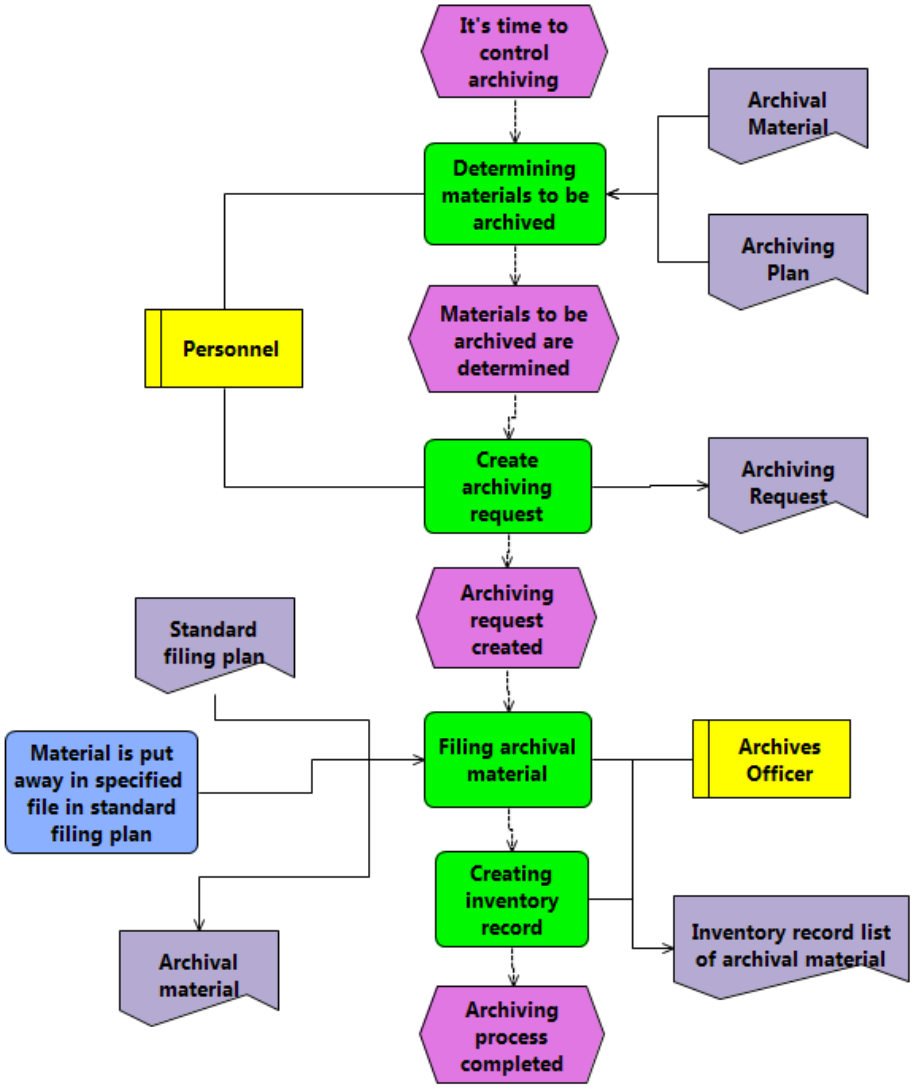

Figure 31. Archiving process in eEPC

Archiving process includes following elements/patterns to transform;

- Event (start, end and internal),
- Function with following event,
- Function without following event
- Multiple data with outgoing information flow
- Data with outgoing information flow
- Data with incoming information flow
- Resource object (Business Rule)

Because of the process is performed by two different actors, SBDs for each actor are generated during automatic transformation. Figure 32 shows SBD of "Personnel" (first actor) and Figure 33 shows SBD of "Archives Officer" (second actor) for archiving process.

All syntactic information given in the source model are also shown in automatically transformed models. All functions, events, incoming and outgoing data and business rules are also visualized in SBDs. However; semantics of the source model is not entirely same with the target models. In the source model input data which are used to perform related task are transformed into receive states. Thus, SBDs give the meaning that they are sent to related subject from another subject and after the subject receives data, the following function is triggered. However; in the source model there is not any information which shows whether the data is coming from another subject or not. In eEPC, data with outgoing information flow means that data is used as input. Output data also have the same problem. In eEPC data with incoming information flow is used to show output data which is produced during the execution of the function. It is not have to be sent to another subject. However, in SBD output data are transformed into send states and it is assumed that if there is a produced data, it has to be sent to another subject. Eventually, it is obvious that during the transformation of data objects, semantics of the model changes.

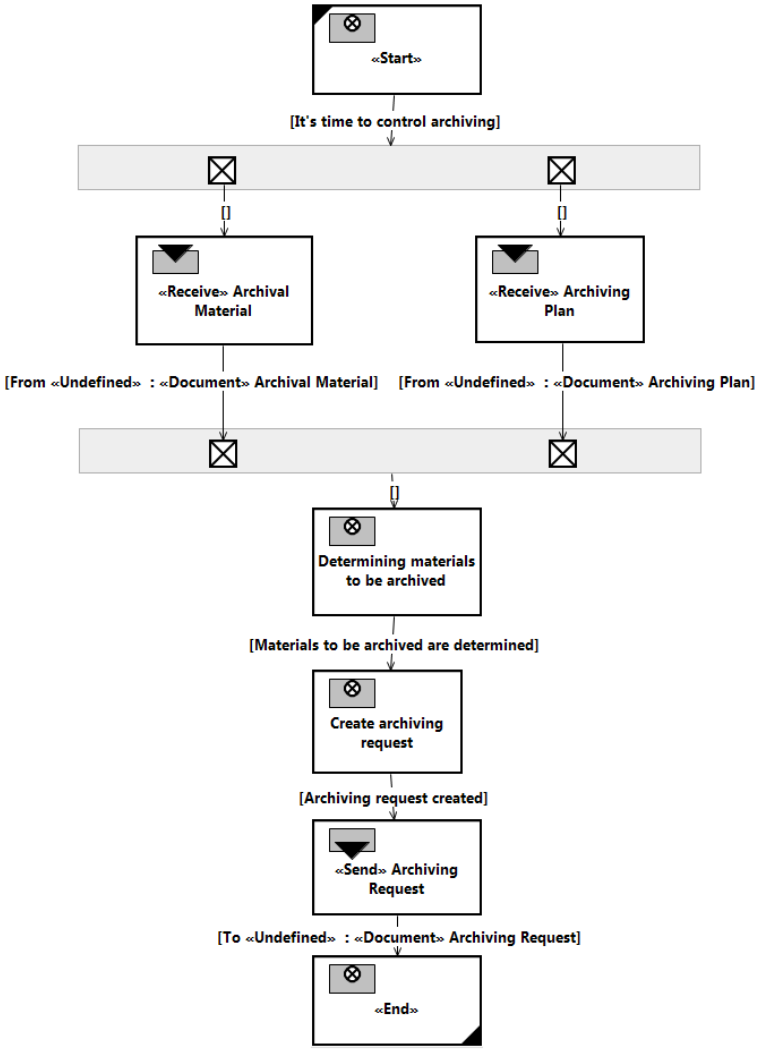

Figure 32. Automatically transformed archiving process for Personnel

In Figure 33, transformation of reference object (Business Rule) is shown newly as distinct from the previously generated model. While separating tasks of archives

officer, a dummy start function is added in order not to lose event which triggers the "Filing archival material" function. In this way, in generated model last event of previous model triggers the first function of achiever officer's process and the semantic meaning in the source model are protected.

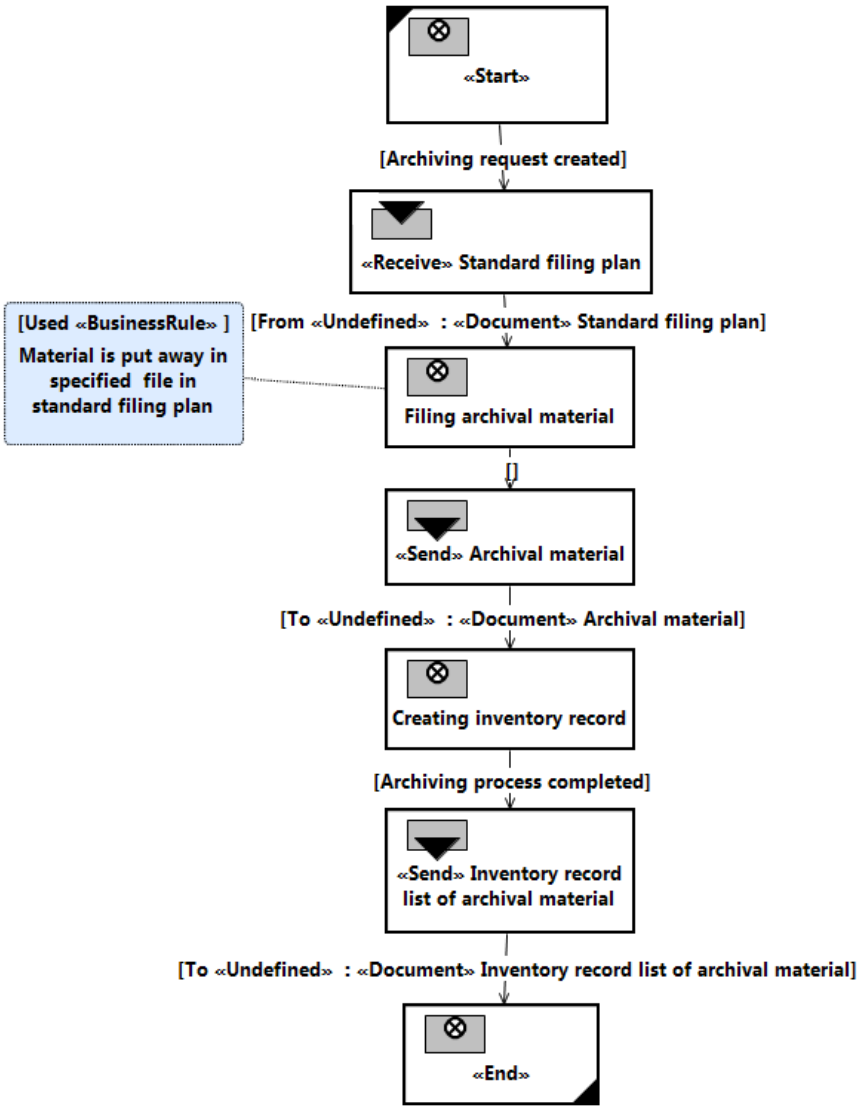

Figure 33. Automatically transformed archiving process for Archives Officer

Figure 34 shows manually generated SBD for "Personnel". When we compare the automatically and manually transformed models for "Personnel", following findings are identified.

- Applied mapping rules for internal event, end event, function with following event, data with outgoing information flow and data with incoming information flow are the same as automatic transformation.

- Instead of using dummy start state modeler prefers to mark the first subject state in target model as start state and the start event in the source model is omitted.

- Instead of using choice operator for multiple input data, modeler prefers to put receive states in order by taking initiative.

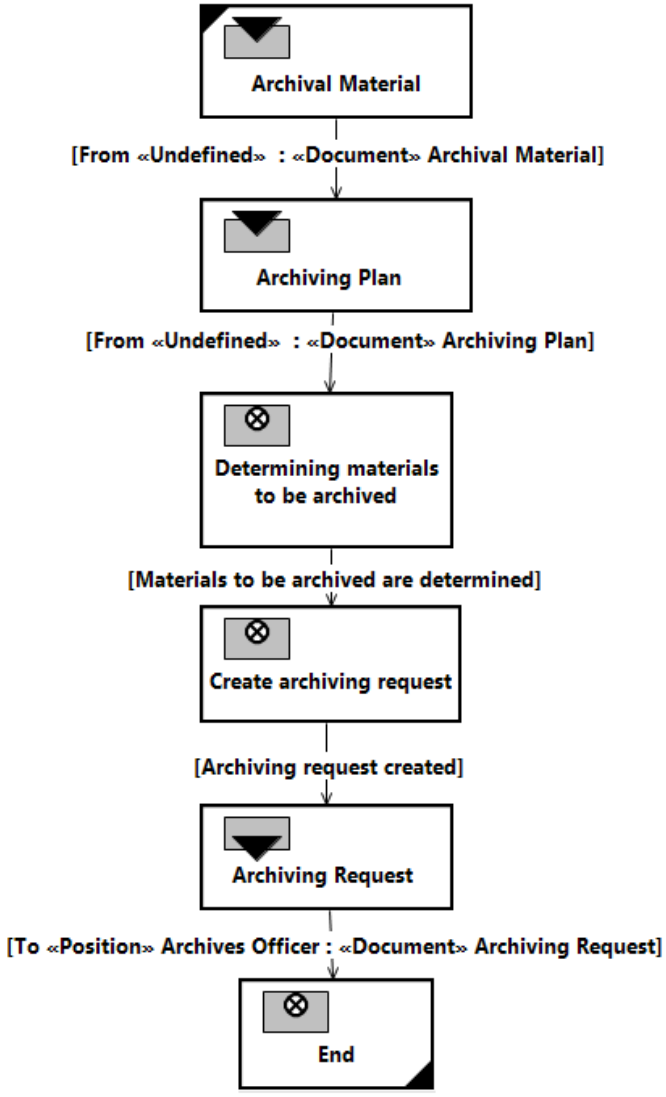

Figure 34. Manually transformed archiving process for Personnel

Figure 35 shows manually generated SBD for "Archives Officer". The comparison of the automatically and manually transformed models reveals the following differences.

- In contrast to automatic transformation, the last produced data by personnel are transformed into receive state and it is marked as start state. Receiving last producing data triggers the first function of achiever officer. In addition

to this; sender name and sender type is also given for the first receive state. Briefly; in automatic transformation last event triggers to following process, however in manual transformation last produced data triggers.

- Resource object (Business Rule) is not transformed in manual transformation. Since there is not any notation in S-BPM for resource objects. Thus the resource objects are omitted during manual transformation in contrast to automatic one.

- Functions without following event are transformed as function state and an outgoing arc without any text in automatic transformation. However; modeler adds post-condition onto outgoing state arc from scratch which is not in the source model. (Transformation of "Filing archival material" function)

- Following event of "Creating inventory record" function is omitted in manual transformation. The event states that archiving process is completed. This expression is omitted because archiving process is completed after the following send state is performed.

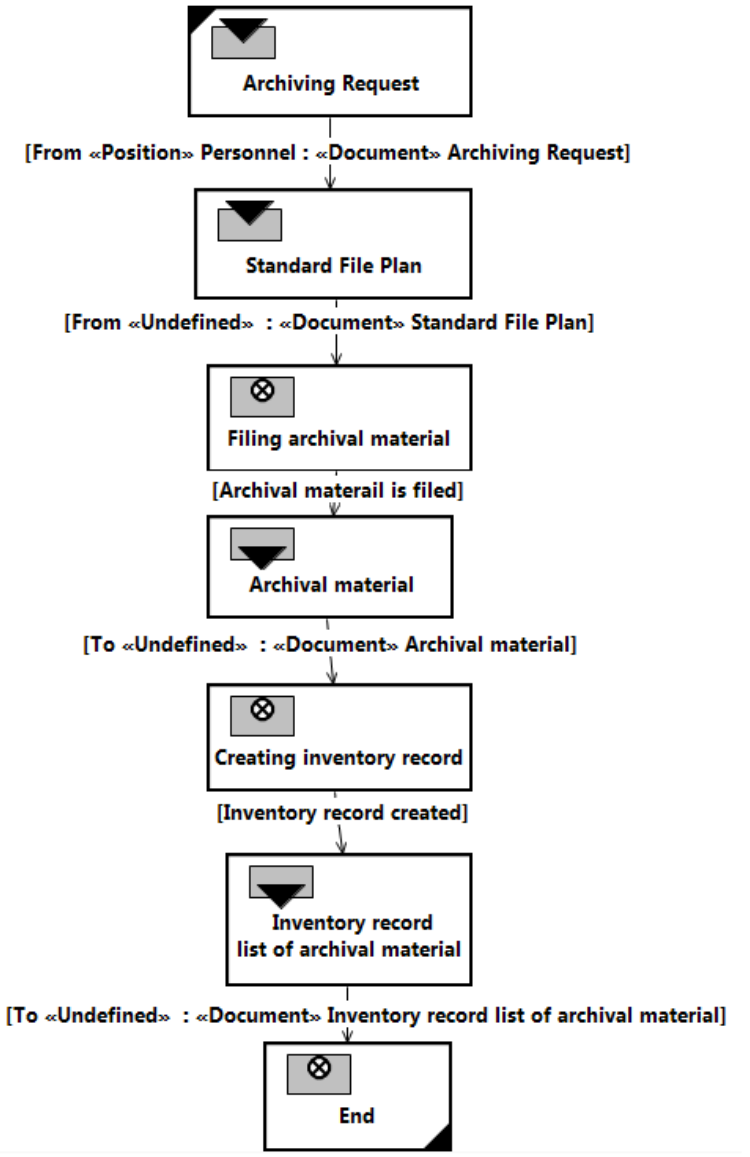

Figure 35. Manually transformed archiving process for Archives Officer

Outgoing document tracking process

Outgoing document tracking process is performed by personnel in the institute. When a document is requested from the archive, personnel creates the draft document that is copy of the original document with its appendices. Then he/she performs to document approval process to certify copied documents. After numbering the document, approved document is distributed to request owner. eEPC model of the process is given in Figure 36.

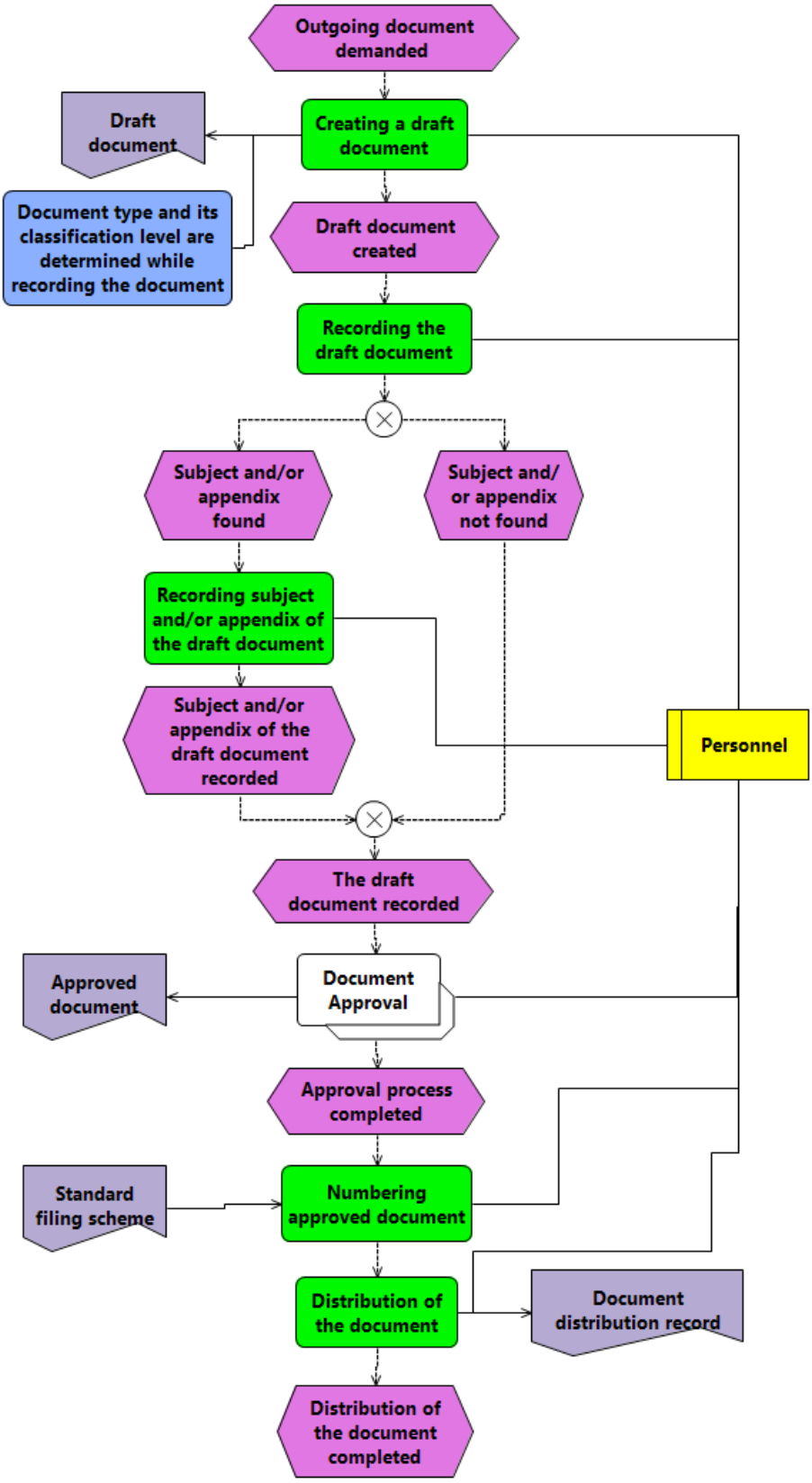

Figure 36. Outgoing document tracking process in eEPC

Outgoing document tracking process includes following elements/patterns to transform;

- Event (start, end and internal),
- Function with following event,
- Function without following event
- Data with outgoing information flow
- Data with incoming information flow
- Resource object (Business Rule)
- Logical connectors (XOR-Split and XOR-Join)
- Process path

All elements in the source model are transformed during automatic transformation (Figure 37). All functions, events, incoming and outgoing data, business rules, logical connectors and process path elements are also visualized in SBDs. Semantic problems mentioned in previous transformation are also available in this transformation. These are related to transformation of input and output data.

In addition to the previously transformed elements in this process, transformation of logical connectors (XOR-Split and XOR-Join) and process path element are seen differently. Logical connectors are transformed into function state elements with related annotation in automatic transformation. In this way, modelers and software developers can easily understand how the outgoing control flows should be activated in process flow.

In automatic transformation, process path element is transformed into macro class element. Macro classes consist of three main parts; start states, macro name and outputs. Macro name is the name of the process path element. However, start states and outputs are not given in the source eEPC model. Therefore; starts states and outputs are acquired from the eEPC model of "Document Approval" process. The model is given at Appendix B for verification. Start event "Draft document saved" is shown as start state of macro class and end event "The draft document transformed into approved document" is shown as output of the macro class. The output of macro class shows that approved document is produced at the end of "Document Approval" process. By using macro class for process path transformation, we also save the information; "Document Approval" is navigation to a sub-process which is modeled in a separate diagram.

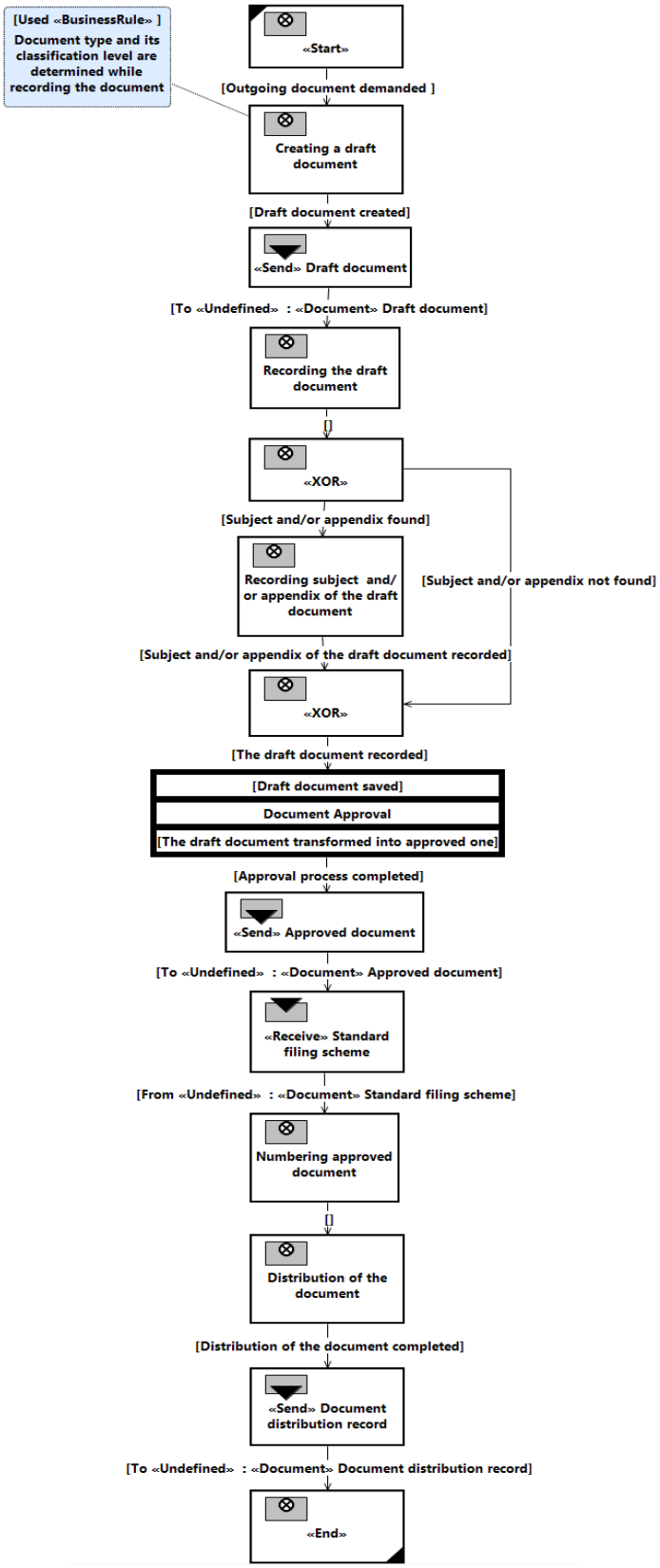

Figure 37. Automatically transformed outgoing document tracking process

Figure 38 shows manually generated SBD for outgoing document tracking process. When we compare the automatically and manually transformed models, following findings are identified.

- The start state in the source eEPC model is omitted during transformation.
- Business rule is not transformed due to the lack of notations.
- No additional function state is used to transform XOR-Split connectors in contrast to automatic transformation. Outgoing control flows of XOR split connector and following events of those flows are transformed as outgoing state arcs of previous function. “Recording the draft document” function state has two outgoing arcs and this gives the “OR” meaning. Therefore; in manual transformation the semantic meaning of “exclusive-OR” connector is not given.
- Similar problems are also available for XOR-Join connector. Incoming control flows of XOR join connector are directly connected to the next function. Thus the semantic meaning of the connector is lost. In addition to this, when the source of the incoming flow and the target of the outgoing control flow is an event, modeler has to omit one of them indispensably. In S-BPM, there is no notation for events thus there is also no way to show sequential events.
- For the transformation of process path element, function state is selected. In manually transformed model, “Document Approval” process is seen like a single task. Therefore, the eEPC diagram of “Document Approval” process is not used during transformation and the navigation meaning is discarded.
- For missing events in eEPC diagrams, new events are added to SBD by modeler.

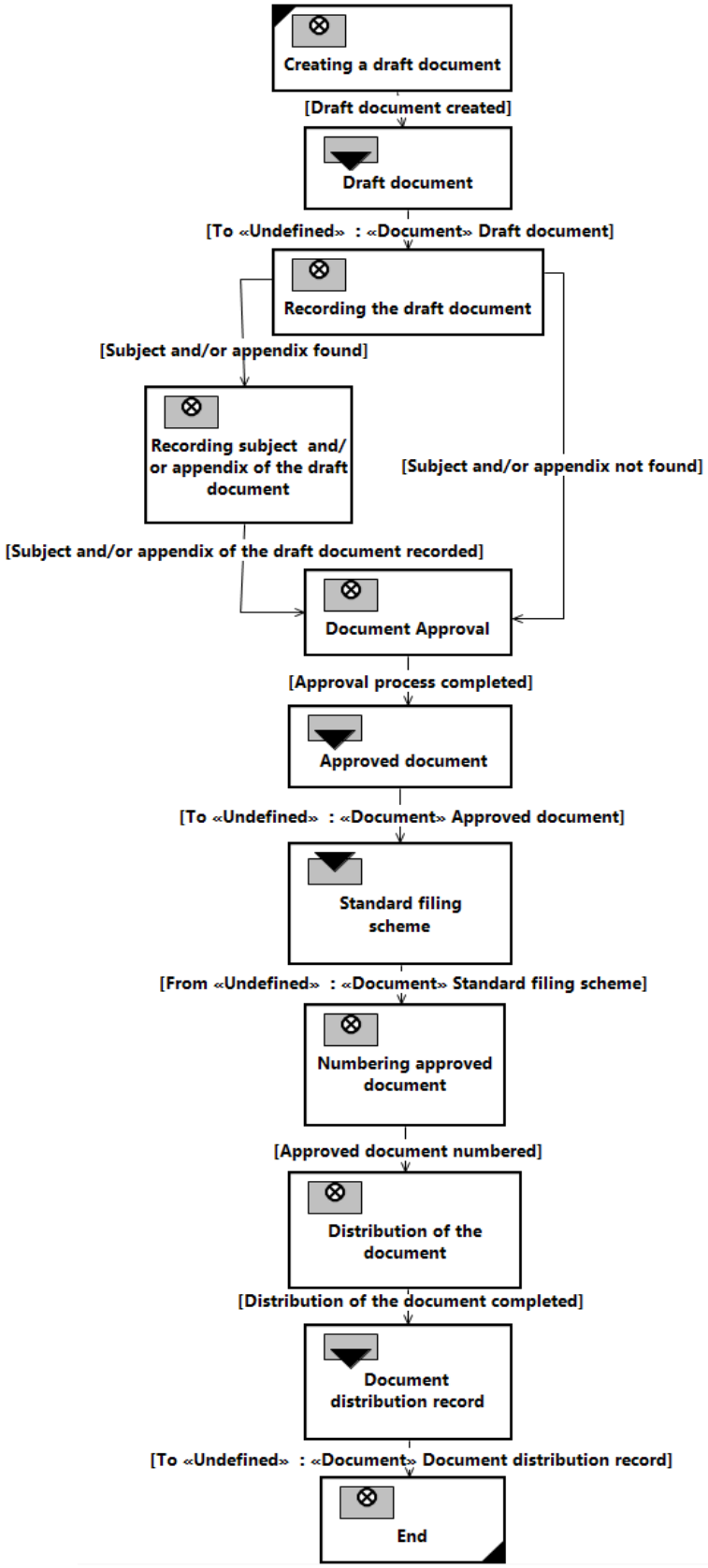

Figure 38. Manually transformed outgoing document tracking process

Incoming document tracking process

Documents which are sent to the institute from outside are prepared for archiving and distribution inside the institute. This preparation and distribution process is called as "Incoming document tracking" in the institute (Figure 39).

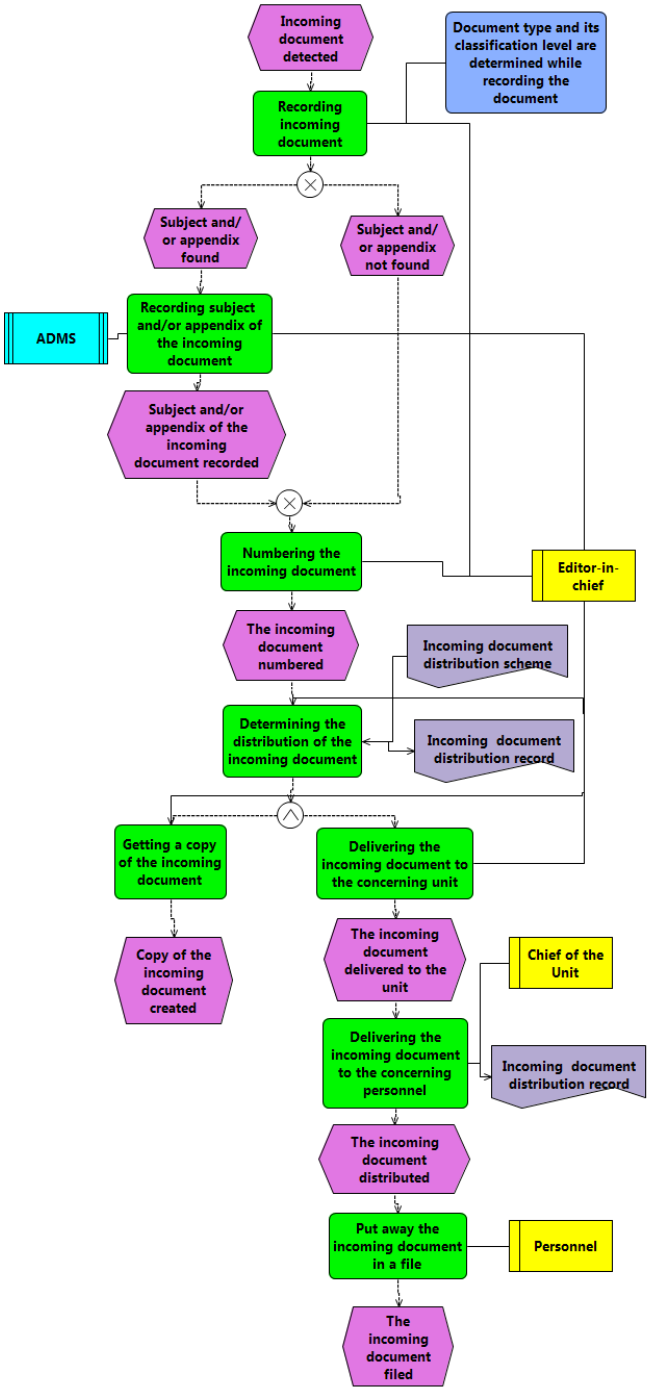

Figure 39. Incoming document tracking process in eEPC

Incoming document tracking process is accomplished by three different actors; Editor-in-chief, Chief of the unit and Personnel. Editor-in-chief is the main actor of

the process. He/she receives the incoming document and registers the document with its appendices by the help of ADMS (Achieve Document Management System). After numbering the document, distribution record of the document is prepared. Editor-in-chief gets a copy of the document for archiving and delivers to the document to the concerning unit. Then chief of the unit prepares another document distribution record and sends the document to personnel. Finally, personnel puts away the incoming document in a file and the process is completed.

Outgoing document tracking process includes following elements/patterns to transform;

- Event (start, end and internal),
- Function with following event,
- Data with outgoing information flow
- Data with incoming information flow
- Resource object (Business Rule and Application)
- Logical connectors (XOR-Split, XOR-Join and AND-Split)

This process is selected to show the transformation of Application (Resource object) and AND-Split connector. Because of the process is performed by three actors, SBDs for each actor are generated during automatic transformation. Figure 40 shows SBD of "Editor-in-chief" (first actor). In this automatic transformation, all elements are transformed into S-BPM. For instance; ADMS (Archive Document Management System) is transformed to a used item and connected to recording function by a relation. In this way, in generated model it is obviously seen, "Recording subject and/or appendix of the incoming document" function is used ADMS to perform its job. However; identified semantic problems in previous processes are also available in this transformation.

AND-Split connector is transformed into a function state element with related annotation. However; a single person typically cannot execute two tasks in parallel. Therefore, choice operator is most suitable to transform "AND" block. It provides subject freedom of choice. In other words; subject can perform paths of the "AND" block in any order. However; in order to use choice operator logical operator should be used in pairs. This usage provides to define the beginning and end of the block. In the source model, only one "And" connector is used to split control flow. When it is used alone, it is impossible to use choice operator for transformation due to the lack of information.

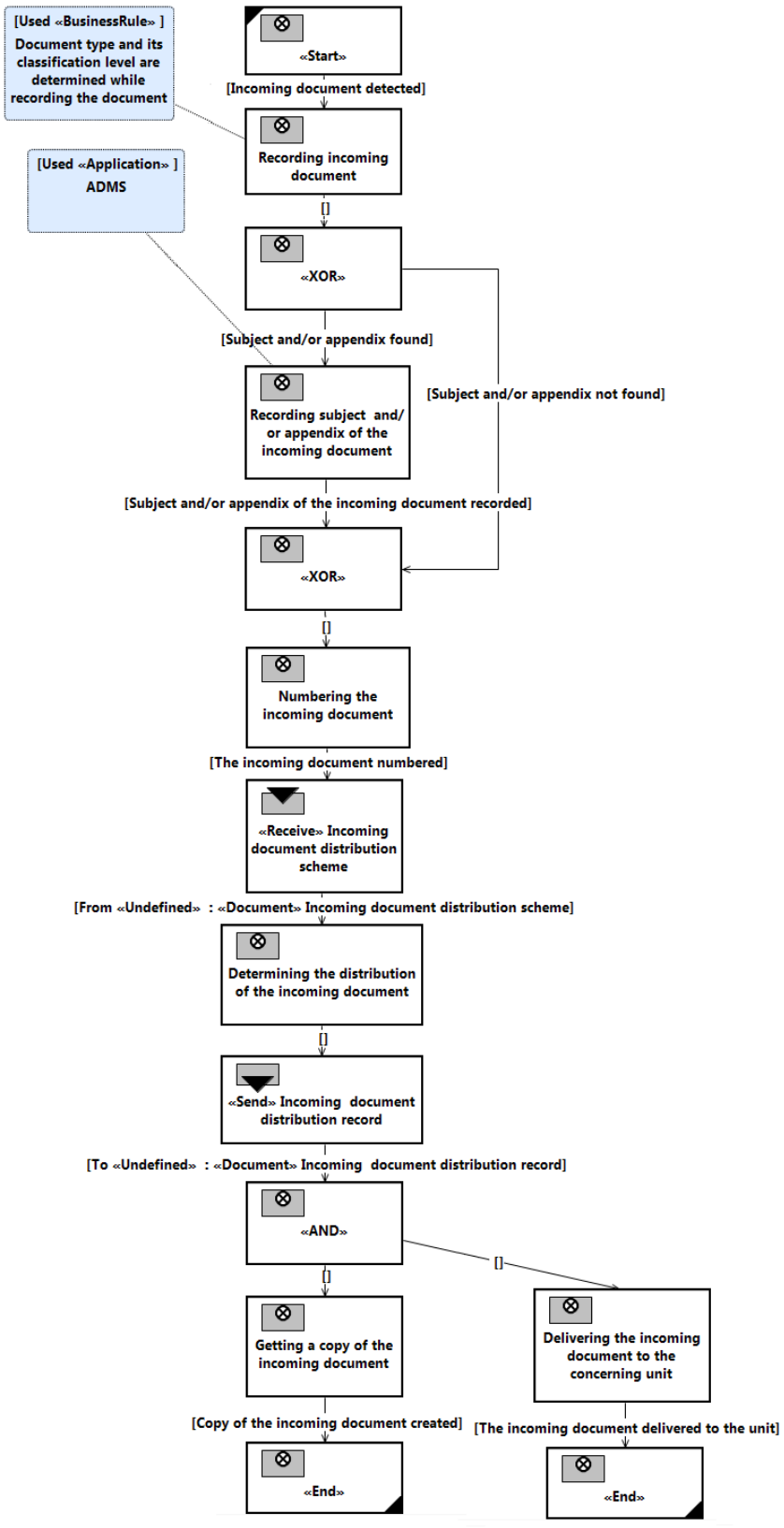

Figure 40. Automatically transformed incoming document tracking process for Editor-in-chief

Figure 41 shows manually generated SBD for editor-in-chief. Visual and semantic differences between automatically and manually transformed models are given in the following.

- In manual transformation, start state is omitted.

- Business rule and application are not transformed due to the lack of notations in S-BPM.

- XOR-Join and XOR-Split connectors are not directly map to any S-BPM element. The meaning of the connector is given by using multiple outgoing arcs. If a subject state has more outgoing arcs with different labels, which means that it splits the control flow into alternative paths. If a subject state has more incoming arcs with different labels, which means that the subject state is activated when one of the paths is completed. However; the type of the connector is not given explicitly. Therefore; manually transformed models can be interpreted differently by different modelers and the implementation of logical connectors are also depends on interpretation of modelers.

- AND-Split connectors are not also directly mapped to any S-BPM element. Multiple outgoing arcs with the same information (label) are used for AND connectors in manual transformation. If a subject state has more outgoing arcs with the same label, which means that, it activates all outgoing control flows. In the transformation of AND-Join connectors incoming arcs have different information because they are transformed independently from the following AND operators, they shows the information about previous subject states. Therefore the meaning of AND-Join operator is not given in the target model clearly like other join operators.

- For missing events in eEPC diagrams, new events are added to S-BPM diagram by modeler.

- Another difference between automatic and manual transformation is that receiver information is added to some of the receive arcs. Modeler interprets the whole model and determines the message flow between subjects. In this way receivers of some of the documents are specified in the model.

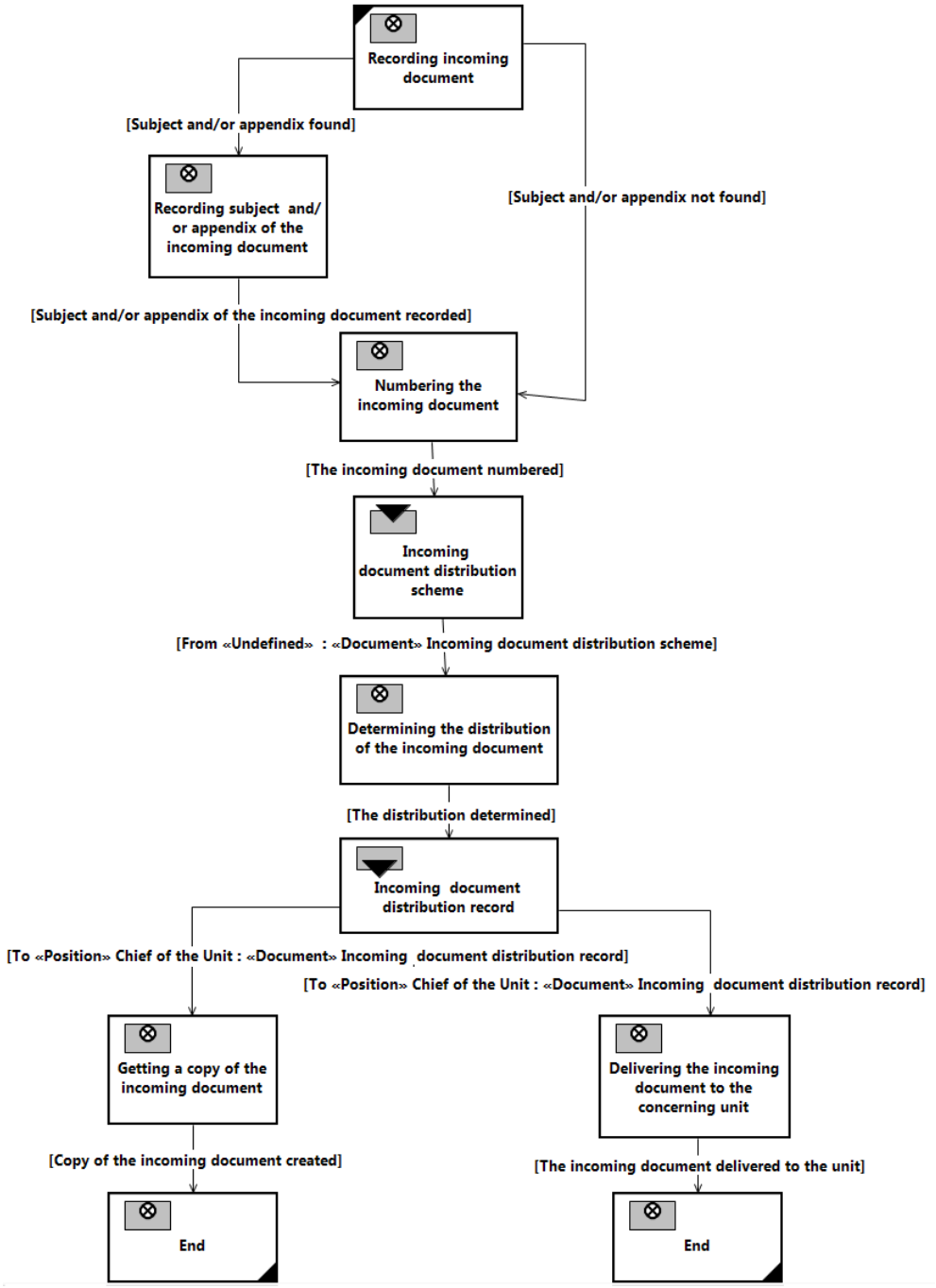

Figure 41. Manually transformed incoming document tracking process for Editor-in-chief

Figure 42 shows the automatic and manual transformation for chief of the unit's responsibilities in incoming document tracking process. The only difference between models is start states. In automatic transformation a dummy start state is used and last event of the previous subject is used to trigger the first subject state of chief of the unit's process. In other words, the last event of previous subject (Editor-in-chief) starts the chief of the unit's process. However; in manual transformation receiving data from previous subject starts the process. When chief of the unit receives

incoming document from editor-in-chief, he/she starts to perform his/her responsibilities. Determination of the start state (receiving data) is not done with a predetermined rule. In archiving process last produced data of previous subject is used to trigger the following process. However; in this case incoming document is not produced by editor-in-chief, it is an input of editor-in-chief's process. Those outcomes show that modeler determines the initial subject state by using initiative. In addition to this, last produced element is sent to next subject. However; in source model this information is not given directly.

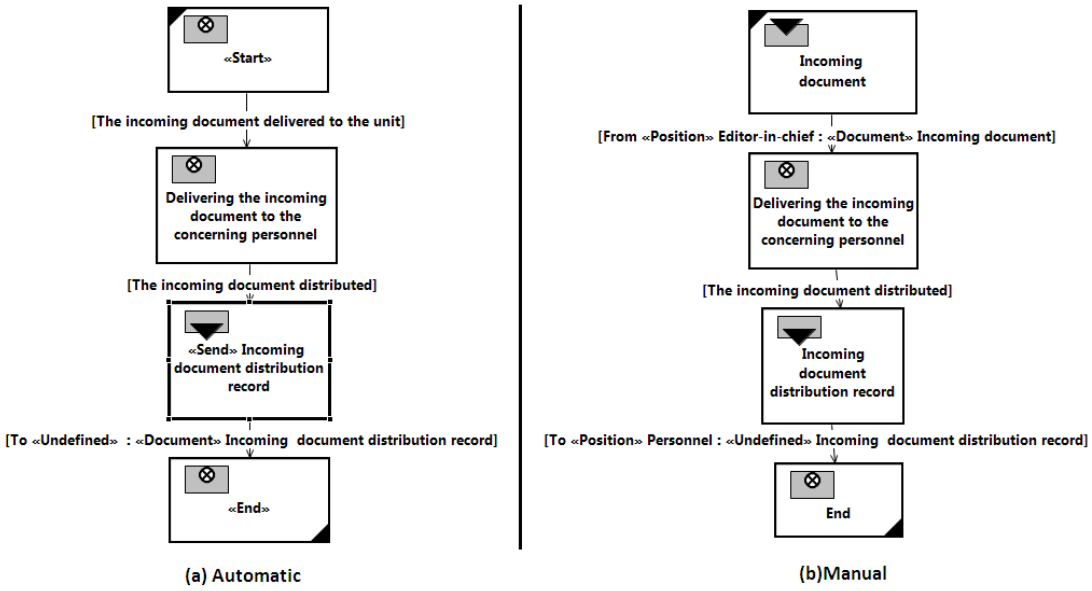

Figure 42. Transformed incoming document tracking process for Chief of the unit

In Figure 43, automatic and manual transformations of incoming document tracking process for Personnel are shown. The same differences with chief of the unit's process are also available in this transformation.

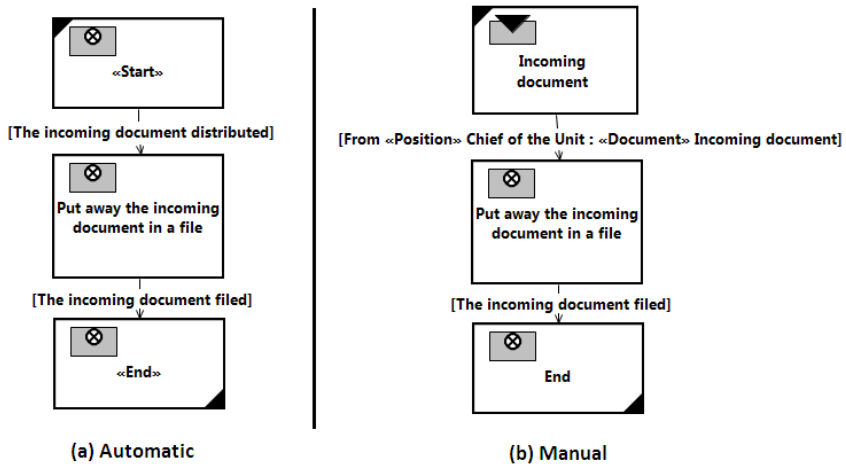

Figure 43. Transformed incoming document tracking process for Personnel

6.3 Results and Discussions

In automatic transformation, source eEPC model is transformed into SBDs without any syntactic information lost by using the pre-defined transformation rules. The generated SBDs are valid and semantics of the input model are majorly preserved in output models. As a result of eEPC to S-BPM transformation, individual models for each subject are generated. The output model names give information about subjects and related processes. However, in this study interactions of the subjects and message flow between them are not taken into consideration. In other words, SID for source model is not generated in automatic transformation. Therefore; generated models do not show which subject starts the process, which subject completes the process, what information are transmitted between subjects and the order of subjects in the process workflow. In manual transformation, modeler traces the message flow between subjects and reflects this information into SBDs. Message flow are especially used while separating source model to SBDs. Receiving message from previous actor starts the processes and processes generally ends with sending message to the next actor. In order to provide traceability, the automatic transformation approach should be improved and SID for input process should also be generated before the SBD generation.

Additionally, data objects are transformed into send or receive states according to direction of the information flow in automatic transformation. However; this transformation changes the meaning of target model in some cases. In eEPC diagrams, data with outgoing information flow is used to show that data is an input of related function and it is used during the execution of related function. When it is transformed into receive state element, the meaning changes to that it is sent by another subject to the owner of related function. Similar to this, data with incoming information flow is used to show produced data in eEPC. Transforming output data into send state element changes the semantics to "data is produced and sent to other subjects". However; since there is no element to map data objects in S-BPM, not only automatic transformation but also manual transformation send and receive states are used for this purpose.

Transformation of multiple input/output data differs in automatic and manual transformation. In automatic transformation choice operator with closed start and end switches (which gives the "and" meaning) is used. Choice operator gives the meaning that data are received from other subjects in any order. However; in manual transformation modeler put receive states in an order. Putting them in an order changes the semantics of the model since it puts constraints. This is the decision of modeler but it is not appropriate to order receiving and sending messages in automatic transformation without any basis.

Duplication is another problem which occurs in the conversion of sending and receiving messages. If there is a function which includes "receive" keyword in its description and an incoming information flow connected to it, information flow part is converted as receive message with «receive» annotation and conversion of the function also includes "receive" keyword which refers to the same data. The duplication problem also occurs in the conversion of outgoing information flow connected to a function that contains "Send" keyword. In order to avoid this

problem, description of functions is also taken into account. Information flow and related function is considered as a different pattern and that pattern is mapped to “Receive Message” or “Send Message” element directly. However, there is no way to show all information flows in a single S-BPM element (e.g. send-receive element) if there are more than one information flow related to the function. Thus, possible duplicates are not handled in order not to complicate transformation.

The most problematic part the transformation is logical connectors. In the first glance, we have evaluated to use choice operator for branches of the logical connector. However, in this approach the beginning and the end of the control flows have to be known. Additionally, combinations of connectors are not mapped to combinations of alternative clauses because of S-BPM syntax. Therefore, instead of this notation, we preferred to define new functions with “«and»”, “«or»” and “«xor»” annotations. Logical connectors are transformed into function states the semantics of them are preserved by annotations. Transforming “AND” operator by this way contradicts the S-BPM concept that the work of a subject is executed by a single person and a single person typically cannot execute two tasks in parallel. However; if logical connectors are not used in pairs in the source model, there is no way to determine the end of those processes. Thus; in manually generated models choice operator is not used, too. In order to solve this problem validation rules for eEPC can be extended and modeling tool can be improved to prevent modelers to use single “And” operator, it only allows operators used in pairs.

Despite all these analysis, automatically and manually generated models are very similar. For direct mappings same rules are applied, but if direct mapping is not possible different approaches are used in transformations. Separations of models according to subjects are performed in the same way and so the output models are similar in two of them. Eventually, automatic transformation is guaranteed that transformation is performed without losing any syntactic information. However; there are small semantic differences which are also available manual transformation. Changes in the semantics can be handled by small modifications in the target models.

During the case study, we have had some observations for more understandable model generations. These remarks can be summarized as follows:

- Each function should be followed by an event and each function should be triggered by an event in the input model. Otherwise, null transitions will be occurred in the model.
- Each data object should be related to a subject. Otherwise, subject information will be marked as «undefined».
- In case of eEPC model belongs to only one actor, less changes in semantics occurs in transformation.
- “AND” connectors should be used in pairs.

6.4 Threats to Validity

In this study, core eEPC elements and a subset of elements in data view and organization view are covered for transformation. Core eEPC elements are events, functions, process path and logical connectors (and, or, exclusive-or) and they

construct the process workflow. In organization view; organizational unit, group and position are taken into consideration. In data view; document, list, log, product, file, application, reference and business rule elements are transformed.

In addition to determining elements' subset, validation rules are also defined for source models. Our approach supports only models which are valid according to those validation rules. In order to apply our transformation algorithm to an eEPC model, following rules should be satisfied by the model.

- There must be at least one start event
- There must be at least one end event
- All elements must be connected
- All functions or process paths must have exactly one incoming and one outgoing control flow
- Events cannot be consecutive to each other
- Split connectors must have one incoming control flow and more than one outgoing control flow
- Join connectors must have more than one incoming control flow and one outgoing control flow
- Except start and end events, logical connectors should be used in pairs

This study supports only modeling and generation of SBDs. SID are not supported by the editor and they are not generated during transformation. As in eEPC, in S-BPM also a subset of elements is used. Elements of SIDs (subject, business objects and message flow) and some of the SBD elements such as multiprocesses, exceptions and extensions are not supported.

Validation rules for S-BPM models are also defined in the scope of this study. Those rules are implemented in S-BPM editor and guide modelers to construct valid SBDs. They are also used to check correctness of automatically generated models.

- There must be at least one start event
- There must be at least one end event
- Subject states must not be start and end state at the same time
- All elements must be connected
- Except start and end events, states must have at least one outgoing and one incoming control flow
- Alternative bars must have at least two switches
- The source of states arc must be function state, macro class, alternative bar or switch.
- The source of receive arc can be receive state
- The source of send arc can be send state
- The source of relation must be Used Items
- The target of relation should be Function States or Macro Classes

Assumptions in transformation;

- Consecutive functions or process paths accomplished by one actor are transformed separately as a new SBD. Since, if functions accomplished by a specific actor are not consecutive, determining the order of those functions is not possible.
- If a subject (organization unit, group, position) is connected to a function or process path, it shows the performer of that function or process path.
- If a subject is connected to a data object, subjects are considered as sender or receiver of that data according to the direction of information flow.
- Data objects with outgoing control flow are considered as receiving messages.
- Data objects with incoming control flow are considered as sending messages.
- All receiving messages take place before the concerning function or process path and all sending messages come after the function or process path in the flow.
- «undefined» annotation is used for all missing information during transformation.

CHAPTER 7

CONCLUSIONS & FUTURE WORK

This thesis presents a contribution about eEPC to S-BPM transformation with a concrete and explicit transformation method. The main motivation is to provide a guideline to generate SBDs from eEPC models. For this purpose; mapping rules are defined and transformation algorithm for realization are described. Defined mapping rules are simplified manual transformation process for modelers. Furthermore, realization of the transformation approach provides modelers to adapt previously modeled business processes to S-BPM paradigm in a short time with minimum effort.

In the scope of this study firstly S-BPM editor as an UPROM plug-in is developed, it visualizes core SBD elements. This editor is also more comprehensive than commercial editors in terms of notations. It also satisfies continues verification during modeling time. Secondly transformation engine is developed as a plug-in in UPROM. The details of transformation algorithm are given in Section 5.3 and realization of the algorithm proves the applicability of the concept.

Analyses depend on case study results show that transformation is performed without losing any information. Semantics of the input model and output models are analyzed manually by different modelers and it is observed that semantics are significantly preserved in output models. However; minor differences in semantics occur during transformation. Since; there is no chance to transform all element combinations with saving its semantics. There are also some differences between automatically and manually generated models. Those differences generally arise from different interpretations. Thus; variance can be also happen in manually transformed models by different modelers as eEPC is not a formal modeling notation. Nevertheless; the results show that our transformation approach needs to be improved in terms of subjects' interactions. In order to visualize interactions and message flow between subjects, SID for input process should also be generated before the SBD generation in automatic transformation. Additionally, restrictions for source business models should be determined and in some cases user interaction is required in order to save semantic meaning.

In the future; S-BPM editor can be extended in order to provide modeling of SIDs. Restrictions and new validation rules can be added to eEPC editor in order to generate S-BPM models with better semantic mapping. In order to avoid misinterpretations of source models user interaction during transformation time can

be added as a new feature. In other words semi-automatic transformation can be developed. Automatic code and requirement generation for S-BPM models can be added to newly developed S-BPM editor.

REFERENCES

[1] Mathias Weske, *Business process management: concepts, languages, architectures*.: Springer, 2012.

[2] Ruth Sara Aguilar-Saven, "Business process modelling: Review and framework," *International Journal of production economics*, pp. 129-149, 2004.

[3] August Wilhelm Scheer, *ARIS- Modeling Methods, Meta-models, Applications*. Berlin: Springer, 1998.

[4] August Wilhelm Scheer, *ARIS - Business Process Modeling*.: Springer, 1999.

[5] Albert Fleischmann, Werner Schmidt, Christian Stary, Stefan Obermeier, and Egon Börger, *Subject-Oriented Business Process Management*.: Springer, 2012.

[6] Albert Fleischmann, "What Is S-BPM?," in *S-BPM ONE–Setting the Stage for Subject-Oriented Business Process Management*.: Springer, 2010, pp. 85-106.

[7] Robert Singer and Erwin Zinser, "Business Process Management—S-BPM a New Paradigm for Competitive Advantage?," in *S-BPM ONE–Setting the Stage for Subject-Oriented Business Process Management*.: Springer, 2010, pp. 48-70.

[8] Stefan Reinheimer, "Modeling Needs in the BPM Consulting Process," in *S-BPM ONE-Learning by Doing-Doing by Learning*. Berlin , Heidelberg: Springer, 2011, pp. 115-125.

[9] Shane Sendall and Wojtek Kozaczynski, "Model transformation: The heart and soul of model-driven software development," *Software, IEEE*, pp. 42-45, 2003.

[10] Barbara Handy, Max Dirndorfer, Josef Schneeberger, and Herbert Fischer, "Methods of Process Modeling in the Context of Civil Services by the Example of German Notaries," in *S-BPM ONE-Learning by Doing-Doing by Learning*.: Springer, 2011, pp. 281-295.

[11] Albert Fleischmann, Werner Schmidt, and Christian Stary, "Open S-BPM= Open Innovation," in *S-BPM ONE-Running Processes*.: Springer, 2013, pp. 295-320.

[12] Jörg Rodenhagen and Florian Strecker, "Using Multi-subjects for Process Synchronization on Different Abstraction Levels," in *Subject-Oriented Business Process Management*.: Springer, 2011, pp. 134-162.

[13] Remco M Dijkman, Marlon Dumas, and Chun Ouyang, "Semantics and analysis of business process models in BPMN," *Information and Software Technology*, pp. 1281-1294, 2008.

[14] Ivo Raedts et al., "Transformation of BPMN Models for Behaviour Analysis," , 2007, pp. 126-137.

[15] Nguyễn Quốc Bảo, "A proposal for a method to translate BPMN model into UML activity diagram," in *13th International Conference on Business Information Systems*, 2010.

[16] María Agustina Cibran, "Translating BPMN Models into UML Activities," , 2009, pp. 236-247.

[17] Volker Hoyer, Eva Bucherer, and Florian Schnabel, "Collaborative e-Business Process Modelling: Transforming Private EPC to Public BPMN Business Process Models," in *Business Process Management Workshops*, 2008, pp. 185-196.

[18] Willi Tscheschner, "Transformation from EPC to BPMN," *Business Process Technology*, pp. 7-21, 2006.

[19] Olga Levina, "Assessing Information Loss in EPC to BPMN Business Process Model Transformation," in *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2012 IEEE 16th International*, 2012, pp. 51-55.

[20] Birgit Korherr and Beate List, "A UML 2 Profile for Event Driven Process Chains," in *Research and Practical Issues of Enterprise Information Systems*.: Springer, 2006, pp. 161-172.

[21] Markus Nüttgens, Thomas Feld, and Volker Zimmermann, "Business Process Modeling with EPC and UML: transformation or integration?," in *The Unified Modeling Language*.: Springer, 1998, pp. 250-261.

[22] Peter Loos and Thomas Allweyer, "Object-orientation in business process modeling through applying event driven process chains (EPC) in UML," in *Enterprise Distributed Object Computing Workshop, 1998. EDOC'98. Proceedings. Second International*, 1998, pp. 102-112.

[23] Kees Van Hee, Olivia Oanea, and Natalia Sidorova, "Colored Petri nets to verify extended event-driven process chains," in *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*.: Springer, 2005, pp. 183-201.

[24] Niels Lohmann, Eric Verbeek, and Remco Dijkman, "Petri net transformations for business processes–a survey," in *Transactions on Petri Nets and Other Models of Concurrency II*.: Springer, 2009, pp. 46-63.

[25] Stephan Sneed, "Mapping Possibilities of S-BPM and BPMN 2.0," in *S-BPM ONE-*

Education and Industrial Developments.: Springer, 2012, pp. 91-105.

[26] Kenneth C Laudon and Jane Laudon, "Management information systems: managing the digital firm," *New Jersey*, 2004.

[27] R G Lee and B G Dale, "Business process management: a review and evaluation," *Business process management journal*, pp. 214-225, 1998.

[28] Marta Indulska, Jan Recker, Michael Rosemann, and Peter Green, "Business process modeling: Current issues and future challenges," *Advanced information systems engineering*, pp. 501-514, 2009.

[29] Marlon Dumas, Wil M Van der Aalst, and Arthur H Ter Hofstede, *Process-aware information systems: bridging people and software through process technology*.: Wiley-Interscience, 2005.

[30] Wil MP van der Aalst, "Formalization and verification of event-driven process chains," *Information and Software technology*, pp. 639-650, 1999.

[31] Albert Fleischmann, Werner Schmidt, Christian Stary, and Florian Strecker, "Nondeterministic events in business processes," in *Business Process Management Workshops*, Berlin, 2013, pp. 364-377.

[32] Albert Fleischmann, Werner Schmidt, and Christian Stary, "A Primer to Subject-Oriented Business Process Modeling," in *S-BPM ONE–Scientific Research*.: Springer, 2012, pp. 218-240.

[33] Christian Böhme et al., "bflow* Toolbox-an Open-Source Modeling Tool," 2011.

[34] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro, *EMF: eclipse modeling framework*.: Pearson Education , 2008.

[35] Sven Efftinge and Markus Völter, "oAW xText: A framework for textual DSLs," in *Workshop on Modeling Symposium at Eclipse Summit*, 2006.

[36] Robert K Yin, "The Case Study Method: An Annotated Bibliography," *COSMOS Corporation*, 1983.

[37] Robert K Yin, *Applications of case study research*.: Sage, 2011.

[38] Ahmet Coşkunçay, Banu Aysolmaz, Onur Demirörs, Ömer Bilen, and İdris Doğan, "An Approach for Concurrent Business Process Modeling and Requirements Analysis," in *Symposium on Software Quality and Software Development Tools*, İstanbul, 2010.

APPENDICES

APPENDIX A: SBD Validation Rules in Check Language

sbpmSyntax.chk context

```
context sbpm::FunctionState if(shallCheck("SBPMSyntaxRule1"))
        ERROR getErrorMessage("SBPMSyntaxRule1",this.name):
        !(this.isStart && this.isEnd);

context sbpm::ReceiveState if(shallCheck("SBPMSyntaxRule2"))
        ERROR getErrorMessage("SBPMSyntaxRule2",this.name):
        !(this.isStart && this.isEnd);

context sbpm::SendState if(shallCheck("SBPMSyntaxRule3"))
        ERROR getErrorMessage("SBPMSyntaxRule3",this.name):
        !(this.isStart && this.isEnd);

context sbpm::FunctionState if(shallCheck("SBPMSyntaxRule4"))
        ERROR getErrorMessage("SBPMSyntaxRule4",this.name):
        !(this.in.toList().size==0 && !this.isStart);

context sbpm::ReceiveState if(shallCheck("SBPMSyntaxRule5"))
        ERROR getErrorMessage("SBPMSyntaxRule5",this.name):
        !(this.in.toList().size==0 && !this.isStart);

context sbpm::SendState if(shallCheck("SBPMSyntaxRule6"))
        ERROR getErrorMessage("SBPMSyntaxRule6",this.name):
        !(this.in.toList().size==0 && !this.isStart);

context sbpm::SubjectState if(shallCheck("SBPMSyntaxRule7"))
        ERROR getErrorMessage("SBPMSyntaxRule7",this.name):
        !(this.isStart && this.in.toList().size>0);

context sbpm::SubjectState if(shallCheck("SBPMSyntaxRule8"))
        ERROR getErrorMessage("SBPMSyntaxRule8",this.name):
        !(this.isEnd && this.out.toList().size>0);

context sbpm::FunctionState if(shallCheck("SBPMSyntaxRule9"))
        ERROR getErrorMessage("SBPMSyntaxRule9",this.name):
        !(this.out.toList().size==0 && !this.isEnd);
```

```
context sbpm::ReceiveState if(shallCheck("SBPMSyntaxRule10"))
        ERROR getErrorMessage("SBPMSyntaxRule10",this.name):
        !(this.out.toList().size==0 && !this.isEnd);

context sbpm::SendState if(shallCheck("SBPMSyntaxRule11"))
        ERROR getErrorMessage("SBPMSyntaxRule11",this.name):
        !(this.out.toList().size==0 && !this.isEnd);

context sbpm::MacroClass if(shallCheck("SBPMSyntaxRule11"))
        ERROR getErrorMessage("SBPMSyntaxRule11",this.name):
        !(this.in.toList().size==0 || this.out.toList().size==0 );

context sbpm::AlternativesBar if(shallCheck("SBPMSyntaxRule12"))
        ERROR getErrorMessage("SBPMSyntaxRule12",this.name):
        !((this.in.toList().size + this.out.toList().size)!=1);

context sbpm::ClosedSwitch if(shallCheck("SBPMSyntaxRule13"))
        ERROR getErrorMessage("SBPMSyntaxRule13",this.name):
        !((this.in.toList().size + this.out.toList().size)!=1);

context sbpm::OpenSwitch if(shallCheck("SBPMSyntaxRule13"))
        ERROR getErrorMessage("SBPMSyntaxRule13",this.name):
        !((this.in.toList().size + this.out.toList().size)!=1);

context sbpm::AlternativesBar if(shallCheck("SBPMSyntaxRule14"))
        ERROR getErrorMessage("SBPMSyntaxRule14",this.name):
        !((this.openSwitches.toList().size + this.closedSwitches.toList().size)<2);

context sbpm::sbpm if(shallCheck("SBPMSyntaxRule15"))
        ERROR getErrorMessage("SBPMSyntaxRule15",""):
        !(this.subjectStates().select(e|e.isStart).size==0);

context sbpm::sbpm if(shallCheck("SBPMSyntaxRule16"))
        ERROR getErrorMessage("SBPMSyntaxRule16",""):
        !(this.subjectStates().select(e|e.isEnd).size==0);

context sbpm::AlternativesBar if(shallCheck("SBPMSyntaxRule17"))
        ERROR getErrorMessage("SBPMSyntaxRule17",""):
        !(this.name==null || this.name.trim()=="");

context sbpm::FunctionState if(shallCheck("SBPMSyntaxRule17"))
        ERROR getErrorMessage("SBPMSyntaxRule17",""):
        !(this.name==null || this.name.trim()=="");

context sbpm::SendState if(shallCheck("SBPMSyntaxRule17"))
        ERROR getErrorMessage("SBPMSyntaxRule17",""):
        !(this.name==null || this.name.trim()=="");
```

```
context sbpm::ReceiveState if(shallCheck("SBPMSyntaxRule17"))
        ERROR getErrorMessage("SBPMSyntaxRule17",""):
        !(this.name==null || this.name.trim()=="");

context sbpm::UsedItem if(shallCheck("SBPMSyntaxRule17"))
        ERROR getErrorMessage("SBPMSyntaxRule17",""):
        !(this.name==null || this.name.trim()=="");

context bflow::Connection if(shallCheck("SBPMSyntaxRule18"))
        WARNING getErrorMessage("SBPMSyntaxRule18",""):
        !(this.name==null || this.name.trim()=="");

context sbpm::FunctionState if(shallCheck("SBPMSyntaxRule20"))
        ERROR getErrorMessage("SBPMSyntaxRule20",this.name):
        !((this.outgoingSendArcs().size + this.outgoingReceiveArcs().size)>0);

context sbpm::ReceiveState if(shallCheck("SBPMSyntaxRule21"))
        ERROR getErrorMessage("SBPMSyntaxRule21",this.name):
        !((this.outgoingSendArcs().size + this.outgoingStateArcs().size)>0);

context sbpm::SendState if(shallCheck("SBPMSyntaxRule22"))
        ERROR getErrorMessage("SBPMSyntaxRule22",this.name):
        !((this.outgoingReceiveArcs().size + this.outgoingStateArcs().size)>0);

context sbpm::MacroClass if(shallCheck("SBPMSyntaxRule23"))
        ERROR getErrorMessage("SBPMSyntaxRule23",this.name):
        !((this.outgoingSendArcs().size + this.outgoingReceiveArcs().size)>0);

context sbpm::MacroClass if(shallCheck("SBPMSyntaxRule26"))
        ERROR getErrorMessage("SBPMSyntaxRule26",""):
        !(this.name==null || this.name.trim()=="");

context sbpm::OpenSwitch if(shallCheck("SBPMSyntaxRule27"))
        ERROR getErrorMessage("SBPMSyntaxRule27",this.name):
        !((this.outgoingSendArcs().size + this.outgoingReceiveArcs().size)>0);

context sbpm::ClosedSwitch if(shallCheck("SBPMSyntaxRule28"))
        ERROR getErrorMessage("SBPMSyntaxRule28",this.name):
        !((this.outgoingSendArcs().size + this.outgoingReceiveArcs().size)>0);

context sbpm::AlternativesBar if(shallCheck("SBPMSyntaxRule29"))
        ERROR getErrorMessage("SBPMSyntaxRule29",this.name):
        !((this.outgoingSendArcs().size + this.outgoingReceiveArcs().size)>0);

context sbpm::UsedItem if(shallCheck("SBPMSyntaxRule30"))
        ERROR getErrorMessage("SBPMSyntaxRule30",this.name):
        !((this.inRelationArcs().size + this.outRelationArcs().size)==0);
```

```
context sbpm::UsedItem if(shallCheck("SBPMSyntaxRule31"))
        ERROR getErrorMessage("SBPMSyntaxRule31",this.name):
        !((this.outgoingSendArcs().size + this.outgoingReceiveArcs().size)>0);

context sbpm::MacroClass if(shallCheck("SBPMSyntaxRule34"))
        ERROR getErrorMessage("SBPMSyntaxRule34",this.name):
        !(this.startState.toList().size==0);

context sbpm::MacroClass if(shallCheck("SBPMSyntaxRule35"))
        ERROR getErrorMessage("SBPMSyntaxRule35",this.name):
        !(this.output.toList().size==0);

context sbpm::StateArc if(shallCheck("SBPMSyntaxRule36"))
        ERROR getErrorMessage("SBPMSyntaxRule36",this.name):
        !(this.from.isReceiveState());

context sbpm::StateArc if(shallCheck("SBPMSyntaxRule37"))
        ERROR getErrorMessage("SBPMSyntaxRule37",this.name):
        !(this.from.isSendState());

context sbpm::StateArc if(shallCheck("SBPMSyntaxRule38"))
        ERROR getErrorMessage("SBPMSyntaxRule38",this.name):
        !(this.from.isUsedItem());

context sbpm::StateArc if(shallCheck("SBPMSyntaxRule39"))
        ERROR getErrorMessage("SBPMSyntaxRule39",this.name):
        !(this.from==this.to);

context sbpm::StateArc if(shallCheck("SBPMSyntaxRule40"))
        ERROR getErrorMessage("SBPMSyntaxRule40",this.name):
        !(this.to.isUsedItem());

context sbpm::StateArc if(shallCheck("SBPMSyntaxRule41"))
        ERROR getErrorMessage("SBPMSyntaxRule41",this.name):
        !((this.from.isAlternativesBar()  ||  this.from.isOpenSwitch()  ||
        this.from.isClosedSwitch())  &&  (this.to.isAlternativesBar()  ||
        this.to.isOpenSwitch() || this.to.isClosedSwitch()));

context sbpm::SendArc if(shallCheck("SBPMSyntaxRule42"))
        ERROR getErrorMessage("SBPMSyntaxRule42",this.name):
        !(this.from==this.to);

context sbpm::SendArc if(shallCheck("SBPMSyntaxRule43"))
        ERROR getErrorMessage("SBPMSyntaxRule43",this.name):
        !(!this.from.isSendState());

context sbpm::SendArc if(shallCheck("SBPMSyntaxRule44"))
        ERROR getErrorMessage("SBPMSyntaxRule44",this.name):
```

```
        !(this.to.isUsedItem());

context sbpm::ReceiveArc if(shallCheck("SBPMSyntaxRule45"))
        ERROR getErrorMessage("SBPMSyntaxRule45",this.name):
        !(this.from==this.to);

context sbpm::ReceiveArc if(shallCheck("SBPMSyntaxRule46"))
        ERROR getErrorMessage("SBPMSyntaxRule46",this.name):
        !(!this.from.isReceiveState());

context sbpm::ReceiveArc if(shallCheck("SBPMSyntaxRule47"))
        ERROR getErrorMessage("SBPMSyntaxRule47",this.name):
        !(this.to.isUsedItem());

context sbpm::Relation if(shallCheck("SBPMSyntaxRule48"))
        ERROR getErrorMessage("SBPMSyntaxRule48",this.name):
        !(this.from==this.to);

context sbpm::Relation if(shallCheck("SBPMSyntaxRule49"))
        ERROR getErrorMessage("SBPMSyntaxRule49",this.name):
        !(!(this.from.isFunctionState()||this.from.isMacroClass()
        ||this.from.isUsedItem()));

context sbpm::Relation if(shallCheck("SBPMSyntaxRule50"))
        ERROR getErrorMessage("SBPMSyntaxRule50",this.name):
        !(this.to.isUsedItem() && !(this.from.isFunctionState() ||
        this.from.isMacroClass()));

context sbpm::Relation if(shallCheck("SBPMSyntaxRule51"))
        ERROR getErrorMessage("SBPMSyntaxRule51",this.name):
        !((this.to.isFunctionState() || this.to.isMacroClass()) &&
        !this.from.isUsedItem());

context sbpm::SendArc if(shallCheck("SBPMSyntaxRule53"))
        WARNING getErrorMessage("SBPMSyntaxRule53",this.name):
        !(this.receiver==null || this.receiver.trim()=="");

context sbpm::SendArc if(shallCheck("SBPMSyntaxRule54"))
        ERROR getErrorMessage("SBPMSyntaxRule54",this.name):
        !(this.data==null || this.data.trim()=="");

context sbpm::ReceiveArc if(shallCheck("SBPMSyntaxRule55"))
        WARNING getErrorMessage("SBPMSyntaxRule55",this.name):
        !(this.sender==null || this.sender.trim()=="");

context sbpm::ReceiveArc if(shallCheck("SBPMSyntaxRule56"))
        ERROR getErrorMessage("SBPMSyntaxRule56",this.name):
        !(this.data==null || this.data.trim()=="");
```

```
context sbpm::ReceiveArc if(shallCheck("SBPMSyntaxRule56"))
        ERROR getErrorMessage("SBPMSyntaxRule56",this.name):
        !(this.data==null || this.data.trim()=="");

context sbpm::FunctionState if(shallCheck("SBPMSyntaxRule57"))
        ERROR getErrorMessage("SBPMSyntaxRule57",this.name):
        !(this.incomingControlFlowsFromBarOrSwitch().toList().size>1);

context sbpm::ReceiveState if(shallCheck("SBPMSyntaxRule58"))
        ERROR getErrorMessage("SBPMSyntaxRule58",this.name):
        !(this.incomingControlFlowsFromBarOrSwitch().toList().size>1);

context sbpm::SendState if(shallCheck("SBPMSyntaxRule59"))
        ERROR getErrorMessage("SBPMSyntaxRule59",this.name):
        !(this.incomingControlFlowsFromBarOrSwitch().toList().size>1);

context sbpm::FunctionState if(shallCheck("SBPMSyntaxRule60"))
        ERROR getErrorMessage("SBPMSyntaxRule60",this.name):
        !(this.outgoingControlFlowsToBarOrSwitch().toList().size>1);

context sbpm::ReceiveState if(shallCheck("SBPMSyntaxRule61"))
        ERROR getErrorMessage("SBPMSyntaxRule61",this.name):
        !(this.outgoingControlFlowsToBarOrSwitch().toList().size>1);

context sbpm::SendState if(shallCheck("SBPMSyntaxRule62"))
        ERROR getErrorMessage("SBPMSyntaxRule62",this.name):
        !(this.outgoingControlFlowsToBarOrSwitch().toList().size>1);
```

sbpm.ext context

```
Boolean hasName(bflow::Element element) :
        element.name != null && element.name.length > 0;
cached Collection[sbpm::SubjectState] subjectStates(sbpm::sbpm sbpm):
        sbpm.elements.typeSelect(sbpm::SubjectState);

cached Collection[bflow::Connection] outgoingControlFlows(bflow::Element
element) :
        element.out.typeSelect(bflow::Connection);

cached Collection[bflow::Connection] incomingControlFlows(bflow::Element
element) :
        (element.in.typeSelect(bflow::Connection));

cached Collection[bflow::Connection] outgoingReceiveArcs(bflow::Element
element) :
        element.out.typeSelect(sbpm::ReceiveArc);

cached Collection[bflow::Connection] outgoingSendArcs(bflow::Element element) :
```

```
        element.out.typeSelect(sbpm::SendArc);

cached Collection[bflow::Connection] outgoingStateArcs(bflow::Element element) :
        element.out.typeSelect(sbpm::StateArc);

cached Collection[bflow::Connection] inRelationArcs(bflow::Element element) :
        element.in.typeSelect(sbpm::Relation);

cached Collection[bflow::Connection] outRelationArcs(bflow::Element element) :
        element.out.typeSelect(sbpm::Relation);

cached Collection[bflow::Connection]
incomingControlFlowsFromBarOrSwitch(bflow::Element element) :
        (element.incomingControlFlows().select(e|(e.from.isAlternativesBar()||
e.from.isOpenSwitch() || e.from.isClosedSwitch())));

cached Collection[bflow::Connection]
outgoingControlFlowsToBarOrSwitch(bflow::Element element) :
(element.outgoingControlFlows().select(e|(e.to.isAlternativesBar()||
e.to.isOpenSwitch() || e.to.isClosedSwitch())));

//************************
// simplified type checks
//************************

cached Boolean isFunctionState(bflow::Element element) :
        sbpm::FunctionState.isInstance(element);

cached Boolean isSendState(bflow::Element element) :
        sbpm::SendState.isInstance(element);

cached Boolean isReceiveState(bflow::Element element) :
        sbpm::ReceiveState.isInstance(element);

cached Boolean isMacroClass(bflow::Element element) :
        sbpm::MacroClass.isInstance(element);

cached Boolean isUsedItem(bflow::Element element) :
        sbpm::UsedItem.isInstance(element);

cached Boolean isAlternativesBar(bflow::Element element) :
        sbpm::AlternativesBar.isInstance(element);

cached Boolean isOpenSwitch(bflow::Element element) :
        sbpm::OpenSwitch.isInstance(element);

cached Boolean isClosedSwitch(bflow::Element element) :
        sbpm::ClosedSwitch.isInstance(element);
```

```
cached Boolean isStateArc(bflow::Connection connection) :
        sbpm::StateArc.isInstance(connection);

cached Boolean isReceiveArc(bflow::Connection connection) :
        sbpm::ReceiveArc.isInstance(connection);

cached Boolean isSendArc(bflow::Connection connection) :
        sbpm::SendArc.isInstance(connection);

cached Boolean isRelation(bflow::Connection connection) :
        sbpm::Relation.isInstance(connection)
```

APPENDIX B: eEPC Diagram of Document Approval Process