Chapter 11
Transformation Framework for Consistent Evolution of UML Behavioral Elements into BPMN Design Element

Jayeeta Chanda, Ananya Kanjilal, Sabnam Sengupta and Swapan Bhattacharya

Abstract There are many software products that have been developed in the object-oriented paradigm. To incorporate the positive aspects of service-oriented paradigm (SOA) and address the issues related to increasing size and complexity of software products, they need to be evolved to service-oriented domain. There are some proven Object Oriented (OO) Design Tools that can be used for Service Oriented Application design incorporating both the behavioral and structural aspects in a seamless, consistent evolution that can be made from object oriented to service oriented domain. In this chapter, we concentrate on the evolution process of behavioral aspect of design from OO to SOA. Business Process Modeling Notation (BPMN) has become the de-facto standard for modeling business process on a conceptual level. Business processes are an integral part of service-oriented architecture. In service-oriented applications Use cases needs to be ordered along business processes. Business Processes visualize global control-flow across Use cases. Therefore, use of a business process language to visualize the dependencies among different use cases is of high importance. Use case diagram along with activity diagrams represents the behavioral aspect of a system in the analysis phase of an object-oriented system. To enable modeling the relationship among different behavioral aspects and evolve from object oriented domain to service oriented domain, a formal approach would help in establishing the foundation. In order to do that, in this work, we propose a formal framework, FAM (Formalized analysis model), which is a set of grammar

J. Chanda (✉) · A. Kanjilal · S. Sengupta
BPPIMT, 137, VIP Road, Kolkata, India
e-mail: jayeeta.chanda@gmail.com

A. Kanjilal
e-mail: ag_k@rediffmail.com

S. Sengupta
e-mail: sabnamsg@gmail.com

S. Bhattacharya
National Institute of Technology Karnataka , Surathkal, India
e-mail: bswapan2000@yahoo.co.in

based formalized Use case and Activity diagram elements of UML and a framework for verification of the diagrams, which includes syntactic correctness and requirement traceability. Along with that, we also propose FAM2BP (Formalized Analysis Model to Business Process) for transformation of Formalized Analysis Model (FAM) of object-oriented systems into BPMN process for SOA application using a set of rules that will help in generating business processes for SOA application directly from object oriented analysis models. This model would help in a consistent evolution of software development paradigms from Object Oriented to Service Oriented systems.

11.1 Introduction

Design and development of software has become much more complex in the last decade, resulting in the evolution of design and development paradigms. Object oriented systems have thus become an integral part of more complex Service Oriented Architecture (SOA) to address complex issues like Separation of Concerns, reusability, granularity, modularity, componentization and interoperability. Evolution of software design and development from OO to SOA domain has become the necessity in this evolving scenario. There are some proven OO design tools that can be used for SOA application design. In object-oriented systems, UML is a widely accepted industry standard for modeling of different aspects of the system under construction. Use case diagrams and activity diagrams are used to model the business functional requirements in the analysis phase. These correlate to the business processes of SOA architecture. In service-oriented architecture, BPMN processes play an important role in the development of services. It is the dynamic behavioral diagrams that are often used for modeling business processes, such as the UML Activity diagram and Use Case diagram . BPMN is related to UML in the sense that it defines a visual notation for business processes that is similar to UML behavioral diagrams. However, BPMN and UML have very different approaches to business process modeling. UML offers an object-oriented approach to the modeling of applications, while BPMN takes a process-centric approach that is more natural and intuitive for the business analyst to use. BPMN also offers the option of explicitly modeling business objects that may be exposed through business services in the process flows. Automatic translation of UML use case and activity models to BPMN design elements is thus necessary to ensure consistent evolution of Object oriented systems to Service oriented paradigm.

In this work, we propose a grammar based framework FAM (Formalized analysis Model) for syntactic and semantic verification of UML diagrams in the analysis phase and a relational model based framework FAM2BP (Formalized Analysis Model to Business Processes) for automated translation of elements of FAM to elements of Business Processes, preserving the use case relationship and dependencies and maintaining the control flow of the Business processes. Our framework would enable a consistent evolution of software systems from object oriented paradigm to service oriented paradigm.

11.2 Related Work

Lots of research work are undertaken presently to address various issues in designing and developing software in service oriented paradigm. We discuss some of the significant contributions and present it in the following two subsections. In the first subsection, we discuss the existing works in the domain of formalization of object oriented design modeled by UML diagrams which forms the basis of automated translation and verification. Our proposed framework FAM is presented subsequently in the context of these existing works. In the second subsection, we discuss the work in the domain of relationship between UML use case model and BPMN process model. This forms the basis of our proposed framework FAM2BP for automated evolution of BPMN process models from UML models.

11.2.1 Formalization Approaches

Formalization of UML has become a prominent domain of research for the last few years. Achievement of automated consistency checking and execution has led the software engineers and researchers to focus in this domain. In this section we will discuss a few works done in this domain related to formalization of UML static and dynamic models. To reduce the risks associated with software development and to increase the safety and the reliability by formalizing the syntax of (a sub-set of the popular UML diagrams (Use Case diagram, Class diagram, and State Machine diagram) using Z specifications has been proposed in [1].

The class diagram being the reference point of the notation, any formalization must start with this diagram. To make it possible to provide computer aided support during the application design phase in order to automatically detect relevant properties, such as inconsistencies and redundancies, in [2], UML class diagram is formalized in terms of a logic belonging to Description Logics, which are subsets of First-Order Logic. An algebraic approach is chosen in [3] because it is more abstract than state-based style languages. UML's class diagram (including type definitions, attributes, operations, aggregation and association) and OCL constraints (syntax and semantics), have been formalized using theorem prover Isabelle using one of its built-in logics, HOL.

RSL (RAISE (Rigorous Approach to Industrial Software) Specification Language) has been used in [4] as a syntactic and semantic reference for UML. An automated tool that implements the translation and the abstract syntax in RSL for the RSL-translatable class diagrams are also presented. The integration of the domain modeling method for analyzing and modeling families of software systems with the SOFL formal specification language is discussed in [5]. A UML 1.5 profile named TURTLE (Timed UML and RT-LOTOS Environment) endowed with a formal semantics given in terms of RT-LOTOS is proposed in [6]. Preliminary results on an approach to formally define UML class diagrams using hierarchical predicate transition nets (HPrTNs) have been presented in [7]. The authors show how to define the main concepts related to class diagrams using HPrTN elements.

The semantics presented in [8] captures the consistency between sequence diagram with class diagram and state diagram. This approach may be useful to develop the model consistent checking functions in UML CASE tools and also to reason about the correctness of a design model with respect to a requirement model. The transformation rules for formalizing UML statechart diagrams have been proposed in [9]. The target language for the transformation is Concurrent Regular Expressions (CREs), which are extensions of regular expression. In [10], the alternative approach of using -calculus to formalize UML activity diagrams is presented to get rich process semantics for activity diagrams. This process model can be automatically verified with the help of -calculus analytical tools. Hoare's CSP (communicating sequential processes) has been used in [1] to formalize the behaviors of UML activity diagrams and provides an approach to model checking during software analysis or design stage. The operational semantics of UML sequence diagrams is specified and this specification is extended to include features for modeling multimedia applications as a case study in [11]. Dynamic Meta modeling has been proposed for specifying operational semantics of UML behavioral diagrams based on UML collaboration diagrams that are interpreted as graph transformation rules. The authors in [12] have defined a template to formalize the structured control constructs of sequence diagram, introduced in UML 2.0.

In all these research works, UML diagrams have been formalized using other formal languages. Our earlier work [13] also used Z to propose a formal model for six UML diagrams. However, Z is a non-executable language and hence automated verification is not possible unless translated or mapped to executable models like XML using ZML. In this work, we use context free grammar to formally define the very widely used UML diagrams namely Use case, Activity and Class. This approach is executable unlike the existing approaches like Z-notation etc and hence can be validated using LEX and YACC.

11.2.2 UML Model and BPMN Model Mapping

There exist some works related to the relationship between use case models and BPMN process model. In [14], Cockburn mentions the possibility of applying Use cases for deriving business processes but no rules are proposed. The field of model-driven development has tried to integrate the concept of Use cases within its UML models. Instead of tabular and textual descriptions, UML sequence diagrams or similar models are used in [15]. In [16], an UML based development of business processes is discussed. Expression of control flow between use cases is missing in this approach.

In [17], it is possible to define control-flow dependencies between Use cases with the introduction of Use case Charts and their formalizations. Use cases are called scenarios that may not have extensions and that are modeled as UML sequence diagrams. However, dependencies between Use cases cannot be derived from the Use Case themselves but have to be modeled explicitly.

In [18], synthesis of state transition graphs from Use Cases is addressing the visualization aspect in a better way. A tabular Use case can be converted to a state transition graph similar to graphical business process languages. In [19], the state transition graphs can be used for simulating one Use case but are not suited for visualizing dependencies between Use Cases. The generation of EPC models from Use Cases is addressed in [20]. EPC models consist of fewer graphical symbol types but are not as powerful as BPMN. BPMN has become the standard business process modeling language in SOA. Therefore, the transformation of Use cases should have BPMN as the target notation and our framework is developed upon this concept.

In [21], an algorithm is proposed that restores the overview of the Use cases and visualizes the control flow of the resulting business process. This approach automatically assembles Use cases to business process. But this work is unable to keep the relationship (includes and extends) among use cases and treated as the flat use case model.

Our work is closely related to these works but improves upon them in several aspects. We capture the use case scenarios as a formalized analysis model (FAM) that is a grammar based representation of UML Use case and Activity models. A formal definition of semantics for the subset of BPMN that is applied here has been presented in [22]. Then, a set of rules are proposed that automatically transforms the FAM to BPMN elements (FAM2BP) maintaining the control flow of scenarios as well as preserving all relationships between the use cases.

11.3 Scope of Work

In this chapter, we propose an integrated framework for automatic evolution of UML analysis models to BPMN design elements of service oriented paradigm. The UML use case and activity diagrams that capture the business functional requirements and their flow of events are formally represented as Formal Analysis Model (FAM) which is the grammar based representation of these artifacts. The elements of UML correspond and correlate to the BPMN elements based on which we have designed the transformation framework FAM2BP. The framework consists of a set of rules to map the UML elements like events and flow of events into BPMN elements like start/stop/intermediate events, parallel/exclusive-OR Gateway, etc. A relational model is proposed to represent the relationship among the artifacts. Finally, an algorithm is presented to automatically transform the UML elements into BPMN elements. The block diagram in Fig. 11.1 depicts our approach.

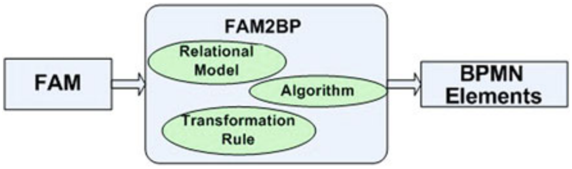

Fig. 11.1 Our integrated framework

11.4 Formalized Analysis Model

UML, being visual in nature, is easy to understand and communicate, but, it lacks the rigor of formal modeling languages and hence verification of a model specified in UML and ensuring requirement traceability within these models becomes difficult. Formalization of UML diagrams is now a dominant area of research. This section is a work in that direction. We have proposed a formal grammar for the Use case, Activity and Class diagrams. We have considered OMG UML 2.0 standard and proposed formal models for some of its constituent diagrams. The production rules, terminals, non-terminals for the grammars are chosen and proposed accordingly. We have also proposed a set of verification criteria that comprises of syntactic correctness rules and traceability rules.

The consistency rules have been proposed by analyzing the inter-relationships among the diagrams so that they together represent a coherent design. Verification of all the rules has been presented based on the proposed grammar. We have used regular expression features (eg. +, *) in the production rules for simplicity and easy understanding. However, for Lex/YACC implementation, we have used a recursive definition of the grammar. Let, the grammar be G = {S, N, T, P}
where S, N, T, P represent start symbol, non-terminals, terminals and production rules
T= {char, digit, +, -, #, association, generalization, aggregation, (,) , .. ,:, basic, alternate, include, extend}
All other symbols used in the production rule P are non-terminals (N).

Grammar for use case Diagram
P:S → usecase_diagram
usecase_diagram → usecase+ actor* UC_relation* actor_relation*
usecase → UC_id UC_name event+
event → event_ID event_name event_type
event_type → basic | alternate
UC_relation → UC_id UC_reltype UC_id
UC_reltype → ≪include≫ | ≪extend≫ | ≪generalization≫
actor_relation → actor actor_reltype actor
actor_reltype → ≪include≫ | ≪extend≫
event_ID → char
event_name → char
UC_id → char
UC_name → char
actor → char
char→ [a-z A-Z 0-9]+
digit → [0 9]

Grammar for activity diagram

```
activity_diagram → activity_state+ transition+ objectflow*
activity_state → act_ID event_ID activity_node act_desc
               className pre_element post_element
activity_node → start | end | join | fork | action | decision | merge
pre_element → className prenode
post_element → className postnode
prenode → start | join | fork | action | decision | merge
postnode → end | join | fork | action | decision | merge
transition → tran_ID prenode postnode objectfl_ID
               | tran_ID prenode postnode
objectflow → objectfl_ID objName className preState poststate
preState → objName = statename
postState → objName = statename
act_ID → char
className → char
objName → char
act_desc → char
tran_ID → char
objectfl_ID → char
statename → char
```

The syntactic rules and traceability rules for the use case diagram and activity diagram are stated as given in the following subsections.

11.4.1 Syntactic Rules

1) The usecase diagram consists of
    a. One or many use cases
    b. Zero or many actors
    c. Zero or many use case relationships
    d. Zero or many actor relationships
2) An usecase consists of One or many events
3) An event has to be of the type basic or alternate.
4) The Use case relationship can be of the type include, extend and generalization.
5) The actor relationship can be of the type include and extend.
6) An activity diagram consists of
    a. A start state
    b. An end state
    c. One or many activity states
    d. Two or many transitions.

7) Zero or many object flows (change in state of an object)
8) Zero or many swimlanes (Transition of state is between two different class)

11.4.2 Traceability Rules

An action/activity state in activity diagram has a one-to-one mapping with an event of a use case in use case diagram.

This traceability rule can be validated using the grammar of use case diagram and activity diagram as stated in the earlier sections.

In this section, we have formalized the analysis phase of OO design elements. These formalized analysis model (FAM) elements are input to the transformation framework FAM2BP. The proposed FAM will establish traceability among the different elements of analysis model. We are establishing traceability in this stage before the transformation because the consistent elements of OO system will generate more robust design elements of the SOA paradigm.

11.5 FAM2BP: Proposed Transformation Model

This section discusses the relational model which maps the artifacts of the two paradigms. The transformation rules and the algorithms for automatic transformation are presented in the following sections.

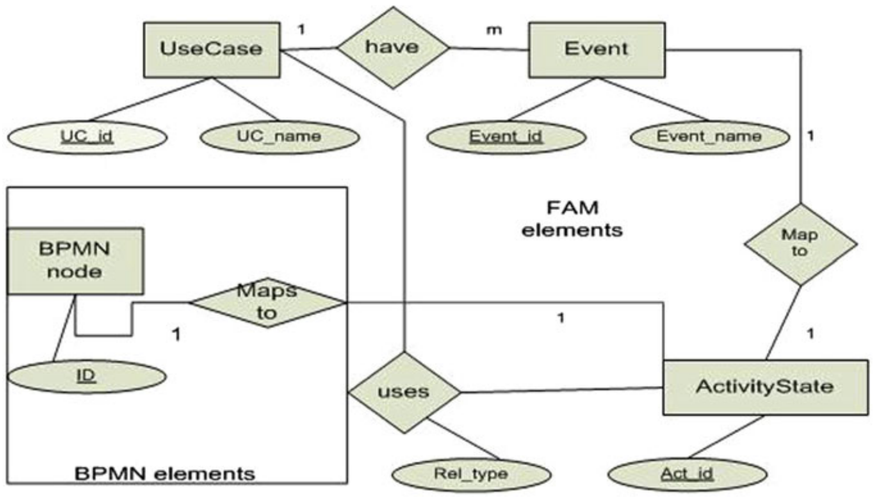

Fig. 11.2 Relational model

11.5.1 Relational Model

The elements of Formalized Analysis Model (FAM) are mapped with the BPMN node. For example, the Use case and Event entity are related to each other. Similarly all other elements of the FAM are related as shown in Fig. 11.2. These elements of FAM model are mapped with the BPMN nodes to ensure automatic transformation from Formalized Analysis Model to Business process. The tables corresponding to this model are generated as follows-

Table Event
(UC_id, event_id, event_name, event_type)

Table ActivityState
(UC_id, event_id, Act_id, activity_node, act_desc, preElement, postElement)

Table ActUCRelation
(Act_ID, UC_id, Rel_type)

Table BPMN_node
(Name, activity_node, Graphical_notation)

The events of the OO behavioral domain are mapped with the BPMN nodes.The SOA behavioral elements BPMN nodes are given below:

- Start Event
- End Event
- Intermediate Event
- Parallel Gateway
- Exclusive-OR Gateway

11.5.2 Transformation Rules

We propose a set of rules to transform Formalized Analysis model into BPMN notation.

Rule 1
The use case activity whose node is marked as *start* will be assigned as the Start Event of the BPMN node. The BPMN node will be labeled as Activity ID (act_ID) of the activity node.

Rule 2
The use case activity whose node is marked as *end* will be assigned as the End Event of the BPMN node. The BPMN node will be labeled as Activity ID (act_ID) of the activity node.

Rule 3
The use case activity whose node is marked as *action / decision* will be assigned as the Intermediate Event of the BPMN node. The BPMN node will be labeled as Activity ID (act_ID) of the activity node.

Rule 4
The use case activity whose node is marked as *fork* will be assigned as the Parallel

Gateway of the BPMN node if both the postElement of the activity node are of the type *basic*. The BPMN node will be labeled as Activity ID (act_ID) of the activity node.

Rule 5

The use case activity whose node is marked as *fork* will be assigned as the Exclusive-OR Gateway of the BPMN node if one the postElement of the activity node are of the type *basic* and the other is of the type *alternate*. The BPMN node will be labeled as Activity ID(act_ID) of the activity node.

These rules are realized in the next section to automate the transformation of Formalized Analysis Model into BPMN nodes.

11.5.3 Algorithm for Automated Transformation

The rules cited in the previous section are realized using two algorithms namely NodeGeneration and FlowGeneration. The flow of the algorithm is as follow:

The elements of Formalized Analysis Model (FAM) in the form of different table schema are used as input to the first algorithm named NodeGeneration The outputs of this algorithm are different BPMN nodes. This output along with the Array FAM_Flow are fed as input to the second algorithm named FlowGeneration. The Array FAM_Flow is formal method of storing the flow information of events of FAM. The FlowGeneration algorithm will generate the BPMN design elements.

11.5.3.1 Algorithm NodeGeneration to Generate BPMN Node

The algorithm NodeGeneration as proposed below will generate the BPMN nodes. We define the algorithm using the tuple relational calculus. The algorithm is proposed as follows-

Query 1:

The following query is the realization of rule 1 of Sect. 11.5.2.It generates the Start of the BPMN node. It selects the *Graphical_notation* from BPMN_node and map that with the *start event* of the activity_node.

{*t.Graphical_notation*| *BPMN_node(t)* ∧ *t.ID =1* ∧
 ∃ *d (d.act_ID* | *ActivityState (d)* ∧ *d.activity_node = t.activity_node* ∧
 t.label = d.act_ID ∧ *d.activity_node = start)*}

Query 2:

The following query is the realization of rule 2 of Sect. 11.5.2. It generates the End of the BPMN node. It selects the *Graphical_notation* from BPMN_node and map that with the *end event* of the activity_node.

{*t. Graphical_notation* | *BPMN_node(t)* ∧ *t.ID = 2* ∧
 ∃ *d(d.act_ID* | *ActivityState(d)* ∧ *d.activity_node =*

t.activity_node ∧ *t.label = d.act_ID* ∧ *d.activity_node = end)*}

Query 3:
The following query is the realization of rule 3 of Sect. 11.5.2. It generates the *Intermediate Event* of the BPMN node. It selects the *Graphical_notation* from BPMN_node and map that with the action or decision event of the activity_node
{t.Graphical_notation | BPMN_node(t) ∧ t.ID = 3 ∧
∧ d(d.act_ID | ActivityState(d) ∧ d.activity_node =
t.activity_node ∧ t.label = d.act_ID ∧ (d.activity_node = action
∨ d.activity_node = decision)) }

Query 4:
The following query is the realization of rule 4 of Sect. 11.5.2. It generates the graphical notation for Parallel Gateway. It selects the particular *graphical notation* and map this with that activity_node of ActivityState where activity_node is *fork* and the event_type of all the postElement of that activity node is basic.
{t.Graphical_notation | BPMN_node(t) ∧ t.ID = 4 ∧
∃ q (q.act_ID | ActivityState(q) ∧ q.activity_node = t.activity_node ∧
t.label = q.act_ID ∧ q.activity_node = fork
∃ r(r.postElement | ActivityState (t) ∧ r.act_ID = q.act_ID ∧
∃ s(s.event_ID | ActivityState(s) ∧ s.act_ID = r.postElement ∧
∃ p(p.event_ID | Usecase(p) ∧ p.event_ID = s.event_ID ∧
s.event_type = basic))))}

Query 5:
The following query is the realization of rule 5 of Sect. 11.5.2. It generates the graphical notation for Exclusive_OR Gateway. It selects the particular *Graphical notation* and map this with that activity_node of ActivityState where activity_node is *fork* and the event_type of the one of postElement of that activity node is *basic* and the event_type of the other postElement of that activity node is *alternate*.

{t.Graphical_notation | BPMN_node(t) ∧ t.ID = 5 ∧
∃ q(q.act_ID | ActivityState(q) ∧ q.activity_node = t.activity_node ∧
t.label = q.act_ID ∧ q.activity_node = fork ∧
∃ r (r.postElement | ActivityState (t) ∧ r.act_ID = q.act_ID ∧
∃ s(s.event_ID | ActivityState(s) ∧ s.act_ID = r.postElement ∧
∃p(p.event_ID | Usecase(p) ∧ p.event_ID = s. event_ID

In this way, the algorithm *NodeGeneration* described in this section will map the different nodes of the OO design elements with that of the BPMN design elements. The algorithm *FlowGeneration* as proposed in the following section will generate the flows between these nodes that are generated by the algorithm *NodeGeneration*.

11.5.3.2 Algorithm FlowGeneration to Generate the Flow Between BPMN Nodes

We use an array representation FAM_flow to represent the flow between different activity nodes. FAM_flow is a part of our Formalized Analysis Model to depict the flow between different events of use cases of objects oriented systems.
The array FAM_Flow is an [n] [3] array where n is the number of flows in the formalized analysis model.

FAM_Flow [0] [i] lists the source activity node of the flow for i=0 to n
FAM_Flow [1] [i] lists the destination activity node of the flow i= 0 to n
FAM_Flow [2] [i] lists the types of flow between A(0,i) and A(1,i) for i= 0 to n

Entries in FAM_Flow [2, i] are of the following types:

1) S indicates sequential flow
2) D indicates Default Flow
3) C indicates Conditional Flow
4) I indicate Iterative flow

Table BPMN_Flow stores different graphical notations of BPMN flows and are assigned with unique IDs.
Table BPMN_Flow will have the following kind of flows:

1) Sequential Flow (*ID is 1*)
2) Default Flow (*ID is 2*)
3) Conditional Flow (*ID is 3*)
4) Iterative flow (*ID is 4*)

The algorithm FlowGeneration is proposed as follow:
Input:

Output of *Nodegeneration algorithm,* FAM_Flow[n] [3], Table BPMN_Flow.

Algorithm:
```
for( m=0 ; m<=n-1;m++)
{
flow. from = FAM_flow[m] [0] ;
flow. to = FAM_flow[m] [1] ;
If FAM_flow [m] [2] = S
Flow.type = Select BPMN_Flow.Graphical_notation where BPMN_Flow.ID=1
If FAM_flow [m] [2] = D
Flow.type = Select BPMN_Flow.Graphical_notation where BPMN_Flow.ID=2
If FAM_flow [m] [2] = C
Flow.type = Select BPMN_Flow.Graphical_notation where BPMN_Flow.ID=3
If FAM_flow [m] [2] = I
Flow.type = Select BPMN_Flow.Graphical_notation where BPMN_Flow.ID=4
}
```

Output:
Different flows of the BPMN design elements.

In this section, we have proposed and described the transformation framework FAM2BP which is BPMN design elements from the elements of the formalized analysis model. As we have ensured the consistency at the OO level before transformation, this transformation will generate consistent design elements at this stage.

11.6 Case Study

Our proposed Automatic Transformation Model FAM2BP is explained with the help of the case study of a Banking System. We have taken four use cases where use case 2 (UC2) is the primary use case that includes use case (UC1) and use case 3 (UC3) and is extended in specialized case like housing loan by use case 4 (UC4).

These use cases are tabulated in Table 11.1 in the form of Use case Schema as defined in Sect. 11.4.2. The events of individual use cases are stored in Table 11.2 as Event schema which is defined in Sect. 11.4.2. These events will be mapped with the ActivityState. The information regarding ActivityState, will be stored in Table 11.3 as ActivityState schema. Table ActivityState contains the information regarding activity node. The entry in the table will have new activity like fork or join or decision etc apart from normal activity (start/action/end) which are mapped from the events of the Event table. The normal activity will carry the same event_ID as in the table Event. And new event_ID will be generated for the new activity. All the activities will be assigned an unique identifier.

Table 11.4 (Table ActUCRelation) which keeps information regarding any activity that includes any use cases. Table 11.4 can be used for extends relation, as well. Here, UC4 extends UC2 and we can replace this with the relation UC4 includes UC2, which implies that UC4 has all the functionalities of UC2, along with its own functionalities. Henceforth, UC4 will have an activity which will include UC2. The Reuse field in Table 11.3 is used to incorporate reusability (include, extend in terms of include relationship) of use cases. If the Reuse field is Y, then Table 11.4 has to be checked to find which usecase has to be included by checking the UC_id field.

Table activitystate (Table 11.3) contains the information regarding activity node. The entry in the table will have new activity like *fork or join or decision* etc apart from normal activity *(start or action or end)* which are mapped from the events of

Table 11.1 Table use case

UC_id	UC_name
UC1	Verify customer
UC2	Sanction loan
UC3	Determine the maximum limit of loan amount
UC4	Sanction home Loan

Table 11.2 Table event

UC_id	Event_id	Event_name	Event_type
UC1	EV1	A customer has called the bank or visit the bank	Basic
UC1	EV2	The customer will be asked the requisite set of questions	Basic
UC1	EV3	Customer is able answer all verification questions successfully	Basic
UC1	EV4	Customer is unable answer verification questions	Alternate
UC1	EV5	Verification is complete	Basic
UC2	EV1	A customer has called the bank	Basic
UC2	EV2	Includes UC1	Basic
UC2	EV3	Includes UC3	Basic
UC2	EV4	Verify address	Basic
UC2	EV5	Finalization of interest rate	Basic
UC2	EV6	Calculation of EMI	Basic
UC2	EV7	Loan is sanctioned	Basic
UC3	EV1	Customer has applied for loan	Basic
UC3	EV2	Income and other factor are taken as input	Basic
UC3	EV3	The maximum loan limit of the customer is calculated	Basic
UC3	EV4	The maximum calculated limit is less than the requested loan limit	Basic
UC3	EV5	Customer loan amount is sanctioned	Basic
UC4	EV1	Customer has applied for home loan	Basic
UC4	EV2	Customer submit property details etc	Basic
UC4	EV3	The searching of property is done and searching result is satisfactory	Basic
UC4	EV4	The searching of property is done and searching result is not satisfactory	Alternate

the Event table (Table 11.2). The normal activity will carry the same event_ID as in the table Event and new event_ID will be generated for the new activity. All the activities will be assigned an unique identifier. As a result , the different BPMN nodes and flows are generated using Tables 11.1, 11.2, 11.3, 11.4 and the array FAM_flow (defined in the previous section). The Table ActUCRelation (Table 11.4) which keeps information regarding any activity that includes any use cases. Table 11.4 can be used for extends relation as well. Here, UC4 extends UC2 and we can replace this with the relation UC4 that includes UC2, which implies that UC4 has all the functionalities of UC2, along with its own functionalities. Henceforth, UC4 will have an activity which will include UC2.

Table 11.3 Table activity state

UC_id	Event_id	Act_id	Activity_node	preElement	postElement	Reuse
UC1	EV1	AC1	start	——	AC2	N
UC1	EV2	AC2	action	AC1	AC3	N
UC1	F	AC3	fork	AC2	AC4,AC5	N
UC1	EV3	AC4	action	AC3	AC6	N
UC1	EV4	AC5	action	AC3	AC2	N
UC1	J	AC6	join	AC4,AC5	AC7	N
UC1	EV5	AC7	end	AC6	——	N
UC2	EV1	AC8	start	——	AC8	N
UC2	F	AC9	fork	AC8	AC10,AC11	N
UC2	EV2	AC10	action	AC9	AC12	Y
UC2	EV3	AC11	action	AC9	AC12	Y
UC2	J	AC12	join	AC10,AC11	AC13	N
UC2	EV4	AC13	action	AC12	AC14	N
UC2	EV5	AC14	action	AC13	AC15	N
UC2	EV6	AC15	action	AC14	AC16	N
UC2	EV7	AC16	end	AC15	——	N

Table 11.4 Table ActUCRelation

Act_id	UC_id
AC10	UC1
AC11	UC3

11.7 Implementation

In this section, we will discuss the implementation details in the direction of evolution from the OO to the SOA domain. We develop a tool that generates the BPMN design elements of the SOA paradigm from the use case design elements of the OO paradigm. The technical environment for this evolution tool comprises of J2SE 1.6. The Integrated Development Environment (IDE) that is used for developing the tool is Net Beans. The tool is developed as a Multi Document Interface (MDI) desktop application using the Java Swing framework (Fig. 11.3).

The program uses a text file as an input. The text file contains the grammatical constructs that define the pre and post elements for each behavioral artifacts, its type etc of OO domain. The grammatical constructs used in the input file are given below:

Usecase name, Element Name, Event Name, Type, Pre Element, Post Element, Reuse

Example:

```
UC1,AC1,EV1,START,-,AC2,N
UC1,AC7,EV7,ACTION,AC1,AC2,N
UC1,AC2,EV2,FORK,AC7,AC3,N
UC1,AC2,EV2,FORK,AC7,AC4,Y
```

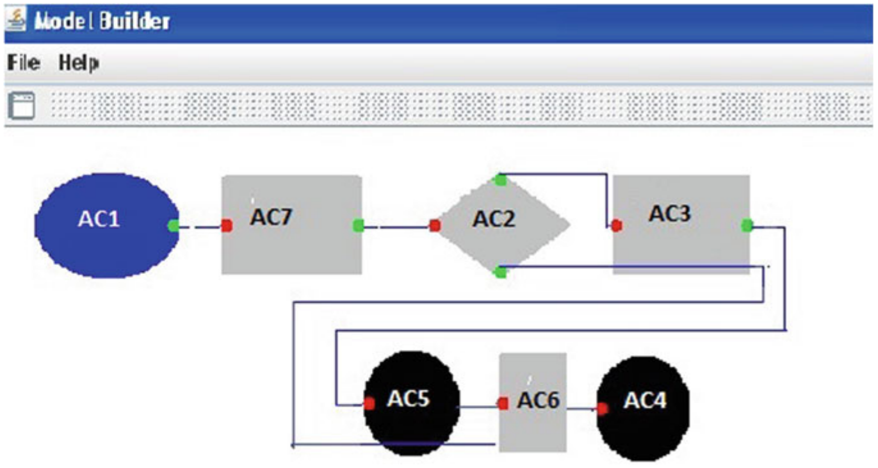

Fig. 11.3 BPMN as generated by our tool

UC1,AC3,EV3,ACTION,AC2,AC5,N
UC1,AC6,EV6,ACTION,AC2,AC4,N
UC1,AC4,EV4,END,AC6,-,N
UC1,AC5,EV5,END,AC3,-,N

11.8 Conclusion

In this chapter, we have proposed an approach for automated translation of Formalized Analysis Models that consists of a formal grammar based description of UML models to Business Processes in the analysis phase. Design and development of software has become much more complex in the last decade resulting in evolution of design and development paradigms. Object oriented systems have thus become an integral part of more complex Service Oriented Architecture (SOA). Evolution of software design and development from the object oriented to the SOA domain has become the necessity in this evolving scenario. This approach would help us in seamless evolution of object oriented systems to the service oriented domain. As this model is based on a formal grammar, this approach can be automated resulting in correct and consistent transformations.

References

1. Mostafa, A.M., Ismail, M.A,. El-Bolok, H., Saad, E.M.: Toward a formalization of UML2.0 metamodel using Z specifications, In: Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007.

SNPD 2007, vol. 1, pp 694–701, July 30–August 1, 2007
2. Cal, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: A formal framework for reasoning on UML Class diagrams. In: Proceedings of the 13th International Symposium on Methodologies for Intelligent Systems (ISMIS 2002), pp. 503–513 (2002)
3. Andre, P., Romanczuk, A., Royer, J.-C.: Checking the consistency of UML class diagrams using Larch prover. In: Rigorous Object Oriented Method (ROOM) (2000)
4. Meng, S., Zhang, N., Aichernig, B.K.: The formal foundations in RSL for UML statechart diagram. Technical Report 299. UNU/IIST, July (2004)
5. Gomaa, H., Liu, S., Shin, M.E.: Integration of the domain modeling method for families of systems with the SOFL formal specification language. In: 6th IEEE International Conference on Complex Computer Systems (ICECCS'00), September 11–15, Tokyo, Japan, pp. 61–71 (2000)
6. Apvrille, L., Courtiat, J.-P., Lohr, C., de Saqui-Sannes, P.: TURTLE: a real-time UML profile supported by a formal validation toolkit. IEEE Trans. Softw. Eng. **30**(7), 473–487 (2004)
7. He, X.: Formalizing UML class diagrams: a hierarchical predicate transition net approach. In: The Twenty-Fourth Annual International Computer Software and Applications Conference, Taipei, Taiwan, 25–28 October 2000
8. Li, X., Liu, Z., He J.: A formal semantics of UML sequence diagram. In: 2004 Australian Software Engineering Conference (ASWEC'04), Melbourne, Australia, 13–16 April 2004
9. Jansamak, S., Surarerks, A.: Formalization of UML statechart models using concurrent regular expressions. In: 27th Australasian Computer Science Conference, The University of Otago, Dunedin, NZ, January (2004)
10. Yang, D., Zhang, s.: Using p - calculus to formalize UML activity diagram. In: 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'03), Huntsville, Alabama, 7–10 April 2003
11. Hausmann, J.H., Heckel, R., Sauer, S.: Towards dynamic meta modeling of uml extensions: an extensible semantics for UML sequence diagrams. In: IEEE 2001 Symposia on Human Centric Computing Languages and Environments (HCC'01), Stresa, Italy, 5–7 September 2001
12. Shen, H., Virani, A., Niu, J.: Formalize UML 2 sequence diagrams. In: 11th IEEE High Assurance Systems Engineering Symposium, HASE 2008, pp. 437–440, 3–5 December 2008
13. Sengupta, S., Bhattacharya, S.: Formalization of functional requirements of software development process, In: In the Journal of Foundations of Computing and Decision Sciences (FCDS). Institute of Computing Science, Poznan University of Technology, Poland **33**(1), 83–115 (2008)
14. Cockburn, A.: Writing Effective Use Cases, 14th edn. Addison-Wesley, New York (2005)
15. Object Management Group (2004). Unified Modeling Language: Superstructure. http://www.omg.org/cgibin/doc?formal/05-07-04. Accessed 1 Sept 2007
16. Oestereich, B., Weiss, C., Schroder, C., Weilkiens, T., Lenhard, A.: Objektorientierte Geschftsprozessmodellierungmit der UML. d.punkt Verlag (2003)
17. Whittle, J.: A formal semantics of Use Case charts, Technical Report ISE Dept, George Mason University, ISE-. TR-06-02. http://www.ise.gmu.edu/techrep
18. Some, S.: An approach for the synthesis of State transition graphs from Use Cases. In: Proceedings of the International Conference on Software Engineering Research and Practice, Las Vegas, Nevada, USA, 23–26 June 2003
19. Some, S.: Supporting Use Cases based requirements simulation. In: Proceedings of the International Conference on Software Engineering and Practice (SERP04), Las Vegas, Nevada, USA, 21–24 June 2004
20. Lbke, D.: Transformation of use cases to EPC models. In: Proceedings of the EPK 2006 Workshop (2006)
21. Lubke, D., Schneider, K., Weidlich, M.: Visualizing Use Case sets as BPMN processes. In: Requirements Engineering Visualization (REV'08), Barcelona, Spain, 8–12 September 2008
22. Dijkman, R., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. In: Information and Software Technology (IST) (2008)