FAM2BP: Transformation Framework of UML Behavioral Elements into BPMN Design Element

Jayeeta Chanda[1], Ananya Kanjilal[1], Sabnam Sengupta[1], and Swapan Bhattacharya[2]

[1] B.P. Poddar Institute of Management & Technology, Kolkata -52
jayeeta.chanda@gmail.com
ag_k@rediffmail.com
sabnam_sg@yahoo.com
[2] National Institute of Technology, Durgapur, 713209, West Bengal
bswapan2000@yahoo.co.in

Abstract. Business processes are an integral part of service-oriented architecture. A typical business process spans multiple Use Cases. Use case diagram along with activity diagrams represents the behavior of a design in the analysis phase for an object-oriented system. In this paper, we propose a relational model, "Formalized Analysis Model to Business Process (FAM2BP)" for transformation of Formalized Analysis Model (FAM) of object-oriented systems into BPMN process for SOA application. FAM consists of a set of grammar based formalized Use case and Activity diagram elements of UML. FAM2BP propose some rules that will help the designer to generate business processes for SOA application directly from object oriented analysis models. This model would help in evolution of software design and development paradigms from Object Oriented to Service Oriented systems.

Keywords: Service-oriented-architecture, BPMN-Process, Relational-model, Translation from OO to SOA, UML to BPMN, Formal UML model.

1 Introduction

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) to address complex issues like Separation of Concerns, reusability, granularity, modularity, componentization and interoperability. Evolution of software design and development from object oriented to SOA domain has become the necessity in this evolving scenario. We already have some proven Object Oriented design tools that can be used for SOA application design. In software projects, Use case diagrams and activity diagrams are used to model the business functional requirements and its flow of events 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. Automatic translation of UML use case and activity models to BPMN design elements would ensure consistent evolution of Object oriented systems to Service oriented paradigm. In this paper, we propose a relational model based framework to address this need. The automatic transformation helps the designer to develop the service oriented architecture in a more convenient manner as well as ensuring consistency and correctness in the design.

2 Review of Related Work

There exists some work related to the relationship between use case models and BPMN process model. In [1], 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 [4]. Use case diagrams are used to define use cases and other models are used to define scenario. In [6], an UML based development of business process is discussed. Expression of control flow between use cases is missing in this approach.

In [9], it is possible to define control-flow dependencies between Use Cases with the introduction of Use Case Charts and their formalization. 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 [7], synthesis of state transition graphs from Use Cases is better addressing the visualization aspect. A tabular Use Case can be converted to a state transition graph similar to graphical business process languages. In [8], 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 [3]. 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 [10], 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 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 in this paper has been presented in [2]. Then a set of rules are proposed that automatically transforms the FAM to BPMN elements maintaining the control flow of scenarios as well as preserving all relationships between the use cases.

3 Scope of Work

In this paper we propose a framework for automatic transformation of UML analysis models to BPMN design elements for Service Oriented paradigm.. In this framework the elements of formalized analysis model (FAM) are transformed into the elements of business processes. FAM consists of some grammar based formalized Use case and Activity diagram elements of UML as done in our work [11]. 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 between all the artifacts. Finally an algorithm is presented to automatically transform the UML elements into BPMN elements. The block diagram in figure1 depicts our approach.

4 FAM2BP: Proposed Transformation Model

The input to our model is the formal analysis model (FAM) which is presented in the following section 4.1. The next section 4.2 discusses the relational model which maps the artifacts of the two paradigms. The transformation rules are presented in 4.3 and the algorithms for automatic transformation are presented in 4.4.

4.1 Formalized Analysis Model

As proposed in our work [11], we are using the Formalized Analysis Model that consists of context free grammar for the Use Case and Activity Diagrams. The grammars are as proposed in our work [11].

4.2 The Relational Model

As shown in figure1, the elements of Formalized Analysis Model (FAM) are mapped with the BPMN node. For example, the *Usecase* and *Event* entity are related to each other by *have* relation. Similarly all other elements of the FAM are related as shown in Figure 2.These elements of FAM model are mapped with the BPMN node to ensure automatic transformation from Formalized Analysis Model to Business process. The tables corresponding to this model are generated in Figure 2.

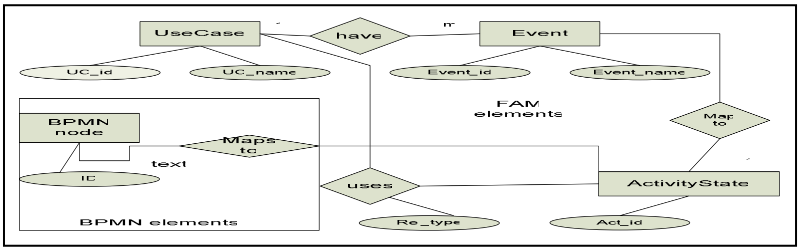

Fig. 1. The proposed Relational Model between different elements of FAM

4.3 Transformation Rules

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

The rules are the defined as follow:

Rule1

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

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

ID	Name	Activity node	Graphical_ notation	label
1	Start event	Start	○	
2	End Event	End	◯	
3	Intermediate Event	Action, decision	◌	
4	Parallel Gateway	Fork, join	◈	
5	Exclusive-OR Gateway	Fork , join	◇	

Fig. 2. Relational Model for FAM in tabular form

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

Rule3
The use case activity whose node is marked as 'action / decision' will be assigned as Intermediate Event of the BPMN node. The BPMN node will 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 Parallel Gateway of the BPMN node if both the postElement of the activity node are of the type 'basic'. The BPMN node will 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 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 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.

4.4 Algorithm for Automated Transformation

These rules cited in section 4.3 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* (devised in section 4.3.1). The outputs of this algorithm are different BPMN nodes. This output along with the Array FAM_Flow is 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.

4.4.1 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 follow.

# The following query is the realization of rule 1 of section 4.3.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')}

# The following query is the realization of rule 2 of section 4.3.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')}

# The following query is the realization of rule 3 of section 4.3.It generates the #*Intermediate* 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')) }

# The following query is the realization of rule 4 of section 4.3.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'))))}*

# The following query is the realization of rule 5 of section 4.3.It generate 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 ^ s.event_type =
' basic' ∨ s.event_type = 'alternate')))))}

The algorithm *FlowGeneration* as proposed in section 4.3.2 will generate the flows between these nodes that are generated by the algorithm *NodeGeneration*.

4.4.2 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

Array FAM_Flow

A [] []	0	1	2	3	4	5	6	7	…	n-1
0(=ACi)	AC1	AC2	AC3	AC3	AC4	AC5				
1(=ACj)	AC2	AC3	AC4	AC5	AC6	AC2				
2(=value)	S	S	D	S	S	I				

Table BPMN_Flow stores different graphical notation of BPMN flow and are assigned with unique ID.

Table BPMN_Flow

ID	Name	Graphical_notation
1	Sequential Flow	$\longrightarrow$
2	Default Flow	$\not\longrightarrow$
3	Conditional Flow	$\xrightarrow{\text{condition}}$
4	Iterative flow	$\dashrightarrow$

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:

BPMN design elements

5 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 1(UC1) & use case 3(UC3) and is extended in specialized case like housing loan by use case 4(UC4). These use cases are tabulated in Table1 in the form of *Use case* Schema as defined in section 4.2. The events of individual use cases are stored in Table2 as *Event* schema which is defined in section 4.2. These events, which will be mapped with the *ActivityState* .The information regarding ActivityState, will be stored in Table 3 as *ActivityState* schema.

Table *ActivityState* contains the information regarding activity node. The entry in the table will have new activity like fork /join/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.

In Figure 5, the different BPMN nodes and flows are generated using Tables 1, 2, 3, 4 and the array FAM_flow (defined in the previous section). The table 4 (Table ActUCRelation) which keeps information regarding any activity includes any use cases. Table 4 can be used for extend 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 3 is used to incorporate reusability (include, extend in terms of include relationship) of use cases. If the Reuse field is 'Y', then table 4 has to be checked to find which usecase has to be included by checking the UC_id field.

Table 1. Table Use Case

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

Table 4. Table ActUCRelation

Act_ID	UC_id
AC10	UC1
AC11	UC3

Table 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	Include 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.	Alternate
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

Table *ActivityState* (table 3) contains the information regarding activity node. The entry in the table will have new activity like fork /join/decision etc apart from normal activity (start/action/end) which are mapped from the events of the *Event* table (table 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 1, 2, 3, 4 and the array FAM_flow (defined in the previous section). The Table ActUCRelation (table 4) which keeps information regarding any activity includes any use cases. Table 4 can be used for extend 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.

Table 3. Table ActivityState

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,AC111	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

6 Conclusion

In this paper, we have proposed an approach for automated translation of Formalized Analysis Models, that consists of a formal grammar based description of UML models used in Analysis phase, to Business Processes. 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 object oriented to SOA domain has become the necessity in this evolving scenario. This approach would help us in seamless evolution of object oriented systems to 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] Cockburn, A.: Writing Effective Use Cases, 14th edn. Addison-Wesley, Reading (August 2005)

[2] Dijkman, R., Dumas, M., Ouyang, C.: Semantics and Analysis of Business Process Models in BPMN. Information and Software Technology (IST) (2008)

[3] Lübke, D.: Transformation of Use Cases to EPC Models. In: Proceedings of the EPK 2006 Workshop, Vienna, Austria (2006), `http://ftp.informatik.rwth-aachen.de/Publications/CEURWS/Vol-224/`

[4] Object Management Group. Unified Modeling Language: Superstructure (2004), `http://www.omg.org/cgibin/doc?formal/05-07-04` (last access 2007-09-01)

[5] Object Management Group. Business Process Modeling Notation (BPMN) 1.1 (January 2008)

[6] Oestereich, B., Weiss, C., Schröder, C., Weilkiens, T., Lenhard, A.: Objektorientierte Geschftsprozessmodellierungmit der UML. d.punkt Verlag (2003)

[7] Somé, 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, June 23 - 26 (2003)

[8] Somé, S.: Supporting Use Cases Based Requirements Simulation. In: Proceedings of the International Conference on Software Engineering and Practice (SERP 2004), Las Vegas, Nevada, USA, June 21-24 (2004)

[9] 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`

[10] Lübke, D., Schneider, K., Weidlich, M.: Visualizing Use Case Sets as BPMN Processes. In: Requirements Engineering Visualization (REV 2008), Barcelona, Spain, September 8-12 (2008)

[11] Chanda, J., Kanjilal, A., Sengupta, S., Bhattacharya, S.: Traceability of Requirements and Consistency Verification of UML Use case, Activity and Class Diagram: A Formal Approach. In: Proceedings of International Conference on Methods and Models in Computer Science (IEEE ICM2CS), New Delhi, December 14-15 (2009