The Function-Behaviour-Structure Diagram for Modelling Workflow of Information Systems

Stanislaw Jerzy Niepostyn and Ilona Bluemke

Institute of Computer Science, Warsaw University of Technology,
Nowowiejska 15/19, 00-665 Warsaw, Poland
{S.Niepostyn,I.Bluemke}@ii.pw.edu.pl

Abstract. Currently, no single UML diagram provides the satisfactory completeness and consistency of the system description. There is also no BPMN diagram to satisfy such requirements. The satisfactory completeness means that the model enables to describe fully a function, a structure, and a behaviour of the IT system. With BPMN diagram one cannot provide a complete data model i.e. the structure of the IT system. The proposed Function-Behaviour-Structure activity diagram introduced in this paper enables to develop consistent and satisfactorily complete models.

Keywords: MDA, Business Process Management, UML, BPMN, activity diagram, consistency, completeness.

1 Introduction

In the beginning of the year 2001 the Object Management Group (OMG) launched Model Driven Architecture [1] based on transformational approaches (generating models from other models). The main concept of the MDA standard specifies rules for building application which are generated from models at higher (business) level of abstraction. Despite a lot of research conducted during many years MDA tools still have not been produced on an industrial scale. The cause of such situation is mainly resulting from lacking rules defining how the elements from different models relate to each other. The reason for the lack of such rules is the semantic of the UML [2] models defined in natural language. UML cannot provide a straightforward way of representing a connector (association) and there is no specific construct for representing architectural styles [3]. UML cannot fully define the relationships between diagrams so the consistency across diagrams must be ensured manually [4] or expressed e.g. in OCL [5].

Model Driven Architecture is in the stagnation for the last few years while Business Process Management (BPM) [6] has been quickly developing. BPM systems support execution of business processes (workflow applications) according to BPM concepts [7]. BPM platforms also allow users to build and deploy business process models supported with Business Process Execution Language for Web Services (BPEL) [8] and XML Process Definition Language (XPDL) [9] standards. Today's

BPM platforms offer their own notations to describe business processes. Therefore, since January 2011, OMG proposed the Business Process Model & Notation (BPMN) [10] language to build and deploy business process models on BPM platforms. It can be noticed that BPMN ensures the consistency of the business processes (workflow). BPMN provides only the description of a system behaviour and the description of its structure and function are neglected. The lack of sufficient structure description of the system can become soon a significant restriction to the full automation in BPM solutions.

In this paper we introduce a new Function-Behaviour-Structure Activity diagram, which enables to keep the consistency and satisfactory completeness of the application model. Some model elements are formally described in Z notation [11, 11] in section 3 to provide the consistency proof. In section 2 the dimensions of software architecture are described, then completeness and types of inconsistencies are described. The rationale of applying the FBS activity diagram in the design view of the software architecture to derive the complete and consistent UML model is given in section 4. Related work and some conclusions are given in sections 5 and 6.

2 Incompleteness and Consistency of Model

According to Function-Behaviour-Structure (FBS) framework introduced by Gero [13] the purpose of the design's description is to transfer sufficient information about target system so it can be constructed. The description must at least enable to incorporate a function, a structure, and a behaviour of the target system. Therefore the development of software in which one cannot take into account these three dimensions, are "doomed to fail". Truyen [14] described a model, in major MDA concepts, as a formal specification of the function, structure and behaviour of a system. He claims, that model must be represented by a combination of UML diagrams. Spanoudakis and Zisman [15] described this as a situation, in which model inconsistencies may arise.

Below we explain informally the model consistency, which we subsequently apply in the analysis of selected diagrams. Then we present our concept of the dimensions of the software architecture which form consistent description of software architecture.

2.1 Model Inconsistencies

To assert that something is consistent we have to declare what it is consistent with. Software models describe system from different points of view, at different levels of abstraction and granularity, in different notations. They may represent viewpoints and goals of different stakeholders. Usually inconsistencies between diagrams are arising. Inconsistencies reveal design problems. The roots of consistency can be found in formal methods. The research on consistency models was started by Finkelstein [16]. Finkelstein stated, that inconsistency is not necessarily a bad thing, and should be evaluated after the translation of the model specification into formal logic. UML is not a formal language so often UML models are translated into more formal notation. UML is widely used in the software design so the problem of inconsistency in UML models received special attention. In UML inconsistencies between class, state machine and sequence diagrams [17] are studied. Inconsistencies arise because some models are overlapping [15]

UML consistency analysis goes far beyond checking syntax and semantics, it should also encompass other domains like targeted programming language, modelling methodology, modelled systems, and application and implementation domains.

Mens [18] proposed five consistency types:

1. Inter model (vertical) consistency. Consistency is evaluated between different diagrams and different levels of abstraction. The syntactic and semantic consistencies are also taken into account.
2. Intra model (horizontal) consistency. Consistency is validated between different diagrams but at the same level of abstraction.
3. Evolution consistency. Consistency is validated between different versions of the same UML diagram.
4. Semantic consistency. Consistency is validated for the semantic meaning of UML diagram defined by an UML metamodel.
5. Syntactic consistency. Consistency is validated for the specification of UML diagrams in an UML metamodel.

Another classification divides consistency into static and dynamic. Static consistency can be verified without running the model while dynamic constraint cannot be verified until runtime. In [19] a survey of consistency checking techniques for UML models is presented. The existing techniques are classified based on intermediate representation into three categories: formally represented, extended UML - intermediate representation is defined as an extension in UML diagrams and without intermediate representation. Many interesting information on consistency problems in UML-based software development can be found in [20].

2.2 Dimensions of the Software Architecture

In the majority of projects using UML diagrams [21, 22], use case diagrams are developed at the beginning of software development to describe the main functions of the software-based system. Then class diagrams are created to show the structure of the system, and state machine diagrams are built to show the behaviour of system's elements ([23, 24]). Subsequently activity or sequence diagram can be used in order to verify consistency of other diagrams. These diagrams are also using visualizing scenarios i.e. – use case realization diagrams.

Activity diagram enables to associate activities with objects (instantiate classes), and use-cases ([23, 24, 25]). It can be noticed that Use Case, Class and State Machine diagrams are orthogonal (Fig. 1), and enable to derive use case realization diagram [26]. A model, which adequately integrates these diagrams thus enables to keep the consistency and the satisfactory completeness of the whole system because these three diagrams do not have common elements. Anyone can interpret the operation of the class (dimension of the structure), the state in statechart (dimension of the behaviour), and use case (dimension of the functionality) as a single element of the integrated model. Such integrated model (diagram) enables to achieve *satisfactory* completeness. We define *satisfactory* completeness as comprising necessary elements (listed above) and at least one element that integrates all those three dimensions of the software architecture.

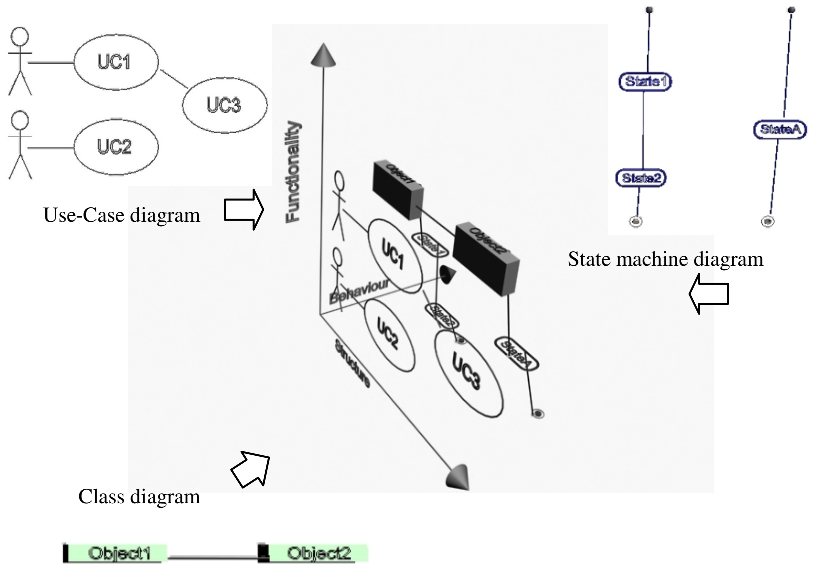


Fig. 1. Three dimensions of the software architecture view

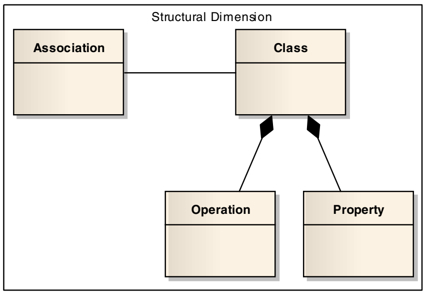
3 Analysis of Some Diagrams

In this section we analyse the consistency and the satisfactory completeness of class, state machine and use case diagrams. In this analysis we use simplified metamodels, without cardinalities, and Z schemas describing some metamodel's elements used in the consistency reasoning. Cardinalities do not change the results of consistency analysis. Next, we show that other diagrams like activity, sequence or BPMN diagrams do not have properties required to sufficiently describe the target system so we propose such a diagram in section 4.

3.1 Class Diagram, State Machine Diagram, and Use Case Diagram

The simplified metamodel of the class diagram is presented in Fig. 2.a and its formalization using Z schemas is shown in Fig. 2.b.

a)



b)

Operation
id:Operation_ID

Property
ident:Property_ID

Class
id:Class_ID
operations:ℙ Operation
properties: ℙ Property

Association
id:Association_ID
classes: ℙ Class
associations: Class<->Class

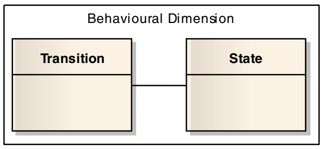
StructureDim
operations:ℙ Operation
properties: ℙ Property
classes: ℙ Class
Association

Fig. 2. a) UML metamodel of a class b) Z formalisation of the class diagram

In the schema *StructureDim*, classes are the set of *Class* instances, operations are the set of *Operation* instances, and properties are the set of *Property* instances. Class diagram describes only the dimension of the structure of the system. All elements describe the structure of the system but some behaviour properties could be generated from *Operation* [27].

The simplified metamodel of the state machine diagram is presented in Fig. 3.a and its formalization is shown in Fig. 3.b. In the schema *BehaviourDim* states are the set of *State* instances. State machine diagram describes only the dimension of the behaviour of the system.

a)



b)

State
id:State_ID

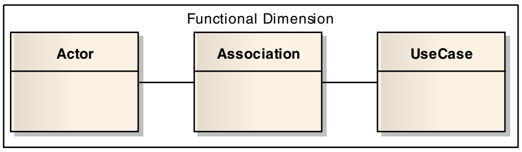
Transition
id:Transition ID
transitions:State<->State

BehaviourDim
states:ℙ State
Transition

Fig. 3. a) UML metamodel – state machine diagram, b) Z - formalisation of the statechart - dimension of behaviour

The use case diagram (dimension of functionality) is presented in Fig. 4.a, and its formalisation in Z schemas is shown in Fig. 4.b. In the schema *FunctionalDim* use cases are the set of *UseCase* instances and actors are the set of *Actor* instances. Use case diagram describes only the dimension of the functionality of the system. *Actor* and *UseCase* are defined in the standard UML as the elements that describe the behaviour rather than the functionality of the system.

a)



b)

```
UseCase
id:UseCase_ID

Actor
id:Actor_ID

Association
id:Association_ID
usecases: ℙ UseCase
actors: ℙ Actor
associations: UseCase<->Actor

FunctionalDim
usecases: ℙ UseCase
actors: ℙ Actor
Association
```

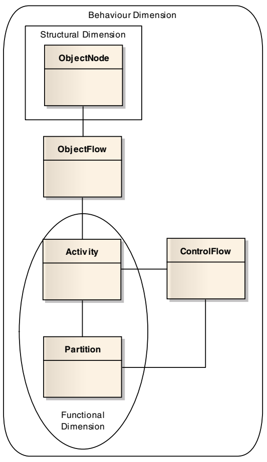
Fig. 4. a) Metamodel of the use case, b) Z - formalisation of the dimension of functionality

Comparing all diagrams (dimensions) with each other no common element can be found so those three UML diagrams must be consistent. Moreover, those three UML diagrams could describe satisfactory completeness of an IT system if the operation, the state, and the use case elements are integrated into a single UML diagram.

3.2 Activity Diagram

The activity diagram is presented in Fig. 5.a and its formalisation in Z schemas is shown in Fig. 5.b.

a)



b)

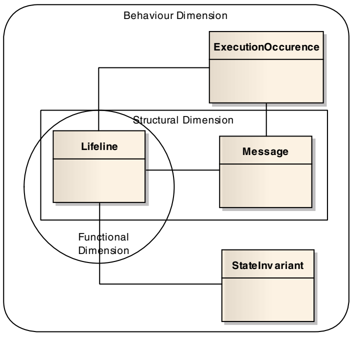
```
Activity
id: Activity_ID

Partition
id:Partition_ID

ObjectNode
id:ObjectNode_ID

ControlFlow
id:ControlFlow_ID
activities: ℙ Activity
controlflows: Activity<->Activity

ObjectFlow
id: ObjectFlow_ID
activities: ℙ Activity
objectnodes: ℙ ObjectNode
objectflows: Activity<->ObjectNode

FunctionalDim
activities: ℙ Activity
partitions: ℙ Partition

StructuralDim
objectnodes: ℙ ObjectNode

BehaviouralDim
activities: ℙ Activity
partitions: ℙ Partition
objectnodes: ℙ ObjectNode
ControlFlow
ObjectFlow

ADConsistency
BehaviouralDim
StructuralDim
FunctionalDim
----------------------------------------
BehaviouralDim ∩ StructuralDim = StructuralDim
BehaviouralDim ∩ FunctionalDim = FunctionalDim
StructuralDim ∩ FunctionalDim = ∅
BehaviouralDim ∩ StructuralDim ∩ FunctionalDim = ∅
```

Fig. 5. a) Metamodel of the activity diagram, b) Z - formalisation of the activity diagram

The dimension of the functionality describes *Partition* and *Activity*. *ObjectNode* represents the dimension of the structure, and the dimension of the behaviour contains all elements of the activity diagram. There is no common element in all three dimensions. Thus it is not possible to integrate the three dimensions of software architecture in this activity diagram. The dimension of the behaviour has common elements with the dimension of the structure, and also with the dimension of the functionality. It means, that the other elements of the activity metamodel are dependent on each other so the three dimensions overlap with each other. According to Spanoudakis and Zisman [15] "inconsistencies arise because the models overlap" therefore, the three dimensions of the activity model are not consistent [28]. This property implies also that the corresponding UML models (use case diagram, state machine diagram, class diagram) may not be consistent.

3.3 Sequence Diagram

The sequence diagram is presented in Fig. 6.a, and its formalisation in Z schemas is shown in Fig. 6.b. The dimension of functionality describes *lifelin*e. Lifeline and *message* represent the dimension of structure, and the dimension of behaviour contains all elements of the sequence diagram. There is a common element in the three dimensions so it is possible to integrate the three dimensions of software architecture in this sequence diagram. The other elements of the interaction metamodel are dependent on each other so the three dimensions overlap with each other. According to Spanoudakis and Zisman [15] inconsistency definition the three dimensions of the interaction model are not consistent. This property implies also that the corresponding UML models (use case diagram, state machine diagram, class diagram) may not be consistent.

a)


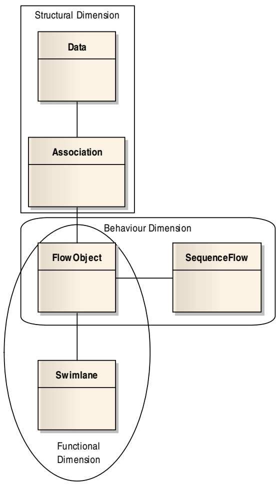
b)

```
┌ Lifeline ─────
│ id: Lifeline _ID
└──────────────

┌ StateInvariant──
│ id:StateInvariant_ID
└──────────────

┌ ExecutionOccurence ─────
│ id:ExecutionOccurence_ID
└──────────────

┌ Message ─────────
│ id:Message_ID
│ occurences: ℙ ExecutionOccurence
│ messages:ExecutionOccurence<->ExecutionOccurence
└──────────────────

┌ StructuralDim ─────────
│ lifelines: ℙ Lifeline
│ Message
└──────────────────

┌ FunctionalDim──────────
│ lifelines: ℙ Lifeline
└──────────────────

┌ BehaviouralDim ────────
│ lifelines: ℙ Lifeline
│ occurences: ℙ ExecutionOccurence
│ states: ℙ StateInvariant
│ Message
└──────────────────

┌ SDConsistency ─────────
│ BehaviouralDim
│ StructuralDim
│ FunctionalDim
├──────────────────
│ BehaviouralDim ∩ StructuralDim = StructuralDim
│ BehaviouralDim ∩ FunctionalDim = Lifeline
│ StructuralDim ∩ FunctionalDim = Lifeline
│ BehaviouralDim ∩ StructuralDim ∩ FunctionalDim = Lifeline
└──────────────────
```

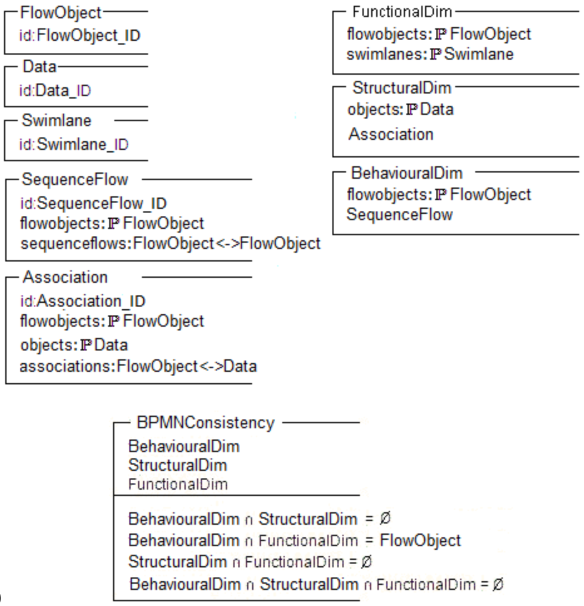
Fig. 6. a) Metamodel of the sequence diagram, b) Z - formalisation of the sequence diagram

3.4 BPMN Diagram

The BPMN metamodel is presented in Fig. 7.a and its formalisation in Z schemas is shown in Fig. 7.b.



a)



```
FlowObject
id:FlowObject_ID

Data
id:Data_ID

Swimlane
id:Swimlane_ID

SequenceFlow
id:SequenceFlow_ID
flowobjects:ℙ FlowObject
sequenceflows:FlowObject<->FlowObject

Association
id:Association_ID
flowobjects:ℙ FlowObject
objects:ℙ Data
associations:FlowObject<->Data

FunctionalDim
flowobjects:ℙ FlowObject
swimlanes:ℙ Swimlane

StructuralDim
objects:ℙ Data
Association

BehaviouralDim
flowobjects:ℙ FlowObject
SequenceFlow

BPMNConsistency
BehaviouralDim
StructuralDim
FunctionalDim
----------------------------------
BehaviouralDim ∩ StructuralDim = ∅
BehaviouralDim ∩ FunctionalDim = FlowObject
StructuralDim ∩ FunctionalDim = ∅
BehaviouralDim ∩ StructuralDim ∩ FunctionalDim = ∅
```

b)

Fig. 7. a) Metamodel of the BPMN diagram, b) Z - formalisation of the BPMN diagram

The dimension of functionality describes *Swimlane* and *FlowObjec*t. *Data* and Association represent the dimension of structure, and the dimension of behaviour contains *FlowObject* and *SequenceFlow* elements of the BPMN diagram. As no common element is present in the three dimensions, it is not possible to integrate the three dimensions of software architecture in this BPMN diagram. It means that BPMN model does not have properties to satisfactory describe the target system. It can be noticed that the dimension of the structure is unsatisfactory to perform mapping of a class diagram to *Data* and *Association* elements. Therefore with BPMN diagram one cannot provide a complete data model.

4 FBS Activity Diagram

The FBS activity diagram enables to build a model integrating the three dimensions of software: functional, structural and behavioural. In Fig. 8 an example of a routine task in an office modelled by FBS activity diagram is shown.

The header of the diagram describes the objects and the first column shows the Actors. In following columns the activities are given, each one is performed by an appropriate actor. There are several kinds of the activities: `Creating`, `Checking`, `Archiving`, `Approving` and `Other`. These activities have the incoming and outcoming instances of the classes. Figure 8 presents a request of a service from an office. A *Customer* fills a written request (Creating request), then *Clerk* checks this request (Checking request). After this checking, the *Clerk* looks into it (Creating opinion). The *Supervisor* accepts the request (Approving opinion and request) and

Clerk archives his decision (Archiving request and opinion). Then the *Clerk* prepares the reply (Creating reply), the *Supervisor* accepts it (Approving reply) and, at the end, the *Customer* receives it (Receiving reply).

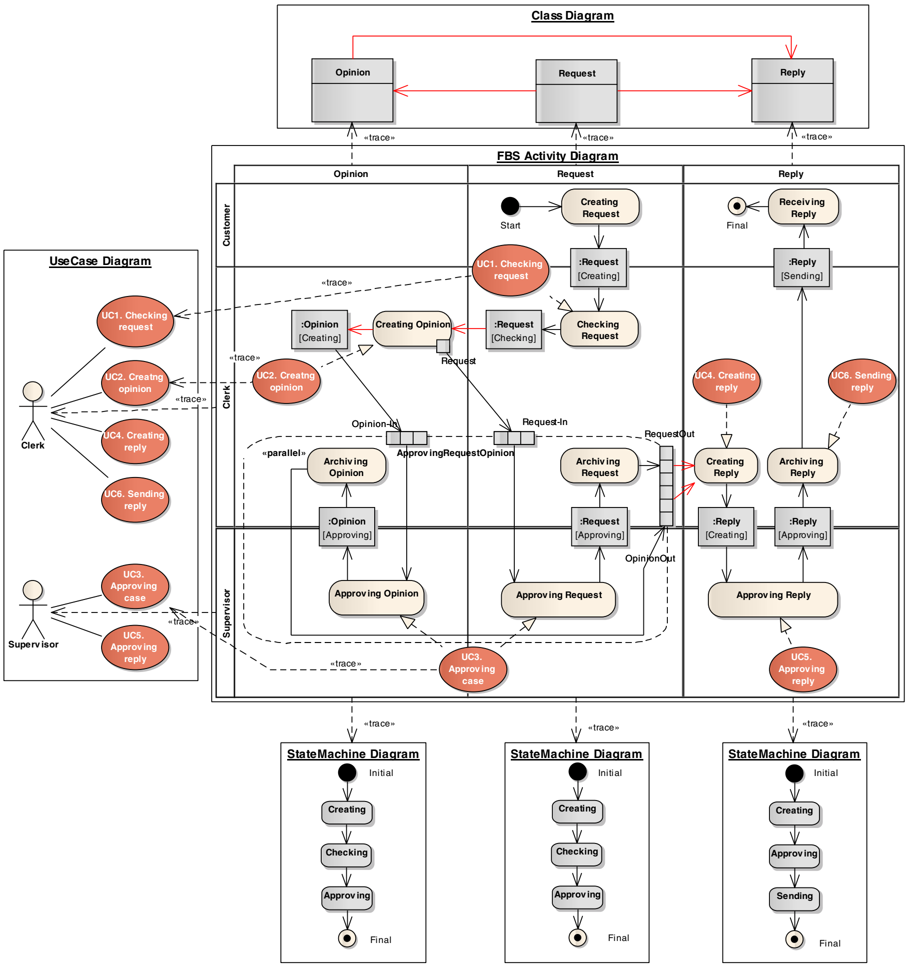


Fig. 8. The Function-Behaviour-Structure activity diagram

In Fig. 8 the mappings between FBS activity model and UML diagrams are also shown. These diagrams describe the design view of the software architecture. The FBS activity model is in simple and unambiguous relationships with class diagram (structure), state diagram (behaviour), and use case diagram (functionality).

Each element in the header of the FBS activity model corresponds to only one object, which is instance of a proper class from the class diagram. The associations between objects are derived from the edges of horizontal object flow (in red colour). Moreover, each FBS object has simple and unambiguous state diagram. Each FBS object with its state corresponds to only one state in the state diagram. Transitions in this state chart are derived from FBS activity with horizontal object flow between FBS objects. In similar way the FBS activities can be mapped onto use case diagram.

A few FBS activities are realized by one use case, and each use case is associated with an actor in use case diagram. The *Actor* is derived from the horizontal *Partition*, which is grouping particular FBS activities. In order to improve the readability of Fig. 8, not all dependencies between diagrams are visible.

4.1 Satisfactory Completeness of the FBS Activity Model

In Fig. 9 the simplified UML activity FBS meta-model is presented. The dimension of functionality describes *Actors*, *Use Cases*, and *Activities*. *States of Objects* with *Activities* and *verticalObjectFlow* represent the dimension of behaviour, and the dimension of structure contains *Objects*, *horizontalObjectFlow* and *Activities*.

Z-formalization of the FBS activity diagram is shown in Fig. 10. The common element of the three dimensions is *Activities,* it could be used to integrate the three dimensions of software architecture in this diagram. Other elements of the FBS model fully describe the three dimensions so the FBS activity diagram is satisfactorily complete.

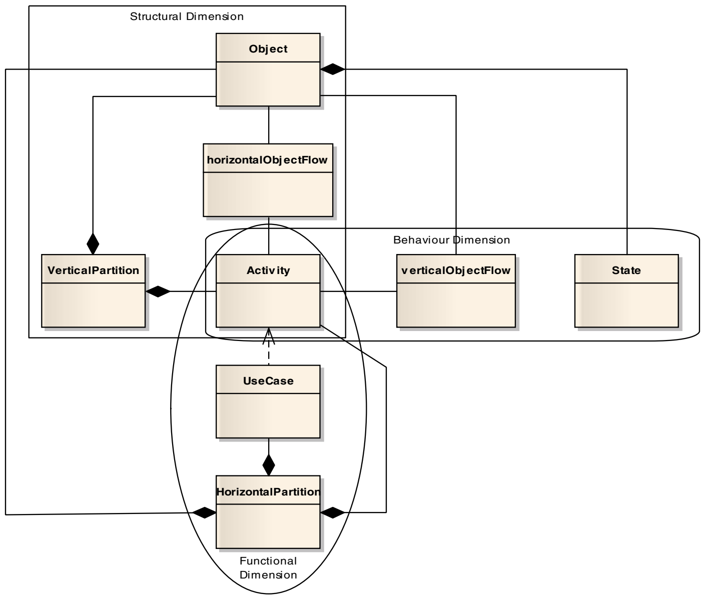
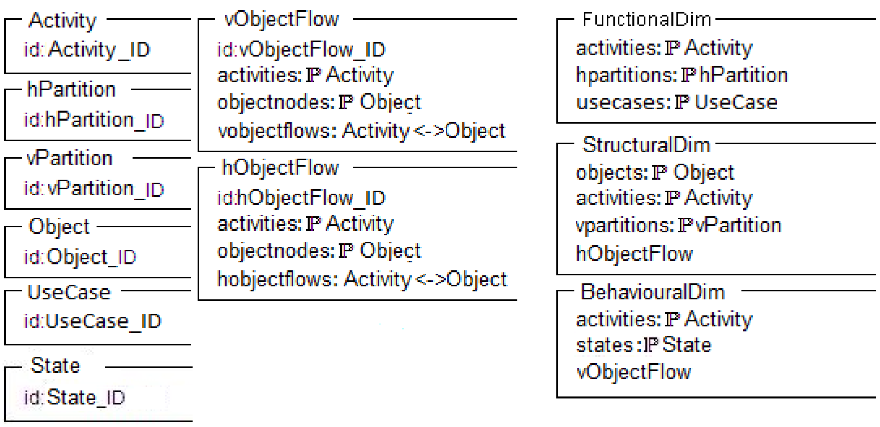
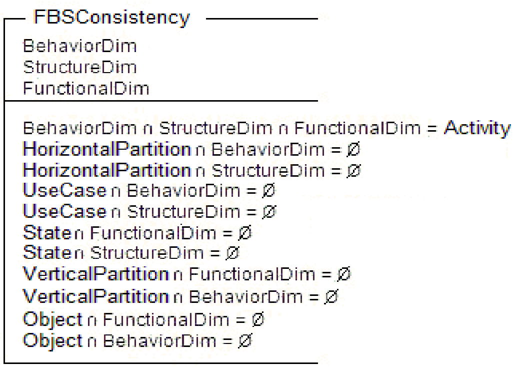


Fig. 9. Metamodel of the FBS activity diagram



```
Activity
  id: Activity_ID

hPartition
  id:hPartition_ID

vPartition
  id: vPartition_ID

Object
  id: Object_ID

UseCase
  id:UseCase_ID

State
  id: State_ID

vObjectFlow
  id:vObjectFlow_ID
  activities: ℙ Activity
  objectnodes: ℙ Object
  vobjectflows: Activity <->Object

hObjectFlow
  id:hObjectFlow_ID
  activities: ℙ Activity
  objectnodes: ℙ Object
  hobjectflows: Activity <->Object

FunctionalDim
  activities: ℙ Activity
  hpartitions: ℙ hPartition
  usecases: ℙ UseCase

StructuralDim
  objects: ℙ Object
  activities: ℙ Activity
  vpartitions: ℙ vPartition
  hObjectFlow

BehaviouralDim
  activities: ℙ Activity
  states : ℙ State
  vObjectFlow
```

Fig. 10. Z-formalisation of the FBS activity diagram



```
FBSConsistency
  BehaviorDim
  StructureDim
  FunctionalDim
  ----------------
  BehaviorDim ∩ StructureDim ∩ FunctionalDim = Activity
  HorizontalPartition ∩ BehaviorDim = ∅
  HorizontalPartition ∩ StructureDim = ∅
  UseCase ∩ BehaviorDim = ∅
  UseCase ∩ StructureDim = ∅
  State ∩ FunctionalDim = ∅
  State ∩ StructureDim = ∅
  VerticalPartition ∩ FunctionalDim = ∅
  VerticalPartition ∩ BehaviorDim = ∅
  Object ∩ FunctionalDim = ∅
  Object ∩ BehaviorDim = ∅
```

Fig. 11. Formalization of consistency of the FBS activity model

4.2 Consistency of the FBS Activity Model

Inconsistencies arise between elements belonging to several models. In Fig. 11. Z-formalization of the consistency of FBS activity diagram is presented. The common element of the three dimensions is *Activity* and others elements are not dependent on each other so the three dimensions do not overlap with each other. Therefore, according to Spanoudakis and Zisman [15], the three dimensions of the FBS activity model are consistent. This property implies that the corresponding UML models (use case diagram, state machine diagram, class diagram) are consistent too. Any change in the *Activity* element is visible in all dimensions of the FBS activity model. The changes of other elements of the FBS activity model do not influence each other. Z-schema *FBSConsistency* (Fig.11) describes the above mentioned rules.

5 Related Work

Different software models describe the same system from different points of view, at different levels of abstraction and granularity, possibly in different notations. They may represent the perspectives and goals of different stakeholders. Usually some inconsistencies between models are arising. Inconsistencies in models reveal design problems. If these problems are detected at the early stages of the design, costs of fixing them are much lower than if they are detected at later stages of software design.

Usually UML models are translated into programming languages. Inconsistent UML model may result in an imprecise code. Inconsistencies highlight conflicts between the views and goals of the stakeholders, indicating those aspects of the system which should be analysed.

The approach to model or describe the system in three dimensions i.e. function, structure and behaviour is widely used. E.g. Goel, Rugaber, and Vattam proposed in [29] the structure, behaviour, and function modeling language. They viewed SBF as a programming language with specified abstract syntax and static semantics. The SBF language captures the expressive power of the programs and provides a basis for interactive construction of SBF models. They also described an interactive model construction tool called SBFAuthor that is based on the abstract syntax and static semantics of the SBF language. The precise specification potentially enables a range of additional automated capabilities such as model checking, model simulation, and interactive guides and critics for model construction. The problems of consistency and completeness of model are not discussed in their paper.

The Integrated Notation for specifying software architecture introduced in [30] also proposes three levels of abstraction i.e.: structure (specified by graphical box and line diagram), behavioural specification using Input/Output Automata and abstract data types (ADT) described by Larch traits. Bastarrica, Ochoa and Rossel claim that starting from the structure, refining it with behavioural details and using abstract data types the software architect can obtain a consistent model. The Integrated Notation does not have elements which are common for several levels so the rules to keep consistency among layers [30] are much more complicated than in the FBS activity model. The Integrated Notation does not meet our preceding definition of satisfactory completeness.

Vondrak presented in [31] the Business Process Studio application based on functional, behavioural, and structural views. He applied the object diagram to structural specification, state diagram to describe behaviour of the system, and dataflow diagram as the functional dimension. In this approach a c*oordination model,* based on functional and object models, is used to show *how* the process will be enacted. The coordination model specifies interactions among objects (active and/or passive) and defines the way all these activities are synchronized based on principles used in Petri Net. The coordination view is the most important because it enables to define the execution order of all activities, including conditions for their potential concurrency. It means that the correct order is defined, as well as sharing of used resources. Each activity can have more than one scenario with the duration time and costs associated to provide necessary information for the analysis. Based on the architecture definition captured in a functional model, the "primitive" activities are accompanied by sub-processes icons that can be refined further into more detailed collaboration models again.

Method that would allow to derive the software architecture of any system based on its analysis model was proposed by Elleuch, Khalfallah, and Ahmed in [32]. For that purpose, they introduce a new layer to the Model Driven Architecture (MDA) that takes into account the software architecture. The analysis model is termed the Architecture Independent Model (AIM), which is compliant to the UML 2.0 metamodel. They consider the software architecture in the Architecture Specific Model (ASM), which complies to the defined architectural meta-model. The mapping of AIM into ASM is conducted by using the both meta-models. In this approach only the dimension of the structure based on the class diagram is used. Authors did not explore the incompleteness or inconsistency in their model ArchMDE.

UML is the notation for software engineering projects and many adequate software systems are built with its use. The incompleteness and the inconsistency allowed by UML are a source for problems in the software development process. An interesting question is to what degree inconsistency and incompleteness in UML designs impact software engineering projects. To answer these questions Lange et al. developed a number of techniques for analyzing UML designs. In [33] they attempt to quantify inconsistency and incompleteness of UML diagrams. In this article the analysis is focused on the four most widely used types of diagrams: class diagrams, state chart diagrams, use case diagrams and message sequence charts. Authors did not take into account the dimensions of the software architecture, but they formed some hypothesis about incompleteness and inconsistency in UML diagrams. They performed a number of experiments based on industrial case studies. From these experiments they observe that quantifying inconsistencies and incompleteness provides insight into the use of UML. Although no reference numbers have been established yet, the absolute number of inconsistencies in UML designs is quite large. They also noticed that the types of inconsistencies appear strongly related to the habits and conventions used by the designers.

6 Conclusions

In this paper we have presented a new Function-Behaviour-Structure activity diagram which has several advantages. Our diagram enables to keep the consistency and satisfactory completeness of the application's model. The FBS activity diagram allows to automatically generate complete workflow applications with no need for any "manual" programming. In addition, we have shown that the UML diagrams mapped from the FBS activity model are consistent.

The practical usage of FSB diagram may be questioned. The presented in Fig. 8 FSB diagram, prepared for six use cases, was not "easy to understand and read". In industrial projects the number of use cases is significantly greater but usually complex models are decomposed into submodels. Such approach is commonly used for UML models and also can be applied for FSB diagram. FSB activity diagrams were successfully applied in several industrial realization of IT systems in Poland.

In the design process UML models are refined and to keep the consistency among them, many complicated techniques are used e.g. [20,34]. Instead, it might be considered, to refine the FBS activity model and consecutively map it to the consistent UML diagrams.

The next step in our work is to develop the tool automatically generating complete workflow applications based on FSB activity diagram.

Acknowledgments. We are very grateful to the reviewers for many valuable remarks.

References

1. OMG Model Driven Architecture, `http://www.omg.org/mda`
2. Unified Modeling Language: Superstructure, version 2.4.1, formal (August 05, 2011), `http://www.omg.org/spec/UML/2.4.1/Infrastructure`
3. Ivers, J., et al.: Documenting Component and Connector Views with UML 2.0, Technical Report CMU/SEI-2004-TR-008, ESC-TR-2004-008. Software Engineering Institute, Carnegie Mellon (2004)
4. Niz, D.: Diagrams and Languages for Model-Based Software Engineering of Embedded Systems: UML and AADL, `http://www.sei.cmu.edu/library/reportspapers.cfm` (retrieved January 25, 2011)
5. Hnatkowska, B., Huzar, Z., Magott, J.: Consistency Checking in UML models. In: 4th Int. Conf. on Information Systems, Modeling ISM 2001 (2001)
6. Business Process Management Initiative, `http://www.bpmi.org`
7. van der Aalst, W.M.P., ter Hofstede, A.H., Weske, M.: Business Process Management: A Survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 1–12. Springer, Heidelberg (2003)
8. Organization for the Advancement of Structured Information Standards, `http://docs.oasis-open.org/wsbpel/2.0`
9. Workflow management Coalition, `http://www.wfmc.org/xpdl.html`
10. Object Management Group, Business Process Model and Notation, `http://www.bpmn.org`
11. Woodcock, J., Davies, J.: Using Z, `http://www.usingz.com/text/online/index.html`
12. Shroff, M., France, R.B.: Towards a formalization of UML class structures in Z. In: Proc. of the 21st Int. Computer Software and Applications Conf., August 11-15, p. 646 (1997)
13. Gero, J.S.: Design prototypes: a knowledge representation schema for design. AI Magazine 11(4), 26–36 (1990)
14. Truyen, F.: The Fast Guide to Model Driven Architecture, The Basics of Model Driven Architecture. Cephas Consulting Corp. (2006)
15. Spanoudakis, G., Zisman, A.: Inconsistency management in software engineering: Survey and open research issues. In: Chang, S.K. (ed.) Handbook of Software Engineering and Knowledge Engineering, vol. 1, pp. 329–380. World Scientific Publishing Co., London (1999)
16. Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., Nuseibeh, B.: Inconsistency Handling in Multi-Perspective Specifications. Transactions on Software Engineering 20(8), 569–578 (1994)

17. Egyed, A.: Instant consistency checking for the UML. In: ICSE, pp. 381–390 (2006)
18. Mens, T., Straeten, R.V.D., Simmonds, J.: A Framework for Managing Consistency of Evolving UML Models. In: Yang, H. (ed.) Software Evolution with UML and XML, ch. 1 (2005)
19. Usman, M., et al.: A Survey of Consistency Checking Techniques for UML Models. In: Advanced Software Engineering & Its Applications. IEEE (2008)
20. Kuzniarz, L., et al. (eds.) Workshop on "Consistency Problems in UML-based Software Development II" 2003, Research Report 2003:06, Blekinge Institute of Technology, San Francisco, USA (2003)
21. Choi, H., Yeom, K.: An Approach to Software Architecture Evaluation with the 4+1 View Model of Architecture. In: Ninth Asia-Pacific Software Engineering Conf., pp. 286–293. IEEE Computer Society (2002)
22. Kennaley, M.: The 3+1 Views of Architecture (in 3D): An Amplification of the 4+1 Viewpoint Framework. In: Seventh Working IEEE/IFIP Conference, pp. 299–302. IEEE Computer Society (2008)
23. Issa, A., Abu Rub, F.A.: Performing Early Feasibility Studies of Software Development Projects Using Business Process Models. In: Proc. of the World Congress on Engineering, WCE 2007, London, UK, July 2-4, vol. I (2007)
24. Dijkman, R.M., Joosten, S.M.: An Algorithm to Derive Use Case Diagrams from Business Process Models. In: 6th Intl. Conf. on Software Engineering and Applications (SEA), pp. 679–684. Acta Press, Anaheim (2002)
25. Odeh, M., Kamm, R.: Bridging the gap between business models and system models. Information and Software Technology 15(45), 1053–1060 (2003)
26. Kruchten, P.: The Rational Unified Process: An Introduction, 3rd edn. Addison-Wesley, Boston (2003)
27. Cabot, J., Gómez, C.: Deriving Operation Contracts from UML Class Diagrams. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MoDELS 2007. LNCS, vol. 4735, pp. 196–210. Springer, Heidelberg (2007)
28. Rasch, H., Wehrheim, H.: Checking Consistency in UML Diagrams: Classes and State Machines. In: Najm, E., Nestmann, U., Stevens, P. (eds.) FMOODS 2003. LNCS, vol. 2884, pp. 229–243. Springer, Heidelberg (2003)
29. Goel, A., Rugaber, S., Vattam, S.: Structure, behavior & function of complex systems: The SBF modelling language. Int. Journal of AI in Engineering Design, Analysis and Manufacturing 23, 23–35 (2009)
30. Bastarrica, M.C., Ochoa, S.F., Rossel, P.O.: Integrated Notation for Software Architecture Specifications. In: SCCC 2004, pp. 26–35 (2004)
31. Vondrak, I.: Business Process Modelling. In: Proc. of the 2007 Conf. on Information Modelling and Knowledge Bases XVIII (2007)
32. Elleuch, N., Khalfallah, A., Ahmed, S.B.: Software Architecture in Model Driven Architecture. In: 3rd Int. Symposium on Computational Intelligence and Intelligent Informatics – ISCIII 2007, Agadir, Morocco, March 28-30, pp. 219–223 (2007)
33. Lange, C., et al.: An Empirical Investigation in Quantifying Inconsistency and Incompleteness of UML Designs. In: Proc. Workshop on Consistency Problems in UML-based Software Development, 6th Int. Conf. on Unified Modelling Language (2003)
34. Wang, S., Jin, L., Jin, C.: Ontology Definition Meta-model based Consistency Checking of UML Models. In: Proceedings of the 10th Int. Conf. on Computer Supported Cooperative Work in Design. IEEE (2006) 1-4244-0165-8/06