Stanisław Jerzy NIEPOSTYN

Warsaw University of Technology

doi:10.15199/48.2015.02.08

The Sufficient Criteria For Consistent Modelling Of The Use Case Realization Diagrams With A New Functional-Structure-Behaviour UML Diagram

***Abstract**. UML activity diagrams are primarily used to visualise scenarios. The verification of activity diagrams consistency is subsequently needed to identify errors in requirements at the early stage of the development process. The consistency verification is difficult due to a semi-formal nature of activity diagrams. We propose to extend the activity diagram to the new Functional-Structure-Behaviour (FSB) UML diagram to enable automatic verification of consistency of scenarios of the visualized use cases. Moreover the FSB UML diagram enables simultaneous modelling of the functionality, of the structure and of the behaviour of the target system model. Thus the proposed Functional-Structure-Behaviour UML activity diagram enables consistent and complete models to be developed from scenarios. Furthermore the FSB UML activity diagram can be used for automatic generation of complete workflow applications without any manual programming.*

***Streszczenie.** Diagramy aktywności UML używane są przede wszystkim do wizualizacji scenariuszy. W celu wyeliminowania błędów w przyszłym systemie niezbędny okazuje się proces weryfikacji tych diagramów UML. Weryfikacja spójności jest jednakże dość złożonym zagadnieniem, gdyż diagramy aktywności nie są formalnym sposobem zapisu wymagań. Proponujemy rozszerzenie diagramów aktywności UML do diagramów nazwanych przez nas diagramami Funkcjonalność-Struktura-Behawioryzm (FSB) UML które umożliwiają automatyzację weryfikacji spójności scenariuszy wizualizowanych przypadków użycia. Co więcej diagram FSB UML umożliwia równoczesne modelowanie funkcjonalności, struktury i zachowania docelowego modelu systemu. Dlatego też zaproponowany diagram aktywności FSB UML umożliwia również opracowywanie kolejnych spójnych i kompletnych modeli na jego podstawie. Ponadto diagram aktywności FSB UML może być wykorzystany do automatyzacji generowania aplikacji typu workflow bez potrzeby ręcznego programowania. **Warunki wystarczające do spójnego modelowania diagramów realizacji przypadku użycia z wykorzystaniem ulepszonego diagramu aktywności FSB UML**.*

Keywords: UML, consistency, sufficient consistency, completeness, FSB UML diagram.
Słowa kluczowe: UML, spójność, wystarczające spójność, kompletność, diagram aktywności FSB UML.

Introduction

In object-oriented software development, the UML [1] has become the standard notation for the software architecture modelling at different stages of the life cycle and at different views of the software system, including the requirements specification. Thus in the majority of projects using UML diagrams [2, 3], 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 elements ([4, 5]). Subsequently activity diagram can be used to verify consistency of other diagrams. This kind of diagram is also used to visualize scenarios and is called “use case realization diagram”. Notably the use case realization diagram has been formally integrated neither with the class diagram nor with the state machine diagram. In this paper we propose to reverse these activities: firstly we prepare the activity diagram, then we derive class, use case and state machine diagrams.

An early consistency check of the use case realization diagram seems crucial for the consistency and completeness of the software architecture, but proves to be difficult due to the informal nature of activity specifications. By the sufficient criteria for consistent modelling use case realization diagram we mean that:

- all flow paths of an activity diagram can be performed;
- a diagram describing the structure aspect of the system can be generated;
- a diagram describing the behaviour aspect of the system can be generated;
- all elements of the activity diagram must be mapped onto generated diagrams.

We propose to extend the activity UML diagram to the Functional-Structure-Behaviour (FSB) UML diagram to enable automatic verification of consistency and completeness of scenarios of the visualized use cases. The aim of the consistency analysis is to validate that all flows are connected. Next we propose to check that each activity and each instance of object in our activity diagram has link with each key element of the subsequent generated UML diagrams.

The object pseudo-code can be used to formalize this problem and to provide a tool to support the analysis. The idea with Z formalization was presented in [6] to keep the sufficient consistency and completeness between class, state machine and use case diagrams based on DCD Diagram. This idea was proved and extended in [7] for class, state machine and use case diagrams based on FSB UML diagram. Here algorithms are presented, because such object pseudo-code may be easily implemented in Java coding tools.

Based on our previous work cited above, this article presents the sufficient criteria for consistent modelling of the use case realization diagrams with a Functional-Structure-Behaviour UML diagram. In order to apply the proposed criteria we provide a semantics for FSB UML diagram with the object pseudo-code. We also improve the already known criteria and introduce new rules for the analysis phase. Our concepts are logically extended from the previous papers based on experience of IT projects publicly procured in Poland within the period 2013 – 2014 to the Ministry of Finance and to the Agency for Restructuring and Modernisation of Agriculture. It is also underlined here that the proposed criteria for consistent modelling enable automatic modelling of IT systems.

The paper is organized as follows: in section 2 the related works on consistency checking of UML diagrams are reviewed, and then completeness and types of inconsistencies are described. In section 3 the rule-based method for consistent modelling of the use case realization diagrams with a Functional-Structure-Behaviour UML diagram is described. The rationale for applying the FSB UML diagram to derive the complete and consistent UML model is given in section 4. Section 5 concludes the paper.

Related Works

Different software models could 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 [8]. Inconsistencies in models reveal design problems. Obviously earlier the problems are detected during the software design, lower is the cost of fixing them.

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

As shown in [9], there are several methods to verify consistency in UML diagrams: meta-model-based method [10], graph-based method [11], scenario-based method, constraint-based methods and knowledge-based methods [12]. We are focusing here on constraint-based methods and on graph-based methods.

Egyed proposed methods for fixing inconsistencies in UML diagrams [13]. Those methods were regarding class, state, object and sequence UML diagrams. Another approach to check consistency of activity diagrams was proposed by Jurack et el. in [14]. In this method the consistency of the activity diagram was validated by checking whether all flow paths could be performed. Shinkawa in his research [15] proposed to generate consistent UML diagrams from the activity diagram based on Coloured Petri Net. A few rules for consistency between activity diagrams and use case diagrams were proposed Ibrahim [16].

Our research focuses attention to consistency of class, state machine, and use case UML diagrams derived from the use case realization diagram. We propose the sufficient criteria for consistent modelling of use case realization diagram to generate consistent class and state machine diagrams. Specifically we propose here to make use of the FSB UML diagram as the use case realization diagram.

Moreover our approach enables to model or to describe the system in three dimensions i.e. function, structure and behaviour. Goel, Rugaber, and Vattam proposed in [17] the structure, behaviour, and function modelling language (hereinafter shortcut SBF) based on the Functional-Structure-Behaviour (FSB) framework introduced by John Gero [18].

Criteria of Consistency

According to Functional-Structure-Behaviour (FSB) framework introduced by John Gero [18] the purpose of the design description is to transfer sufficient information about target system to enable its construction. The description must at least encompass a function, a structure, and 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 [19] described a Model, in major MDA concepts, as a formal specification of the function, structure and behaviour of a system. He claims that any Model must be represented by a combination of UML diagrams. That leads to a situation, in which model inconsistencies may arise (Spanoudakis and Zisman [8]).

In this section we explain in an informal way the model consistency, which we subsequently apply to the modelling of the use case realization diagram. Then we present our concept of the sufficient criteria for consistent modelling UML diagrams which form consistent and complete description of software architecture.

Model Driven Rules for Consistency

To assert that something is consistent we have to declare what it is consistent with. Software models describe each system from different points of view, at different levels of abstraction and granularity, and in different notations. They may represent viewpoints and goals of different stakeholders. Usually inconsistencies between diagrams are arising because some models are overlapping [8]. Inconsistencies reveal design problems. The consistency rules can be found in formal methods. The research on consistency models was outlined by Finkelstein [20]. UML is not a formal language so often UML models are translated into stricter notation. UML consistency analysis goes far beyond checking syntax and semantics; it should also encompass other areas like targeted programming language, modelling methodology, modelled systems, and application and implementation domains.

Many articles which describe UML consistency rules note some sequences of UML diagrams in which these rules work. In Table 1. these sequences are shown in regular expressions, where: P - package diagram, C - class diagram, O - object diagram, U - use case diagram, A - activity diagram, S - state machine diagram, Q - sequence diagram, I - communication diagram. These sequences show that in some IT projects consistency modelling of the software architecture is driving on the consistency rules. In our article we propose next UML diagrams sequence written as regular expressions: UA(C+S+U). First UML use case diagram is the business use case in which we show interaction between actors, then use case realization diagram is placed (activity diagram), and at the end three orthogonal UML diagrams are showed - class, state-machine, and system use case diagrams. The last UML use case diagram is the system use case diagram, in which we show interaction between the actor and the IT system.

Table 1. Sequence of UML diagram driven consistency rules

Author, Year [references]	Sequence of diagrams	Number of consistency rules
Egyed, 2000 [13]	P(CQ+CO+CS)	50
Sapna, 2007 [21]	C(S+U(A+Q))	18
Ibrahim, 2012 [16]	UQC	8
Ha, 2008 [9]	O(Q+A+I)	7
Chanda, 2009 [22]	UAC	4
Shinkawa, 2008 [15]	UAQS	4
Hausmann, 2002 [23]	UAO	3

We believe that the use case realization model (based on the business use case diagram – the second diagram in our proposition) should be created firstly and should be already in a consistent form to define logical supporting rules for creation of the next models. Such approach should prevent cumbersome and continuous searching for inconsistencies during the model construction.

The Sufficient Criteria for Consistent Activity Diagram

In the majority of projects using UML diagrams [2, 3], 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 ([4, 5]). 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 ([4, 5, 24]). It can be noticed that Use Case, Class and State Machine diagrams are orthogonal (Fig. 1), and enable to derive use case realization diagram [25]. A model, which adequately integrates these diagrams, thus enables to keep the consistency and the sufficient completeness of the whole system because these three diagrams do not have common elements. One can interpret the operation of the class (dimension of the structure), the state in State-Chart (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 construct the completely described three-diagram model. We define sufficient completeness as comprising necessary elements (listed above) and at least one element that integrates all these three dimensions of the software architecture.

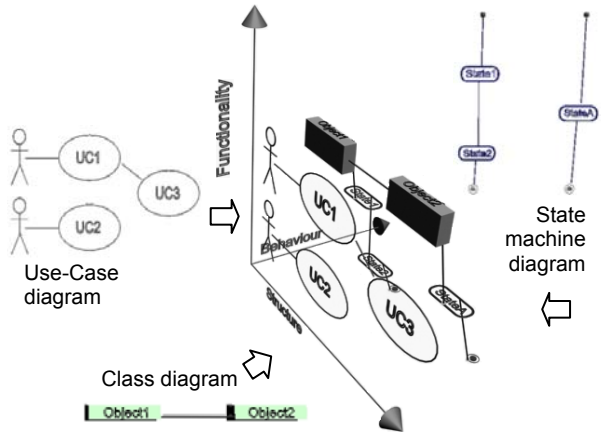

Fig.1. Three dimensions of the software architecture view

Activity diagram based on the UML standard do not satisfy conditions mentioned above. If we add to the activity diagram the structure elements (e.g. instances of class) then we get the new UML diagram with the behaviour, structure, and functional aspects of the system. Therefore such activity diagram gives us a chance to develop sufficiently consistent diagrams.

Besides activity diagram must have all flow paths connected. It means that all flows within an activity diagram can be performed. Activity diagram may be treated as the directed graph with connected vertices. Such graph is said to be connected if every pair of vertices in the graph is connected.

Considering the criteria mentioned above, we define sufficient criteria for consistent Activity Diagram in the following conditions:

- the Activity Diagram is a connected graph;
- the Activity Diagram describes the structure aspect of system;
- the Activity Diagram describes the behaviour aspect of system;
- sufficiently consistent activity diagram enables to create subsequent class and state machine diagrams;
- all elements of the Activity Diagram must be mapped onto generated class and state machine diagrams.

The new Functional-Structure-Behaviour UML Diagram fulfils all above conditions.

Function-Structure-Behaviour UML Diagram

The FSB UML Diagram, based on an activity UML diagram, enables to build a model integrating the three dimensions of software: functional, structural and behavioural. In Fig. 2. there is an example of a routine task in an office modelled with FSB UML diagram. It shows that complete and consistent class, state machine and use case UML diagrams could be derived from the sufficiently consistent FSB UML diagram.

The header of the FSB diagram describes the objects and the first column depicts the Actors. In following columns the activities are presented, each one is performed by an appropriate actor. There are several activities defined: Creating, Checking, Archiving, Approving and Other. These activities have the incoming and outgoing instances of the classes. Fig. 2. 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).

In Fig. 2. the mappings between FSB UML model and UML diagrams are also shown. The FSB UML model is in simple and unambiguous relationships with class diagram (structure), state diagram (behaviour), and use case diagram (functionality) based on consistency rules.

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

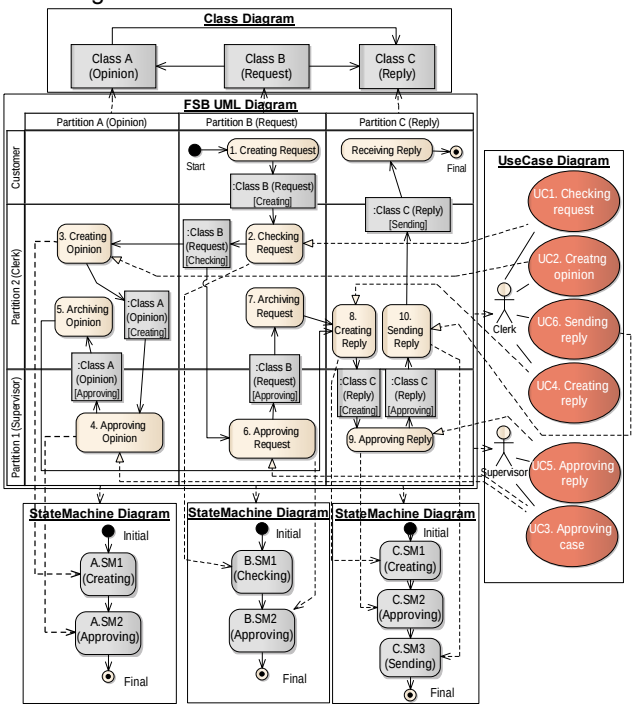

Fig.2. Three dimensions of the software architecture view

A few FSB activities are realized with one use case, and each use case is associated with an Actor in the Use Case diagram. For sake of the readability of Fig. 2, not all dependencies between diagrams are set as visible.

The FSB UML diagram was presented in Object Pseudo-Code in Fig. 3.

```
CLASS fsbDiagram
ATTRIBUTES:
  actors: List<Actor>                 //functional dimension
  objects: List<Object>               //structure dimension
  nodes: List<ControlNodes>           //behaviour dimension
  activities: List<Activity>          //behav. & functional dim.
  instances: List<Instances>          // structure dimension
  controlflows: List<ControlFlow>     // behaviour dimension
  objectflows: List<ObjectFlow>       // structure. dimension
METHODS:
  verifyConnectivity(FSBModel) RETURN result
  checkConsistency(FSBModel) RETURN result
  checkCompleteness(FSBModel) RETURN result
  createCLDiagram(FSBModel, CLDiagram) RETURN result
  createSMDiagram(FSBModel, SMDiagram) RETURN result
  createUCDiagram(FSBModel, UCDiagram) RETURN result
```

Fig.3. Object Pseudo-Code of the FSB UML diagram

Methods showed in the class *fsbDiagram* are described in the next sections.

Sufficient Completeness of the FSB UML Diagram

We described in Section 3 the sufficient criteria for the consistency of modelling. Amongst other criteria our FSB UML diagram must describe the function, structure and behaviour aspects of the system. It means that FSB UML diagram contains the elements, which enable to describe function, structure and behaviour. This property we call the sufficient completeness.

```
METHOD checkCompleteness(fsbDiagram)
  IF fsbDiagram has no start OR has no stop THEN
      RETURN false
  END IF
  IF fsbDiagram has no activity OR has no controlflow THEN
      RETURN false
  END IF
  IF fsbDiagram has no instance OR has no objectflow THEN
      RETURN false
  END IF
  IF fsbDiagram has no object OR has no actor THEN
      RETURN false
  END IF
  RETURN true
```

Fig. 4. Object Pseudo-Code of the completeness checking of the FSB UML diagram

In Fig. 4. the simplified completeness checking method of FSB UML Model is presented. The dimension of functionality describes Actors, and Activities. Objects, Instances and ObjectFlow with Activities represent the dimension of structure, and the dimension of behaviour contains ControlFlow and Activities.

The common elements of the three dimensions are Activities, what could be used to integrate the three dimensions of software architecture in this diagram. Other elements of the FSB UML model fully describe the three dimensions therefore the FSB UML diagram is sufficiently complete.

Sufficient Consistency of the FSB UML Diagram

Other sufficient criteria presented in Section 3 for consistency were related with connectivity of our FSB UML diagram. If we assume that all nodes of the FSB UML diagram are vertices of the graph then if the graph is connected then the FSB UML diagram is also connected. In Fig. 5 the Depth-first search (DFS) algorithm for searching graph paths is shown. The original algorithm was extended with eliminating duplicated paths, but with vertex and edges covering. This algorithm is recursive. If there is lacking start or end node or the graph is not connected.

```
METHOD verifyConnectivity(fsbDiagram)
  IF current vertex in scenario is the last vertex THEN
      Add the main flow to scenarios list
      RETURN 0
  END IF
  Add current vertex to the unique vertices list
  Search for next vertex connected with current vertex
  IF no new next vertex THEN RETURN 0 END IF
  FOR founded vertices
      Push founded vertex to the scenario
      CALL verifyConnectivity METHOD WITH
                                    founded vertex
      IF result no 0 THEN RETURN result END IF
      Pop founded vertex from the scenario
  END FOR
  RETURN 0
```

Fig. 5. Object Pseudo-Code of the sufficient consistency of the FSB UML diagram

Our criterion of the connectivity is similar to the proposition of Jurack [14]. In his method the consistency of the activity diagram was validated by checking whether all flow paths could be applicable. This condition is based on the graph transformation.

The latter sufficient criteria for consistency showed in Section 3 was related to mapping all elements of the activity diagram onto subsequently generated class, use case and state machine diagrams. Because inconsistencies arise between elements belonging to several models therefore the best method to avoid these inconsistencies is to create the subsequent diagrams based on the one consistent diagram. This property implies that the corresponding UML models (use case diagram, state machine diagram, and class diagram) are consistent too. Any change in the Activity element is visible in all dimensions of the FSB activity model. The changes of other elements of the FSB UML diagram do not influence each other. Below we proposed adequate methods in the object pseudo-code.

```
  METHOD createSMDiagram(smDiagram)
  FOR founded instance
      Create UML State WITH Name of instance state
      Relate UML State WITH Class
      Relate UML State WITH other UML States Linked by
                                          Activities
  END FOR
  RETURN
```

Fig. 6. Object Pseudo-Code of the method to create state machine diagram from the FSB UML diagram

```
METHOD createCLDiagram(CLDiagram clDiagram)
  FOR founded UML Instance
      Create UML Class WITH Name of instance
      Relate UML Class WITH founded instance
  END FOR
  RETURN
```

Fig. 7. Object Pseudo-Code of the method to create class diagram from the FSB UML diagram

```
METHOD createUCDiagram(ucDiagram)
  FOR founded UML horizontalPartition
      Create UML Actor WITH Name of horizontalPartition
      Relate UML Actor WITH horizontalPartition
  END FOR
  FOR founded activity
      Create UML UseCase WITH Name of activity
      Relate UML UseCase WITH
            horizontalPartition contains founded activity
  END FOR
  RETURN
```

Fig. 8. Object Pseudo-Code of the method to create use case diagram from the FSB UML diagram

Conclusions

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

The practical usage of FSB UML diagram may be questioned. The FSB UML diagram presented in Fig. 2, 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 sub-models. Such approach is commonly used for UML models and also can be applied for FSB UML diagram. FSB UML diagrams have been already successfully applied in several industrial realizations of IT systems in Poland.

In the design process UML models are usually refined and to keep the consistency among them, many complicated techniques are used e.g. [24, 25]. Alternatively it might be considered to refine only the FSB UML diagram and then consecutively map it onto the consistent UML diagrams.

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

REFERENCES

[1] Unified Modeling Language: Superstructure, version 2.4.1, formal/2011-08-05, http://www.omg.org/spec/UML/2.4.1/Infrastructure

[2] Choi, H., Yeom, K.: An Approach to Software Architecture Evaluation with the 4+1 View Model of Architecture. In: Ninth Asia-Pacific Software Engineering Conference, pp. 286—293. IEEE Computer Society, 2002.

[3] 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.

[4] Issa, A., Abu Rub, F.A.: Performing Early Feasibility Studies of Software Development Projects Using Business Process Models, Proceedings of the World Congress on Engineering 2007 Vol I WCE 2007, July 2 - 4, 2007, London, U.K.

[5] Dijkman, R.M., Joosten, S.M.: An Algorithm to Derive Use Case Diagrams from Business Process Models, 6th International Conference on Software Engineering and Applications (SEA), Anaheim, CA, USA, Acta Press, pp. 679-684, 2002.

[6] Bluemke, I., Niepostyn, S.J., From Three Dimensional Document Circulation Diagram into UML Diagrams, w: Emerging Trends in Computing, Informatics, Systems Sciences, and Engineering / Sobh Tarek, Elleithy Khaled (red.), Lecture Notes in Electrical Engineering, vol. 151, 2013, Springer, ISBN 978-1-4614-3557-0, ss. 319-329.

[7] Niepostyn, S.J., Bluemke I., The Function-Behaviour-Structure Diagram for Modelling Workflow of Information Systems, w: Advanced Information Systems Engineering Workshops / Bajec Marko, Eder Johann (red.), Lecture Notes in Business Information Processing, vol. 112, 2012, Springer, ISBN 978-3-642-31068-3, ss. 425-439

[8] Spanoudakis, G., Zisman, A.: Inconsistency management in software engineering: Survey and open research issues. In S. K. Chang (Ed.), Handbook of software engineering and knowledge engineering, 1, 329-380. London: World Scientific Publishing Co, 1999.

[9] Ha, I., Kang, B.: Cross Checking Rules to Improve Consistency between UML Static Diagram and Dynamic Diagram. In: Fyfe, C., Kim, D., Lee, S.Y., Yin, H. (eds.) IDEAL 2008. LNCS, vol. 5326, pp. 436–443. Springer, Heidelberg (2008)

[10] Paige, R.F., Brooke, P.J.: Metamodel-Based Model Conformance and Multi-View Consistency Checking, ACM Transactions on Software Engineering and Methodology, Volume 16 Issue 3, July 2007

[11] Shuzhen, Y., Shatz, S.M.: Consistency Checking of UML Dynamic Models Based on Petri Net Techniques. In: Gelbukh, A., Guerra, S.S. (eds.) Proc. of the 15th International Conference on Computing (CIC 2006), pp. 289–297. IEEE Computer Society, Washington (2006)

[12] Wang, Z., He, H., Chen, L., Zhang, Y.: Ontology based semantics checking for UML activity model. Information Technology Journal. 11, 3, 301-306

[13] Egyed, A.F.: Heterogeneous View Integration and its Automation, PhD diss., University of Southern California, 2000

[14] Jurack, S., Lambers, L., Mehner, K., Taentzer, G.: Sufficient Criteria for Consistent Behavior Modeling with Refined Activity Diagrams, Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science Volume 5301, 2008, pp 341-355

[15] Shinkawa, Y.: Inter-Model Consistency in UML Based on CPN Formalism, in: 13th Asia Pacific Software Engineering Conference (APSEC '06) 2006, pp. 414-418

[16] Ibrahim, N., Ibrahim, R., Saringat, M. Z., Mansor, D., Herawan, T.: Definition of Consistency Rules between UML Use Case and Activity Diagram, in T.h. Kim, H. Adeli, R. J. Robles & M. Balitanas (Eds.), Ubiquitous Computing and Multimedia Applications, ed., Communication of Computer and Information Sciences vol. 151, Springer Berlin / Heidelberg, Daejeon, Korea, 2011

[17] Goel, A., Rugaber, S., Vattam, S.: Structure, behavior & function of complex systems: The SBF modeling language. International Journal of AI in Engineering Design, Analysis and Manufacturing, 23, 23–35 (2009)

[18] Gero, J.S., Kannengiesser, N.: The Situated Function-Behavior-Structure Framework, Design Studies, vol. 25, no. 4, 2004, pp. 373–391.

[19] Truyen, F.: The Fast Guide to Model Driven Architecture, The Basics of Model Driven Architecture, Cephas Consulting Corp, January 2006.

[20] Finkelstein, A.C.W., Gabbay, D., Hunter, A., Kramer, J., Nuseibeh, B.: Inconsistency Handling in Multi-Perspective Specifications. Transactions on Software Engineering, 20(8): 569-578, August 1994; IEEE Computer Society Press.

[21] Sapna, P. G., Mohanty, H., Ensuring Consistency in Relational Repository of UML Models, in: 10th *International Conference on Information Technology* (ICIT 2007), 2007, pp. 217-222.

[22] Chanda, J., Kanjilal, A., Sengupta, S., Bhattacharya, S., Traceability of Requirements and Consistency Verification of UML UseCase, Activity and Class diagram: A Formal Approach, in: *International Conference on Methods and Models in Computer Science* 2009 (ICM2CS), 2009, pp. 1-4.

[23] Hausmann, J., Heckel, R., Taentzer, G., Detection of Conflicting Functional Requirements in a Use Case-Driven Approach. In: *Proc. of Int. Conference on Software Engineering* 2002, Orlando, USA (2002).

[24] Kruchten, P., The Rational Unified Process: An Introduction, 3 ed., Boston: Addison-Wesley, 2003

[25] Usman, M., Nadeem, A., Taihoon, Kim, Eunsuk, Cho, A Survey of Consistency Checking Techniques for UML Models, in *Advanced Software Engineering & Its Applications*, 2008, IEEE

***Authors**: mgr inż. Stanisław Jerzy Niepostyn, Instytut Informatyki Politechniki Warszawskiej, ul. Nowowiejska 15/19, 00-665 Warszawa, E-mail: s.niepostyn@ii.pw.edu.pl.*