UML MODELS CONSISTENCY MANAGEMENT: GUIDELINES FOR SOFTWARE QUALITY MANAGER

Raja Sehrab Bashir (Corresponding Author)
Department of Software Engineering, University of Malaya,
Kuala Lumpur, 50603, Malaysia
sehrabraja@siswa.um.edu.my

Sai Peck Lee (Corresponding Author)
Department of Software Engineering, University of Malaya,
Kuala Lumpur, 50603, Malaysia
saipeck@um.edu.my
Saif Ur Rehman Khan
Department of Software Engineering, University of Malaya,
Kuala Lumpur, 50603, Malaysia
saif_rehman@siswa.um.edu.my

Shahid Farid
Department of Computer Science, Bahauddin Zakariya University,
Multan, Pakistan
shahidfarid@bzu.edu.pk
Victor Chang
International Business School Suzhou, Xi'an Jiaotong Liverpool University,
Suzhou, China
ic.victor.chang@gmail.com

Abstract—Unified Modeling Language (UML) has become the de-facto standard to design today's large-size object-oriented systems. However, focusing on multiple UML diagrams is a main cause of breaching the consistency problem, which ultimately reduces the overall software model's quality. Consistency management techniques are widely used to ensure the model consistency by correct model-to-model and model-to-code transformation. Consistency management becomes a promising area of research especially for model-driven architecture. In this paper, we extensively review UML consistency management techniques. The proposed techniques have been classified based on the parameters identified from the research literature. Moreover, we performed a qualitative comparison of consistency management techniques in order to identify current research trends, challenges and research gaps in this field of study. Based on the results, we concluded that researchers have not provided more attention on exploring inter-model and semantic consistency problems. Furthermore, state-of-the-art consistency management techniques mostly focus only on three UML diagrams (i.e., class, sequence and state chart) and the remaining UML diagrams have been overlooked. Consequently, due to this incomplete body of knowledge, researchers are unable to take full advantage of overlooked UML diagrams, which may be otherwise useful to handle the consistency management challenge in an efficient manner.

Keywords: UML model consistency, UML model transformation

1. Introduction

Software maintenance is the most important phase of a software development life cycle as maintenance consumes almost 40-80% of the total software development cost (Fernández-Sáez, Genero, Caivano, & Chaudron, 2016). Moreover, 60% of the total maintenance cost is spent on enhancing the existing functionality of the software. Therefore, it is of vital importance to prepare appropriate software artifacts at each phase to reduce the maintenance cost. Maintenance cost can be reduced by improving the comprehension of a software system. Comprehension of a software system consumes about 50% of time of the maintenance phase. Different modeling languages have emerged

to help represent the software system in a graphical notations and increase the system comprehension (Dzidek, Arisholm, & Briand, 2008).

Model-Driven Software Engineering (MDSE) is a discipline aimed at promoting the models for software development and maintenance. Model-based representation of a software system provides better understanding about underlying concepts(e.g., classes, methods, aggregation, association, multiplicity, and operations) (Misbhauddin & Alshayeb, 2015). Moreover, models provide a complete and detailed specification to better communicate the structure and behavior of a software system under development (Brambilla, Cabot, & Wimmer, 2012). Unified Modeling Language (UML) is the most popular modeling language and become the de-facto standard to design today's large object-oriented systems. UML offers 14 different diagrams to represent the structure and behavior of a software system (Ahmad, Gani, Hamid, Shiraz, et al., 2015; Holt, 2004). The structural diagrams represent the static aspect of a software, whereas behavioral diagrams are used to represent the dynamic aspects of a under development software.

UML has gained popularity in the last 15 years due to its multi-view support. A good quality model requires associated UML diagrams, which should be consistent with each other since they represent the same system but from different viewpoints. Change in one diagram (e.g., due to change in user requirements) can ultimately affect the underlying model, and as a result, it may cause unwanted changes in other associated UML diagrams of the same model. The term model consistency is defined by Spanoudakis et al., as "*a state in which two or more elements, which overlap in different models of the same system, have a satisfactory joint description*" (Spanoudakis & Zisman, 2001). For example, if a method (mi) is used in a sequence diagram and mi is also mentioned in the class diagram, then we can conclude that both sequence and class diagrams are consistent with respect to mi (Object, 2007). UML model consistency is heavily affected by the number of UML views that may cause a number of errors in the developed system (Muskens, Bril, & Chaudron, 2005). UML model consistency ensures two types of correct transformations: (i) model-to-model: one UML diagram is transformed into other diagrams (e.g., UML sequence diagrams are transformed into the corresponding UML interaction diagrams), and (ii) model-to-code: code is generated automatically from the UML diagram (Lucas, Molina, & Toval, 2009).

Model consistency management includes a number of activities for finding and managing the consistency problems. In order to identify the consistency problems, the software engineer focuses on finding the ambiguities between UML models. In UML models, consistency problems can be occurred in the naming convention, integration of two models, or interaction of models, etc. (Spanoudakis & Zisman, 2001). Figure 1 depicts the abstract overview of model consistency management stages. At the top level, it contains two main stages including model transformation and consistency management (Figure 1). First of all, the developer edits the original source model based on the users' given new requirements. After that, the developer performs model analysis (in the consistency management stage), where two possible scenarios can be occurred: (i) models are consistent, and (ii) consistency problems are identified. In the case of consistent models, the original model will be updated. On the other hand, the consistency problem is analyzed to determine the type of consistency (i.e. syntactic, semantic, horizontal and vertical).

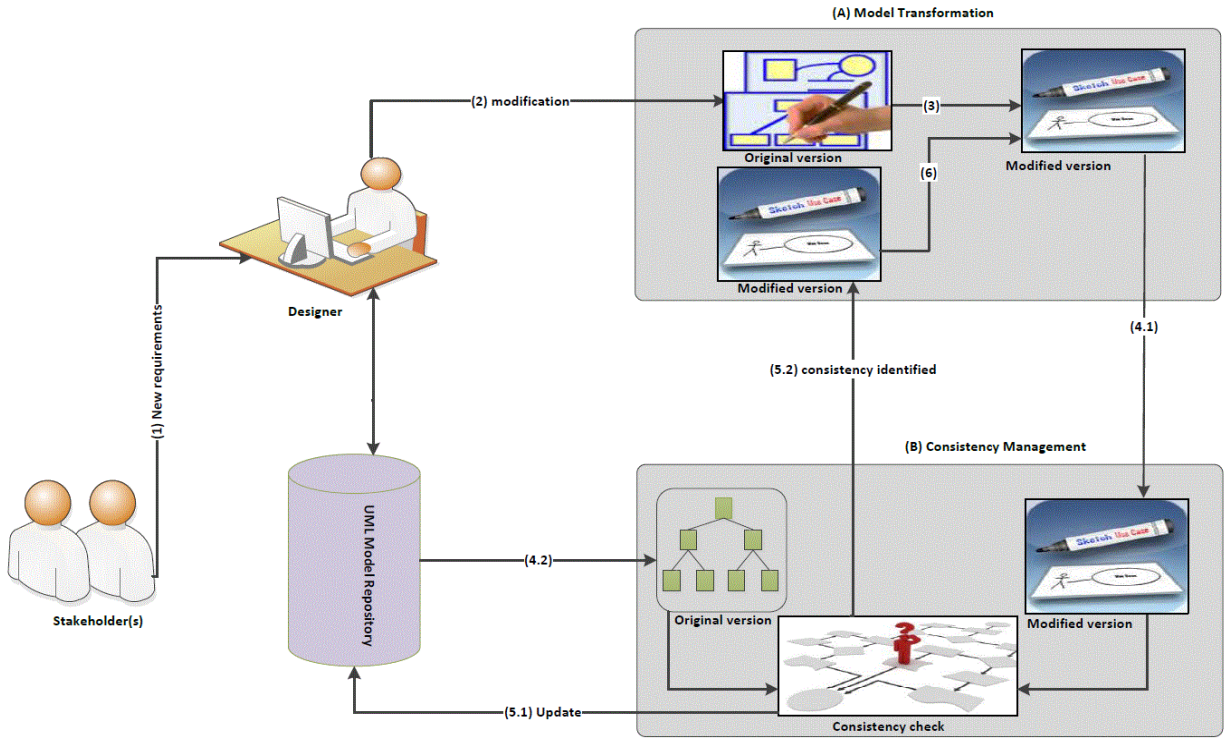


Figure 1: Abstract representation of model consistency management

After identifying the consistency problems, forwarded to the model transformation stage. Subsequently, perform most suitable transformation based on the developer feedback and type of the consistency problem. Finally, consistency between the transformed model and the original model is checked to ensure consistency.

The Model-Driven Architecture (MDA) has been attracting more attention for the last 15 years. The main objective of MDA is to promote system modeling using UML diagrams. In the scope of MDA, consistency management becomes one of the most important research issues, especially when a software system evolves. In this regard, an extensive amount of research has been conducted for proposing new techniques to improve model quality and also maintain consistency between models. However, current state-of-the-art consistency management techniques lack in exploring and analyzing consistency problems in a complete manner. The existing studies mainly focused on a single view (i.e., structural or behavioral) and horizontal (e.g., intra model) consistency management problems. Moreover, A few studies considered multi-aspects of the system during model transformation. However, prior work overlooked vertical (e.g., inter model) and semantics model consistency problems. The observations and contradictions indicate that the current research on consistency management needs more attention on vertical and semantic model consistency problems. Hence, new approaches need to be developed to support enhancement in UML model with new concepts and ideas. However, new approaches cannot be developed unless the deficiencies in current approaches are analyzed, compared, and research gaps are identified. Motivated by this, we conducted an extensive survey of state-of-the-art approaches focusing on UML model consistency management.

We nevertheless acknowledge that prior work provides an overview of the consistency management techniques. There are two main surveys published in this field of study (Ahmad, Gani, Hamid, Xia, & Shiraz, 2015; Lucas et al., 2009; Spanoudakis & Zisman, 2001). Spanoudakis and Zisman (2001) conducted a first ever survey that focuses on consistency management. The survey broadly focused on consistency management activities and their limitations. Similarly, Lucas et al. (2009) performed a review that covers only consistency type and supporting diagrams. To the best of our knowledge, there is no work that provides a complete broader overview of UML model consistency management. Therefore, we categorized the current state-of-the-art techniques based on the technique types, inconsistencies, view support, diagram support and versions. Moreover, we classify each parameter of taxonomy into

further attributes. Furthermore, we also performed a qualitative analysis of existing UML consistency management techniques.

The main objective of this paper is to analyze, evaluate, and compare current state-of-the-art inconsistency identification and removal techniques. In order to achieve these objectives, we formulated the following research questions:

RQ1: *What is the abstract categorization of model consistency management*?
Answering RQ1 will help researchers to identify the main taxonomical parameters which highlight related issues and underlying concepts of model level consistency management.
RQ2: *What are the main techniques used to manage consistency in UML modeling*?
Answering RQ2 will help software engineers and researchers to find the existing consistency management approaches that have been proposed to improve the system quality. Moreover, it will help the researcher to identify the commonalities and differences between relevant approaches.

RQ3: *What are the types of UML diagrams that have been used to manage consistency in each approach*?
Answering RQ3 will help researchers to identify the UML diagrams that come to be more focused of research.

RQ4: *What are the types of consistency problems that have been identified and resolved in each approach*?
Answering RQ4 will help software engineers and researchers to identify the overlooked consistency problems at the model level.

RQ5: *What are the types of UML diagrams views that have been used in state-of-the-art consistency management approaches?*
Answering RQ5 will help researchers to identify the UML diagram views (e.g., structural and behavioral) that provide support to identify and resolve consistency problems at the model level.

The major contributions of this work are:

- An extensive literature review on UML consistency management to highlight the current research trends in this field of study.
- Presenting a thematic taxonomy of existing state-of-the-art UML consistency management based on a set of parameters extracted from the literature.
- Analytically comparing current state-of-the-art consistency management techniques including their commonalities and differences.
- Outlining a number of potential research issues in this field of study.

1.1. *Research Methodology*

1.1.1. Source Information

To obtain the most relevant studies, we have widely explored various digital libraries. The main aim of selecting digital libraries was to ensure that we do not miss any relevant study. The following are the selected digital libraries that have been used for search.

- Google Scholar (https://www.scholar.google.com)
- ACM Digital Library (https://www.portal.acm.org)
- IEEE eXplore (https://ieeexplore.ieee.org)
- ScienceDirect – Elsevier (https://www.sciencedirect.com)
- Springer (https://www.springer.com/)
- Web of Science (https://www.isiknowledge.com)

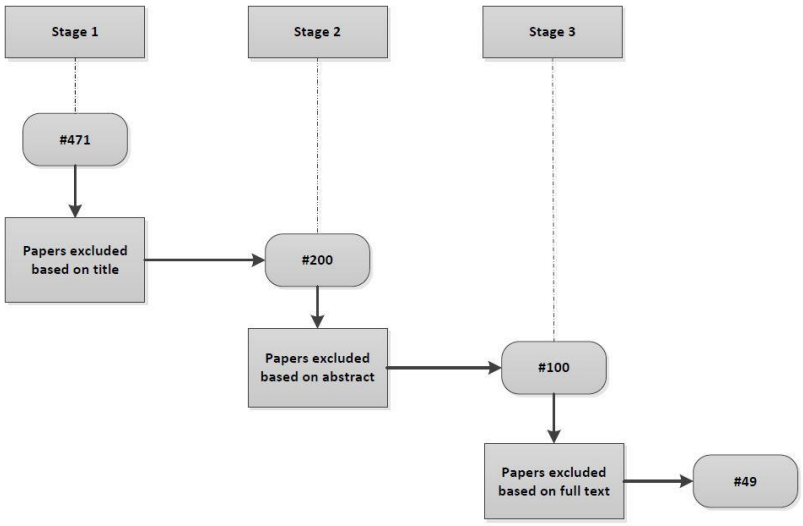


Figure 2: Flow chart of search strategy

1.1.2. Search Criteria

The search criteria were selected by formulating a search query and picking important terms, keywords based on our research objectives. We used (AND) and (OR) logical operators to formulate the search query. Note that we refined the query based on the searching facility and conditions provided by the selected digital libraries. For example [(model) AND ((consistency) OR (inconsistency)) OR (model) AND ((consistency) AND (management))]

Based on the suggestions of Kitchenham et al. (2009),we only considered the papers written in English (Kitchenham et al., 2009). The earliest selected primary study was published in 2000. Therefore, we set the start year to 2000 in order to confirm that related studies within this area of research would be included, and the last date was set to 2016. The initial search resulted in collecting 471 potential papers.

1.1.3. Inclusion and Exclusion Criteria

In this section, we present the inclusion/exclusion criteria used for papers selection. Based on the formulated objectives of this survey, we formulated the following inclusion criteria:

- The papers targeting the focus of our survey including model consistency, consistency types, and approaches were selected for initial evaluation.
- The papers written in English were selected.
- The publication period starting from the year 2000 to year 2016 was considered.

Similarly, we formulated and adopted the following exclusion criteria in order to exclude irrelevant papers:

- The papers, which do not cover model consistency were excluded.
- The papers do not written in English languages were excluded.
- Duplicate papers were manually excluded to avoid reporting similar results.
- Papers published pre 2000 were excluded.
- Mapping studies, systematic literature reviews and surveys were excluded since they lack in new approach.
- Due to the lack of technical details, the extended abstracts and short papers were also excluded.

1.1.4. Search Strategy

In order to consider most relevant papers, three researcher were involved in a three-stage selection process as depicted in Figure 2.

Initially, we obtained 471 papers from all mentioned digital libraries. Then we excluded irrelevant and duplicated papers based on the title. After this stage, we considered 200 papers for further evaluation. In the second stage, we analyzed the abstracts of all 200 papers according to the formulated objectives. We observed that various papers were

not according to objectives and were excluded from the list. The most relevant 100 papers were selected. In the final stage, the full contents of the 100 papers were read. Finally, the researchers selected 49 papers, which satisfy the defined objectives and inclusion/exclusion criteria. Our classification is based on final selected papers.

This paper is structured as follows: In Section 2, we present and describe the taxonomy of the UML model consistency management, consistency problem classification, UML consistency management techniques, View support, UML diagram support and UML version. Section 3 provides analyses and discussion on the current state-of-the-art consistency management techniques. Moreover, it identifies main challenges in this field of study. Section 4 concludes the study and outlines a number of potential research directions.

2. Thematic Taxonomy of UML Consistency Management

This section presents a thematic taxonomy for the classification of consistency management techniques. We classify the UML consistency management at the model level based on the identified parameters including *consistency problem classification*, *UML diagram type*, *view type*, techniques, and *UML version type*, in existing state-of-the-art as depicted in Figure 3. The first parameter of the taxonomy is *consistency problem classification* which categories into three main inconsistency classifications: (i) Engels (Engels, Küster, Heckel, & Groenewegen, 2001), (ii) Simons (Simmonds, 2003), and (iii) Driss. Engels categorized UML inconsistencies into syntactical, semantic, vertical and horizontal inconsistencies. In contrast, Simons divided UML inconsistencies into syntactical, semantic, instance level and specification level. Driss provided hybrid classification by combining both Simons and Engels classifications(Driss Allaki 2015).

The parameter *UML diagram type* represents 14 different diagrams to represent the system. There are two associated attributes of UML diagram type: (i) structural diagrams, and (ii) behavioral diagrams. The structural type contains seven different UML diagrams including class, object, deployment, component, composite structural, package, and profiling diagrams. In contrast, the behavioral type has seven UML diagrams that represent the behavior of the software system including sequence, state chart, activity, use case, communication, timing, and interaction diagram. The parameter *view type* is used to identify which aspect of the UML model has been considered in existing state-of-the-art. There are two associated attributes of view support: (i) single, and (ii) multi view. Single-view attribute represents only one aspect of the system (e.g., structural or behavioral), while multi-view attribute contains more than two aspects of the system together to identify and resolve the consistency problems.

UML consistency management *techniques* parameter represents techniques that provide support in consistency violation identification and resolution to ensure that the model is syntactically and semantically correct. The associated attributes of UML consistency management techniques are: (i) formal, and (ii) non-formal based techniques. Formal-based UML consistency management techniques rely on logics, algebraic formulation and state transitions. On the other hand, non-formal based techniques are based on some defined constraints or rules.

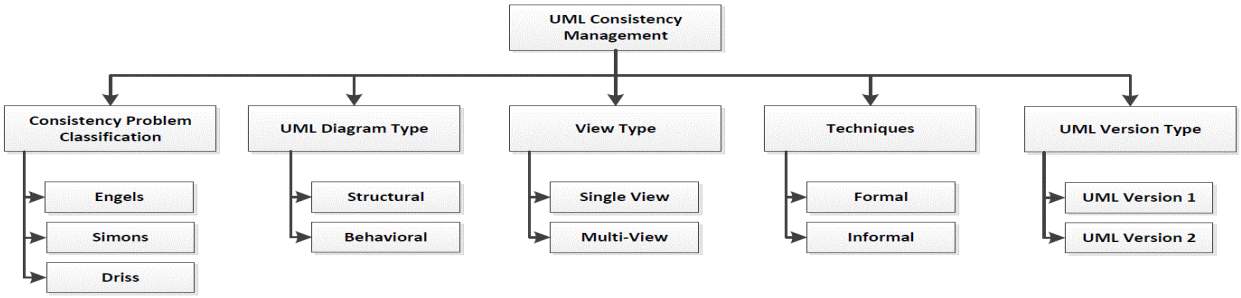


Figure 3: Taxonomy of UML model consistency management

The last parameter is *UML version type* that represents the version of the UML used in existing state-of-the-art techniques to validate the proposed work.

2.1. Model Consistency problems classification

Over the past few years, UML consistency management has become the hot issue due to an emergence of model-driven architecture (MDA). Consistency problems arise due to multiple view and diagrams in UML. These diagrams are used to represent the various aspects of the software system. Moreover, consistency problems also arise due to the iterative nature of the software system development. Several techniques have been proposed for checking consistency problems in UML based model. However, few studies have classified the consistency problems. The most famous and prominent classifications of consistency problems are known as Engels', Simonds' and Driss classifications as shown in Figure4.

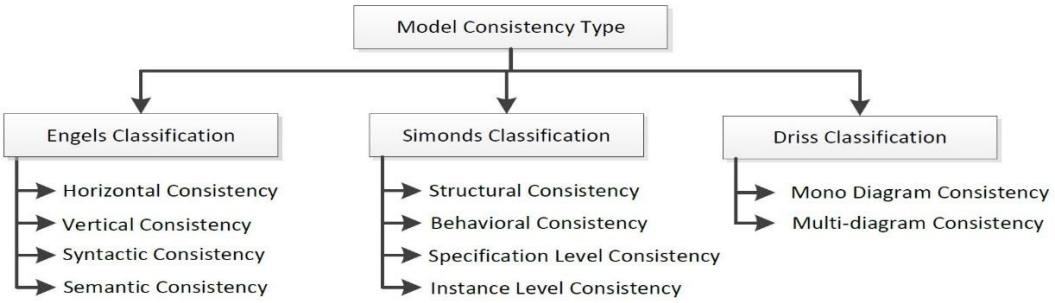


Figure4: UML model inconsistencies classification

- Engels et al., (Engels et al., 2001) proposed a classification of model consistency problems into two orthogonal dimensions. The first dimension of Engels classification is based on vertical and horizontal consistency problems notations. The second dimension is based on syntactic and semantic consistency problems. Horizontal model consistency is also known as the intra-model consistency problem, occurs when consistency is not validated between different UML models that conform to the same meta-model (Hilken, Seiter, Wille, Kühne, & Drechsler, 2014). For example, the class diagrams and their associated state diagram should be consistent. An operation must not be removed from a class diagram while an object of this class in the associated state chart diagram still relies on this operation to handle a message passing. Vertical model consistency or inter-model consistency occurs when the consistency is not validated between models with different meta-models or abstraction level (Sapna & Mohanty, 2007). Vertical consistency is the result of model refinements. Syntactic consistency occurs when the user define UML model that does not conform to its abstract syntax. Abstract syntax is a set of class diagrams defined in the meta-model. The last consistency problem belonging to Engels category is semantic consistency, which is formed when the behavior of the model is not semantically compactible (Banerjee et al., 2012). Semantic consistency problem is difficult to identify in UML, due to the absence of formal semantics in UML. UML semantics is defined using natural and OCL languages.
- Simonds (Simmonds, 2003)proposed two dimensions of the model inconsistencies based on conceptual aspects. The first dimension represents the structural and behavioral inconsistencies. The structural consistency problem arises in the class diagram, which represents the static structure of the system while behavioral consistency is due to non-consistent behavior of the specification of the system. It is related to sequence and state diagram and represents the dynamic behavior of the system. On the other hand, the second dimension shows the abstraction level such as specification level and instance level. Specification level contains the elements of the class diagram such as classes and associations. Instance level contains the elements of the sequence and state chart diagrams such as objects, links and events.
- Driss Allaki et al.,(Driss Allaki 2015) proposed a new classification of the model consistency problems. In the proposed classification, they combined both Engels and Simonds categorization. The authors proposed single-diagram and multi-diagram consistency problems. Figure 5 and 6 represent the single and multi-view UML inconsistency categorizations respectively.

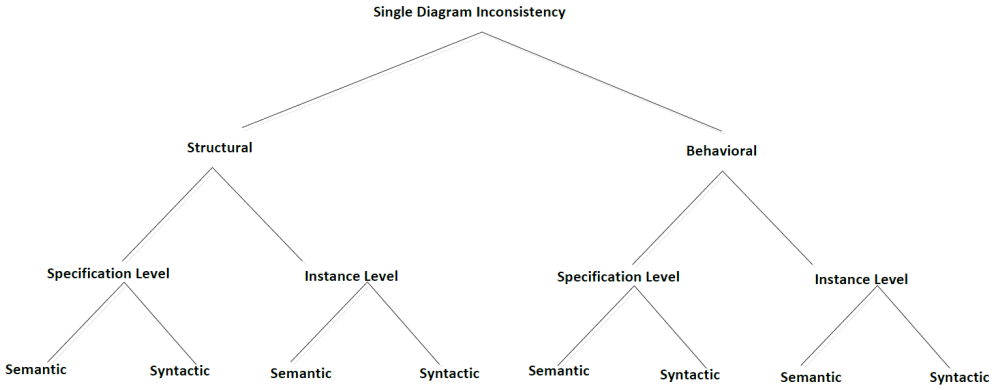


Figure 5 Consistency type for single diagram

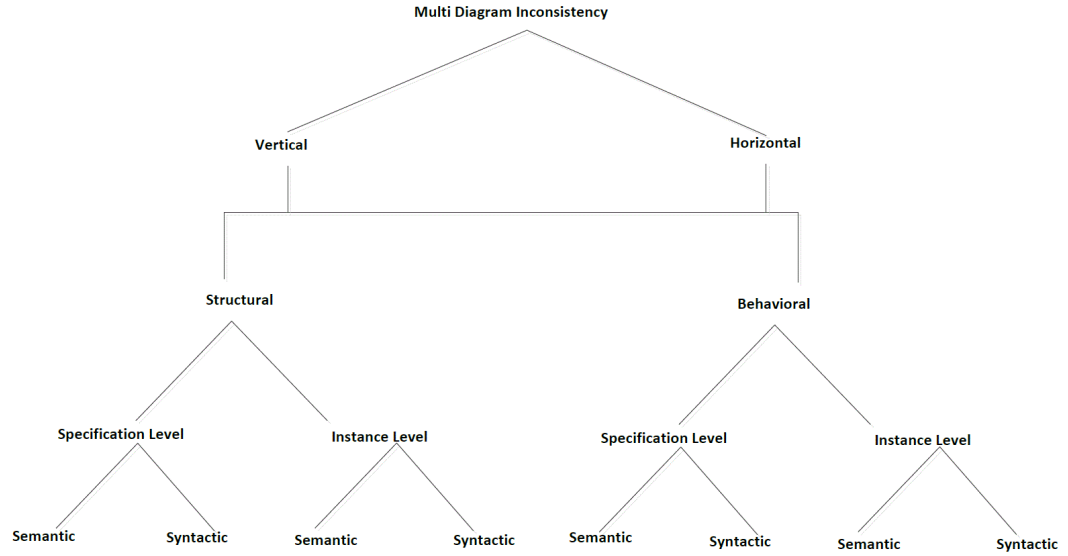


Figure 6: Consistency types for multi-diagrams

2.2. UML diagram and view

Model-Driven Architecture (MDA) is a field that promotes the use of models at different levels of abstraction for developing, maintaining and evolving a software system. These models are often used as sketches for communication and system documentation, and as blueprints to provide a complete and detailed specification of the system (OMG, 2011; Shuja, Gani, Shamshirband, Ahmad, & Bilal, 2016). UML has become standard for a wide range of application domains due to its diversity and multi-view approach to represent a system. UML contains models and views that can represent various aspects (e.g., structural and behavioral) of a system in MDA. Object Management Group (OMG) defines 14 different diagrams as part of its UML 2 specifications (Misbhauddin & Alshayeb, 2015). These 14 diagrams can be categorized into structural and behavioral diagrams as shown in Figure 7. The structural diagram represents the things or objects in a system being modeled. The following are seven structural UML diagrams:

- *Class diagram* is the basic building block of Object-Oriented system. Class diagrams consist of classes, attributes, methods and relationships between these classes.
- *Package diagram* represents the dependencies between different packages.

- *Component diagram* is used when a system has large and complex components. It shows structural relationships between the components of a system. The communication between components is achieved through interfaces in a component diagram.
- *Deployment diagram* represents the hardware and associated software in a system. It is useful when software is deployed on multiple machines having different configuration.

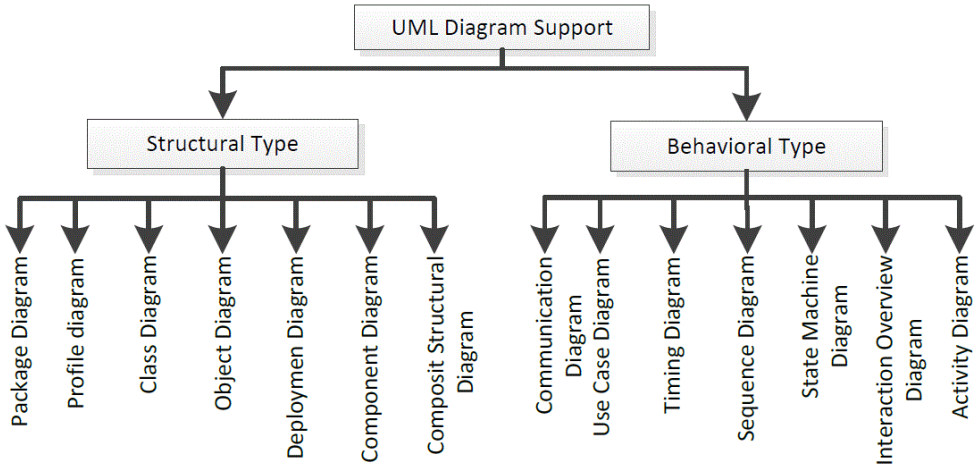


Figure 7: UML diagram support

- *Object diagram* is also known as the instance diagram. Object diagram represents the relationships between objects of the system like a class diagram. The key difference is that object diagram uses real data in objects and shows how the real system looks at a given time.
- *Composite Structural diagram* is used to show the internal structural of a classifier, classifier interactions with environment through ports and a behavior of a collaboration.
- Profile diagram describes stereotypes, tagged values and constraints. It allows different platforms and domains for adaptation of UML meta-model.

Behavioral diagrams describe how different objects interact with each other in a system. The following are the main seven UML diagrams which lie in the behavioral category:

- *Sequence diagram* is used to represent interactions between objects and the order of interactions of a particular scenario.
- *Communication diagram* is also called collaboration diagram. It is similar to sequence diagram, but the main focus is on a message passed between different objects.
- *Activity diagram* is used to represent business or operational workflow of any component in a system.
- Use case diagram provides a graphical overview of the actors, functions and interactions between these functions.
- *State machine diagram* is also known as state chart diagram. The representation of state machine diagram is quite similar to activity diagram, although the usage and notations change a bit. State machine is very useful to present the behavior of the objects that act according to the current state.
- *Timing diagram* is similar to sequence diagram. However, it depicts the behavior of objects in a given time frame.
- *Interaction overview diagram* is similar to activity diagram. Activity diagram provides a sequence of process's interactions, whereas overview diagram shows a sequence of interaction diagrams.

To improve the quality of the system in MDA, the UML diagrams are used to model the different aspects of the system (e.g., structural and behavioral).A view is a collection of models that describe similar characteristics of the underlying system. UML diagrams are divided into two views: single-view and multi-view models. The existing state-of-the-art proposed schemes mainly focused on single-view models. Single-view UML modeling considers only one aspect of

the system(Banerjee et al., 2012; Hilken et al., 2014).On the other hand, multi-view modeling considers both structural and behavioral aspects (e.g., class and sequence diagrams) together to represent the system(Khan & Porres, 2015). The benefit of multi-view over single-view is its diversity and representational methods of a system in different angles. Table 1 depicts the studies that used single-view and multi-view aspects of the system. Figure 8 and 9represent the examples of single-view and multi-view representations of a system by using class and sequence diagrams, respectively. Figure 8 represents a simple example of a single-view depiction of the system in which only structural aspects are represented. On the other hand, Figure 9 represents multi-view depiction of the academia system. Moreover, the structural aspect is represented by a class diagram, while the behavioral aspect of the system is represented by a sequence diagram.

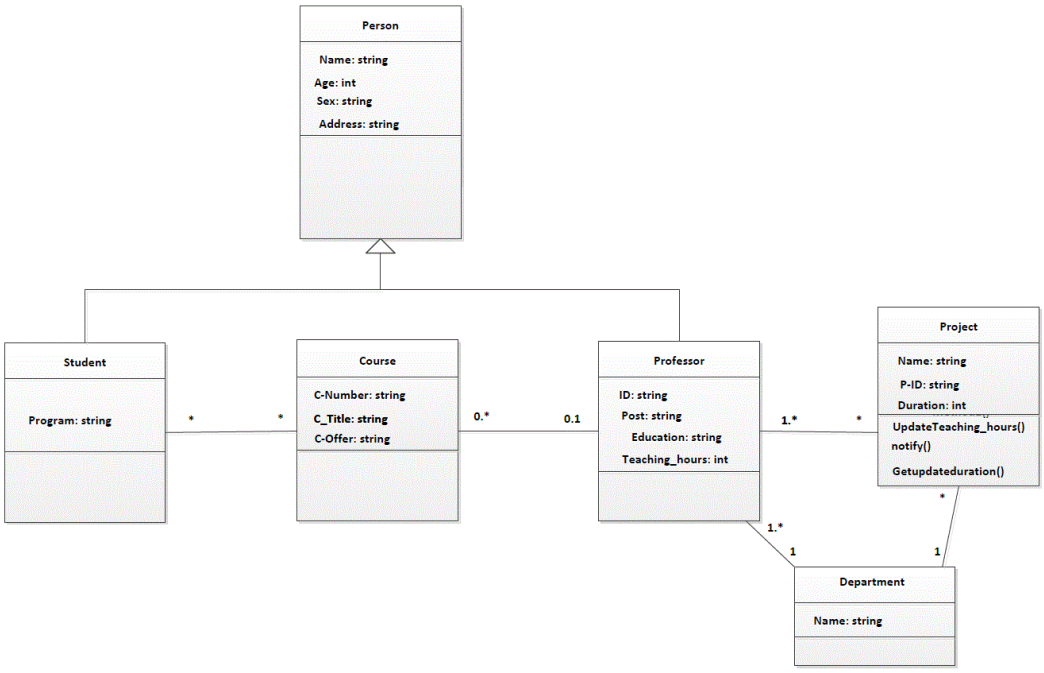


Figure 8: Single View of academia system

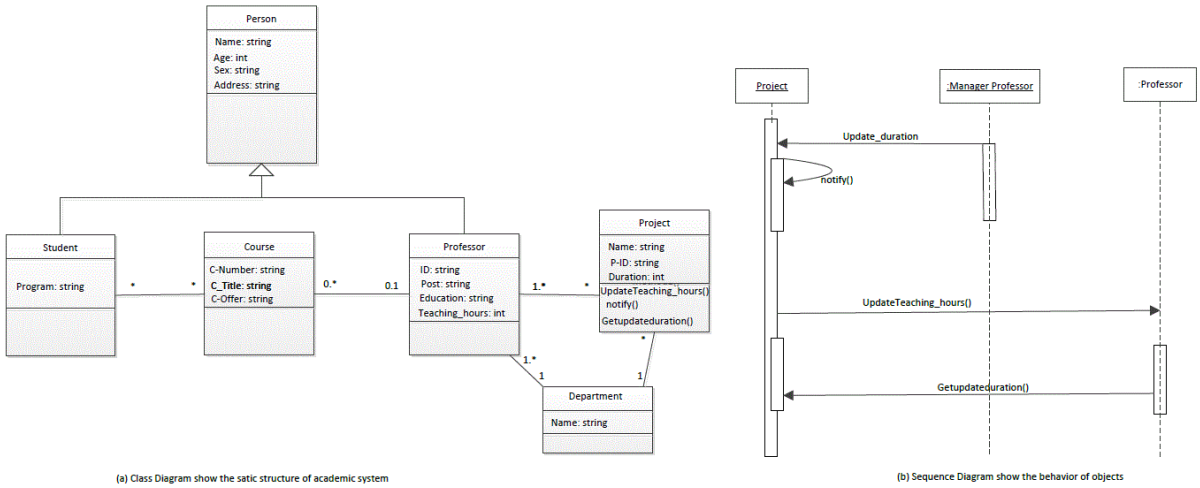


Figure 9: Multi-view representation of the academia system

Table 1 presents a comparison of consistency management techniques based on seven different parameters including: (i) UML Diagram Support, Model-View, Syntactic Consistency, Semantic Consistency, Horizontal consistency, Vertical Consistency, and View Category. We observed that single-view diagrams got more attention as compared to multi-view diagrams support. Moreover, class, state chart and sequence diagrams are the most frequently used diagrams in multi-view modeling to ensure the correct model transformation and preserve the consistency. Moreover, syntactic and horizontal consistency have given much importance. In contrast, vertical consistency and semantic consistency have not been explored in depth.

Table 1. Comparison of consistency management techniques

Ref.	UML Diagram Support	Model-View	Syntactic Consistency	Semantic Consistency	Horizontal Consistency	Vertical Consistency	View Category
(Dam, Egyed, Winikoff, Reder, & Lopez-Herrejon, 2016)	Sequence Class Activity	Multi	✓	✓		✓	Structural Behavioral
(Hilken et al., 2014)	Activity Contracts	Single	×	×	✓	×	Behavioral
(Banerjee et al., 2012)	State chart	Single	×	✓	×	×	Behavioral
(Inverardi, Muccini, & Pelliccione, 2001)	State chart Sequence	Single	×	✓	✓	×	Behavioral
(Lam & Padget, 2005)	State chart Sequence	Single	×	✓	✓	×	Behavioral
(Martínez & Álvarez, 2005)	Class Communication	Multi	✓	×	✓	×	Structural Behavioral
(Diethers & Huhn, 2004)	Sequence Statechart	Single	✓	×	✓	×	Behavioral
(Egyed, 2007b)	Sequence Class Statechart	Multi	✓	×	✓	×	Structural Behavioral
(Egyed, 2004)	Class	Single	✓	×	×	✓	Structural

(Hausmann, Heckel, & Sauer, 2002)	Objects Class	Single	✓	×	✓	×	Structural
(van Hee, Sidorova, Somers, & Voorhoeve, 2004)	Sequence Class Statechart	Multi	✓	×	✓	×	Structural Behavioral
(Kuster & Engels, 2004)	State chart	Single	×	✓	✓	✓	Behavioral
(Fryz & Kotulski, 2007)	Class Use case	Multi	✓	×	✓	×	Structural Behavioral
(Malgouyres & Motet, 2006)	Class	Single	✓	×	✓	×	Structural
(Mens, Van Der Straeten, & D'Hondt, 2006)	Class Statechart	Multi	✓	×	×	✓	Structural Behavioral
(Wagner, Giese, & Nickel, 2003)	Class	Single	✓	×	✓	×	Structural
(Van Der Straeten, Mens, Simmonds, & Jonckers, 2003)	State chart Sequence	Single	×	✓	✓	×	Behavioral
(Yao & Shatz, 2006)	State charts Sequence	Single	×	✓	✓	×	Behavioral
(Sapna & Mohanty, 2007)	Use case Activity Class Sequence Statechart	Multi	✓	×	×	✓	Behavioral
(Amaya, Gonzalez, & Murillo, 2006)	Sequence Class Use case	Multi	×	✓	✓	×	Structural Behavioral
(Amálio, Stepney, & Polack, 2004)	Class State charts	Multi	✓	✓	✓	×	Structural Behavioral
(Chiorean, Paşca, Cârcu, Botiza, & Moldovan, 2004)	Class	Single	✓	×	✓	×	Structural

2.3. Consistency Detection and Management Techniques

This section provides a brief discussion on existing state-of-the-art consistency management techniques. The existing state-of-the-art techniques can be categorized into formal and informal techniques.

2.3.1. Formal techniques

A technique is considered to be formal if it specifies syntax, semantics at a meta-level (Gogolla, 2004). The existing formal techniques can be classified with respect to three formal methods: (i) state transition, (ii) process algebra, and (iii) logic methods. The state transition based formal techniques describe a relation of transition on a set of states. The process algebra based formal techniques use algebraic operations to define how events may occur. Logic-based formal techniques use equations that represents how functions are related.

2.3.2. Non-formal techniques

In non-formal based techniques, constraints are defined for UML meta-model. Then, consistencies are traced according to these defined constraints. The existing work of non-formal techniques considered both single diagram (Berkenkötter, 2008; Pakalnickiene & Nemuraite, 2015) as well as multi diagrams(Egyed, 2007a; Kalibatiene,

Vasilecas, & DUBAUSKAITE, 2013) for model inconsistencies identification and resolution. Table 2 represents a state-of-the-art studies based on formal and non-formal methods.

Table2: Comparison of UML consistency management techniques based on formaland non-formal techniques

Reference	Formal Technique	Non-formal Technique
(Zafar, 2016)	State transition	Nil
(Khan & Porres, 2015)	Logic	Nil
(Cabot, Clarisó, & Riera, 2014)	Nil	OCL constraints
(Zhang Chen & Zhenhua 2011)	State transition	Nil
(Choppy, Klai, & Zidani, 2011)	State transition	Nil
(Kim & Carrington, 2004)	State transition	Nil
(Bernardi, Donatelli, & Merseguer, 2002)	Logic	Nil
(Hoffmann, Lichter, Nyßen, & Walter, 2009)	Nil	Consistency & sanity constraints
(Simmonds, Van Der Straeten, Jonckers, & Mens, 2004)	Logic	Nil
(Wang, Feng, Zhang, & Zhang, 2005)	State transition	Nil
(Mens, Van Der Straeten & Simmonds, 2003)	Logic	Nil
(Zhao, Long, & Qiu, 2006)	Logic	Nil
(Litvak, Tyszberowicz, & Yehudai, 2003)	Logic	Nil
(Alanazi, 2008)	State transition	Nil
(Engels, Hausmann, Heckel, & Sauer, 2002)	Logic	Nil
(Graaf & Van Deursen, 2007)	Nil	Consistency constraints
(Engels et al., 2001)	Process algebra	Nil
(Inverardi et al., 2001)	Logic	Nil
(Lam & Padget, 2005)	Process algebra	Nil
(Diethers & Huhn, 2004)	State transition	Nil
(Egyed, 2007b)	Nil	Consistency constraints
(Egyed, 2004)	Nil	Consistency constraints
(Hausmann et al., 2002)	State transition	Nil
(Fryz & Kotulski, 2007)	State transition	Nil
(Malgouyres & Motet, 2006)	Logic	Nil
(Mens et al., 2006)	State transition	Nil
(Wagner et al., 2003)	State transition	Nil
(Yao & Shatz, 2006)	State transition	Nil
(Sapna & Mohanty, 2007)	Nil	Consistency rules
(Amaya et al., 2006)	Nil	XMI rules
(Amálio et al., 2004)	State transition	Nil
(Zapata, González, & Gelbukh, 2007)	Logic	Nil

Table 2 highlights the

(Rasch & Wehrheim, 2003)	Logic	Nil
(Van Der Straeten, Simmonds, & Mens, 2003)	Logic	Nil
(Feng & Vangheluwe, 2003)	Logic	Nil
(Kaneiwa & Satoh, 2006)	Logic	Nil
(Elaasar, Briand, & Labiche, 2011)	Nil	QVT
(Pakalnickiene & Nemuraite, 2015)	Nil	Integrity constraints
(Berkenkötter, 2008)	Nil	OCL rules
(Zhe Chen & Motet, 2009)	Nil	XML, C-control

importance of formal methods in MDA for model verification and validation. We compare the existing state-of-the-art consistency management techniques on the basis of two parameters i.e., formal and non-formal techniques. We found that mainly consistency management techniques are based on formal methods (i.e. state transition, algebra and logic). The sate transition and logic methods are most frequently used in this field of study. Moreover, we found that algebra methods got less attention as compared to other two formal methods.

2.3.3. State-of-the-art consistency detection and management techniques

Ali et al., (Khan & Porres, 2015) proposed ontology reasoners based consistency management technique. The authors analyzed UML models containing multiple class, object and state chart diagrams using logic reasoning for OWL2. The main objective of their proposed work is to automatically analyze models and identify the concepts that cause model inconsistencies. They used UML2 version 3.0.0 for consistency validation. The work of (Engels et al., 2001) is based on detecting deadlocks and other ambiguities in UML behavioral diagrams. The authors used UML 1.3 collaboration, sequence and state chart diagrams. The model aspects related to consistency are mapped to a semantic domain in which precise consistency tests can be formulated. In order to validate the consistency, the authors also employed the monitoring strategy to present the consistency rules. (Van Der Straeten, Simmonds, et al., 2003) validated consistency with the help of a descriptive logic approach. The approach uses UML 1.5 class, state chart and sequence diagrams to validate consistency. The authors presented two tools LOOM and RACER to validate the proposed technique. They concluded that RACER tool offers a more extensive query language.

Egyed (Egyed, 2007a) treated the consistency rules as black-box entities and observed their behavior during their evaluation. The author used class and sequence diagrams to validate consistency using 40 case studies. Briand et al., (Briand, Labiche, & Sullivan, 2003) focused on change management in UML models. The models are elaborated using UML 1.4 class, sequence and state chart diagrams which are used before the implementation of the changes. The author formally defined the impact analysis rules by OCL constraints on an adaptation of the UML meta-model. Jay panchan et al., (Pancham & Millham, 2015)proposed a tool that ensures the consistency from specification to the design phase. They presented an algorithm that used a set of action and action link rules to specify the activity and scenario diagrams. Engles et al., (Engels et al., 2002) presented a dynamic meta-modeling rule for consistency validation and used the concept of automated tested environment. The authors employed a monitoring strategy and used UML 1.4 sequence, state chart and collaboration diagrams for consistency management. Finally, they concluded that the proposed technique helps to capture the consistency condition in a formal and precise way.

(Pap, Majzik, Pataricza, & Szegi, 2001) proposed a technique that automatically check the consistency in UML models. It includes state charts to verify completeness and consistency. The authors presented two types of analysis: (i) checking completeness and consistency based on the static structure of the specification, and (ii) performing dynamic analysis by checking safety-related reach ability properties using a model checker. Bernardi , (Bernardi et al., 2002) used UML 1.4 state chart and sequence diagrams to validate consistency in UML models. These helped in automatic translation of state chart and sequence diagrams into Generalized Stochastic Petri Nets (GSPNs). The

authors used GSPN as a formal representation and briefly discussed the consistency rules to perform a monitoring strategy.

Rasch and Wehrheim (Rasch & Wehrheim, 2003) validated consistency problems arising between static and dynamic diagrams. This validation was based on rule-based approach, class and state machine diagrams, and UML version 1.5. The authors have discussed several definitions of consistency, which are based on a common formal semantics for both classes and state machines. They validated the proposed technique using a case study. Long et al., (Long, Liu, Li, & Jifeng, 2005)used the semantic definitions of the UML diagrams. It includes UML 2.0 class, sequence and state chart diagrams to propose a consistency checking looms. The authors provided a theoretical proof to prove the Object-Oriented Language (OOL). Litvak et al., (Litvak et al., 2003) used sequence and state machine diagrams for consistency validation. It also applied an analysis strategy. The authors proposed an algorithm for complex diagram's inconsistency that includes forks, joins, and concurrent composite states. Furthermore, they described that BVUML provides a simple solution for the UML dynamic diagrams related consistency problem.

Kotb and Katayama proposed an intermediate technique (Kotb & Katayama, 2005)to detect inconsistency in UML model. It includes XML semantic approach, which consists of two steps: (i) translating the UML diagrams to its equivalent XML documents, and (ii) checking the consistency of these XML documents. Feng and Vangheluwe (Feng & Vangheluwe, 2003) used to validate the consistency at different development stages. The authors used a chat room case study to validate the consistency by using UML class, sequence and state chart diagrams. They considered intra-model consistency. The proposed technique is based on rules, which consist of four parts as pre-condition, post-condition, guard and counter-rule property.

Haesen and Snoeck (Haesen & Snoeck, 2005)used UML patterns to implement different consistency strategies. The authors used UML 1.5 class and finite state machine diagrams. They represented the static structure using class diagram while maintaining an Object Event Table (OET) and demonstrated the OET events using finite state machine diagram. Furthermore, they presented a MERODE methodology to provide a formal definition of various UML diagrams. Kim and Carrington (Kim & Carrington, 2004) mainly focused on defining the integrity consistency constraints between various UML models. It includes formal object-oriented meta-modeling approach. The authors described the invariants on which integrity consistency constraints between UML models and the provided meta-model level rules against consistency validation in a state chart. In order to apply consistency rules, the authors represented state chart in Object-Z. Simmonds et al.,(Simmonds et al., 2004)used sequence diagram to maintain the consistency by the descriptive logic. The authors provided a mechanism that enables the identification and resolution of consistency problems. In order to deal with inconsistencies of evolving UML, the authors used a description logic tool called Loom, with an extensive query language and associated production rule system.

Grischick (Girschick & Darmstadt, 2006) automatically detected differences between two versions of a class diagram of UML 1.5. The author proposed an algorithm that visualized these differences using a color code that differentiates the properties of class diagram. Shen et al.,(Shen, Wang, & Egyed, 2009) includes UML class diagram to correct class model refinement. It includes Separated Abstraction/Comparison (SAC) approach. The authors proposed Implemented Consistency Rules (IAC) approach and also apply a monitoring strategy. Wang et al. (Wang et al., 2005)used UML sequence and state chart diagrams to validate the consistency of UML behavioral models. The technique formalizes the state chart using finite state processes and then represents the message trace using sequence diagram. In order to support the technique, the authors used an existing tool nemed LTSA. Satoh et al. (Satoh, Kaneiwa, & Uno, 2006) proposed a debugging system which includes the logic programming standard for UML 2.0 class diagram. The authors translated the class diagram into logic program to determine consistency between various versions of a class diagram. This translation was performed using translational approach that maps rules to translate a UML class diagram into logic program.

Bellur and Vallieswaran (Bellur & Vallieswaran, 2006)discussed three important issues related to design evolution, where first two are resolved by the relational meta-model of design and code entities. The authors used use case, class, state chart, sequence, component and deployment diagrams of UML 1.5 version. Shinkawa (Shinkawa, 2006) proposed a use case driven approach for heterogeneous UML models consistency validation. The experimental results shows proposed technique is suitable for both structural and behavioral UML models. The experiment was performed on use case, class, activity, sequence, and state chart diagrams of UML 2.0

Kaneiwa and Satoh (Kaneiwa & Satoh, 2006)focused on tractable consistency checking of UML class diagram using a descriptive logic approach. The technique first translates the class diagram into first predicate logic and then validates the consistency. The authors defined an algorithm which computes consistencies with respect to the size of a class diagram. Zapta et al.,(Zapata et al., 2007)used class and use case diagrams for consistency management. The authors provided a rule-based system to detect consistency problem in UML diagrams. For that purpose, the authors presented an approach to use Xquery and Xpath, which also includesa monitoring strategy. Graaf and Deursen (Graaf & Van Deursen, 2007)proposed a MDA based technique based on UML state machine, state chart and sequence diagrams. The technique includes normalization, transformation, and comparison steps, where transformation is based on Atlas Transformation Language. (Larson & Chang, 2016)Highlights the future trends of model-based agile development for business. Alanazi (Alanazi, 2008) discussed inconsistency in link transitions of multiple state diagrams. The author presented a consistency analysis technique for the UML state and sequence diagrams, known as Super State Analysis (SSA) suitable for multiple diagrams. SSA model uses a transition set that captures relationship information that is not specifiable in UML diagrams.

Hoffmann et al. (Hoffmann et al., 2009) used the case model for consistency validation. The authors discussed a complete meta-model for textual use case descriptions from which the narrative meta-model is perceived. They also presented NaUTiluS, a use case modeling tool to implement the UML use case meta-model. The approach in(Mens et al., 2003)is based on descriptive logic for Object-Oriented legacy systems. The authors validated the approach using state chart and sequence diagrams. They used two tools named Loom and Racer. They performed many small-scale experiments using these tools for inconsistency inspection and proved about the usefulness of descriptive logic. Wahler et al.,(Wahler, Basin, Brucker, & Koehler, 2010)work focused on detecting the weak and strong consistency problems among class and object diagrams using a polynomial-time approach. Zhao et al.,(Zhao et al., 2006)validate the consistency between sequence and state chart diagrams using SPIN model checker. They also proposed an algorithm that translates the diagrams to Promela, which is the input language of model checker SPIN model.

Web apps are highly intensive applications and needs efficient languages for modeling. UML can be used to model web application like user interfaces for clear representation. Web application complexity can be reduced using various UML diagrams. UML modeling offers several of diagrams that matches the need of development better. UML includes following benefits: (i) Modeling web applications using UML models (e.g., use case model, security model, deployment model, implementation model), (ii) UML model the web pages, dynamic contents and hyperlink on both server and client side, (iii) UML can also be used to model the HTML forms, and (iv) UML also model the execution of business logic in web elements and technology (Fatolahi, Some, & Lethbridge, 2011) (Hennicker & Koch, 2001). UML models provide support in requirement analysis and gathering in real software systems. For example embedded software systems design needs efficient representation for software specification and analysis. UML model contains features that can be used in real-time domain for development of new design flows. UML model the system from functional requirements through executable specifications for embedded system (R. Chen et al., 2003; Kukkala, Riihimaki, Hannikainen, Hamalainen, & Kronlof, 2005)

UML diagrams used for software requirement analysis, but it can be used for business modeling. Later, these business can be the direct input to the requirement model, which helps in requirement refinements. Although in well-established business situations, business modeling is often invaluable. Similarly, if you are starting a new business from scratch, then defining a business model can provide valuable insight. In today's competitive market, making sure that you solve the right problem in the business context can mean the difference between success and failure for your entire business. Using the UML to model your business and requirements can help you get there.

Table 3 provides a comparison of the existing state-of-the-art consistency management techniques based on taxonomy parameters identified in the research literature.

Table 3: Comparison of UML models consistency management techniques based on taxonomy parameters

Ref. No.	Required Software Support	UML Version	UML Diagrams	Consistency Support	Intermediate Representation	Consistency Strategy	Case Study	Tool	VAlidation	Formal Approach	Informal Approach
(Kim & Carrington, 2004)	NG	1.3	SC	Intra-Model	OZ	Monitoring	N	N	Object-Oriented Application	State Transition	Nil
(Zafar, 2016)	NG	NG	SD	Intra-Model	Z	Analysis	Y	Y	Object-Oriented	State Transition	Nil
(Bernardi et al., 2002)	NG	1.4	SD, SC	Semantic and Syntactic	LGSPN	Monitoring	Y	N	NG	Logic	Nil
(Hoffmann et al., 2009)	ViPER Platform	2.0	UC	Inter and Intra Model	Narrative Modal, OCL	Monitoring Analysis	Y	Y	All Desktop Application		Consistency Constraints
(Simmonds et al., 2004)	XMI support	NG	SD	Intra-Model	Extended UML, OCL, Loom	Monitoring	N	Y	Small Desktop Application	Descriptive Logic	Nil
(Wang et al., 2005)	NG	NG	SD	Inter-Model	Labelle Transition System	Analysis	N	Y	NG	State Transition	Nil
(Mens et al., 2003)	CASE Tool	NG	SD, SC	Intra and Inter Model	Descriptive Logic	Analysis	Y	Y	Object-Oriented Legacy Systems	Description Logic	Nil
(Zhao et al., 2006)	Split Automata	NG	SD	Semantic	Promela	Analysis, Monitoring	Y	Y	NG	Description Logic	Nil
(Litvak et al., 2003)	Software Project Model Description File, XML	2.0	SD	Intra-Model	NG	Analysis	Y	Y	Real Time Application	Description Logic	Nil
(Alanazi, 2008)	XML	2.0	SD	Inter and Intra Model	NG	Monitoring	Y	Y	NG	State Transition	Nil

(Engels et al., 2002)	CASE	1.4	SD, CLD	All	Extended UML	Monitoring	Y	Y	NG	Descriptive Logic	
(Graaf & Van Deursen, 2007)	NG	1.4	SD, SC, SM	Intra-Model	NG	Analysis and Construction	Y	N	Embedded Software Application		Consistency Constraints
(Engels et al., 2001)	FDR Software	1.3	SD, CLD, SC	Inter- Model	CSP-OZ	Monitoring	Y	Y	Real Time	Process Algebra	Nil
(Pap et al., 2001)	UML-based CASE Tool	NG	SC	All	NG	OCL, XML	Y	Y	Embedded System Application		
(Dam et al., 2016)	OO	3.0.0	CD, SD, AD	Inter Model		Construction and Monitoring	Y	Y			Nil
(Khan & Porres, 2015)	NG	3.0.0	CD, OD, SCD	Inter-Model	OCL OWL	Analysis	N	Y	Web-based	Descriptive Logic	Nil
(Shinkawa, 2006)	NG	2.0	UCD, CD, SD, AD, SC	Inter-Model	CPN	Analysis	N	N	NG	Use Case Driven	Nil
(Long et al., 2005)	OO Language	2.0	CD,SD, SC	Inter-Model	OOL	Analysis	Y	N	Object-Oriented Application	Sate Transition	Nil
(Kotb & Katayama, 2005)	XMI	NG	NG	NG	XMI Documents.	Analysis	N	Y	XML Application and Documents	XML Semantic Approach	Nil
(Briand et al., 2003)	UML	1.4	CD,SD, SC	Intra-Model	N	Monitoring	Y	Y	NG	UML Model-based	Nil
[28]	UML Interface Wrapper Component	1.3	CD, SD	Inter-Model	N	Construction and Monitoring	N	Y	All Applications	Logic	Nil
(Zapata et al., 2007)	XML and CASE Tool	2.0	UC, CD	Intra-Model	OCL, XQuari, Xpath	Construction and Monitoring	N	Y	Rule-Based Application	Logic	Nil
(Rasch & Wehrheim, 2003)	N	1.5	CD, SM	Intra-Model	CSP-OZ	Monitoring	Y	N	NG	Logic	Nil

(Haesen & Snoeck, 2005)	NG	1.5	CD, SC (FSM)	Intra-Model	MERODE	All	N	Y	Object -Oriented Applications	Model-Driven Approach	Nil
(Van Der Straeten, Simmonds, et al., 2003)	NG	1.5	CD, SD, C	Intra-Model & Evolution	ALCQI	Monitoring	N	Y	NG	Descriptive Logic	Nil
(Feng & Vangheluwe, 2003)	N	1.5	CD, SD, SC	Intra-Model	N	Monitoring	Y	N	Chat room Application	Descriptive Logic	Nil
(Bellur & Vallieswaran, 2006)	XMI	1.5	UCD, CD, SD, SC, CPD, DD	Inter-Model	N	Analysis and Construction	N	Y	Agile Process Development Applications	Step-Wise Approach	Nil
(Satoh et al., 2006)	NG	2.0	CD	Evolution	Logic program	Monitoring	N	N	NG	State Transitional	Nil
(Wahler et al., 2010)	NG	2.0	CD, OD	Inter and Intra Model	OCL	Construction and Monitoring	Y	Y	Industrial Scale Applications		NG
(Shen et al., 2009)	IBM Rational Rose	2.0	CD	Inter-Model	N	Monitoring	Y	Y	Integrated Applications		Separated Abstraction/Comparison (SAC) Approach
(Kaneiwa & Satoh, 2006)	OCL	NG	CD	NG	First-Order Predicate Logic	Analysis	N	N	NG	Descriptive Logic	Nil
(Girschick & Darmstadt, 2006)	NG	1.5	CD	Evolution	N	Analysis	Y	Y	All Type of Applications	Descriptive Logic	Nil

Table 3 provides a qualitative comparison of state-of-the-art UML consistency management techniques. The following are the main observations concluded from the qualitative comparison:

- It is observed that sequence and state chart diagrams got more attentions as compared to other types of behavioral diagrams.
- It is observed that intra-model consistency problems have given much importance as compared to the inter-model consistency problems.
- We found that the different behavioral diagrams have been used to represent the behavioral aspect of the system. The combination of sequence and state chart diagrams has got more attention as compared to the combination of other behavioral diagrams in consistency management techniques.
- In UML model verification and validations the most frequently used method is the formal method (i.e., state transition, logic and process algebra).
- State transition and logic based techniques are largely used in UML model consistency management while process algebra has not been given much importance in this field of study.
- The version 1.5 and 1.4 have been widely used in current state-of-the-art. However, the version 2 has become most focused version in latest studies.
- We found that only the class, sequence and state chart diagrams have been mainly explored in hybrid techniques (e.g., the techniques that use both views, structural and behavioral) as compared to the combination of other structural and behavioral diagrams in consistency management techniques.
- It is observed that class diagram got more attention as compared to other structural UML diagrams.
- The most important observation is that only one study used two structural diagrams together (i.e., class and object diagrams). Therefore, it is concluded that other combinations of structural diagrams are still unexplored.

3. Results and Discussion

This section presents observed trends, analysis, and potential research issues regarding UML consistency management techniques. We analyzed the current state-of-the-art research in order to obtain responses to the identified research questions, which are briefly summarized below. Note that the percentages included below are based on studies mentioned in Tables 1, 2 and 3. RQ1 is excluded from the results and discussion section because it is based on taxonomy, which is already briefly described in taxonomy section.

(RQ2): The second research question asked, "What are the main techniques used to manage consistency in UML modeling?" The analysis reveals that there are two general techniques used to handle the consistency problems in UML modeling: (i) formal, and (ii) non-formal. Formal techniques are based on formal methods such as logics, state transition, and process algebra. On the other hand, non-formal techniques (25%) rely on constraints and rules. We observed that 75% consistency management techniques are based on formal methods. This big percentage shows the importance of the formal methods in dealing with UML consistency management. This importance is due to many reasons such as formal methods permit precision to UML models, and they also lead to a wide range of applications such as model checker, coherence checker, etc. to be developed. Despite all of these advantages of formal methods in an academic research, these techniques are still unpopular in the software industry. The main reason for formal method's unpopularity in industry is due to their difficult mathematical formulations. It is difficult to verify a model with these techniques, especially when a modeler is not expert enough. Figure 10(a) represents the number of studies regarding different consistency management techniques based on formal and non-formal methods.

We found that 51% of the selected studies are based on logic methods as shown in Figure 10(a). The logic methods appeared as the most important method for UML model verification and validation. This is due to their simplicity in representing the knowledge of the world by defining the simple concepts of the application domain and then using these concepts to specify properties of the individuals that exist in the domain. Another important reasoning is that

logic algorithms are well studied and understood due to their extensively well-defined and sound semantics. The remaining formal methods are state transition (41%) and algebra (7%). Moreover, the state transition method is also appeared as the most frequently used method in this domain due to its graphical representation of domain knowledge. The algebra has been overlooked in this field of study due to its equational representation of the domain knowledge. In algebra, developer needs to convert even simple object into complex equations. This is time and resource consuming, which restricts the use of algebra in model verification and validation process.

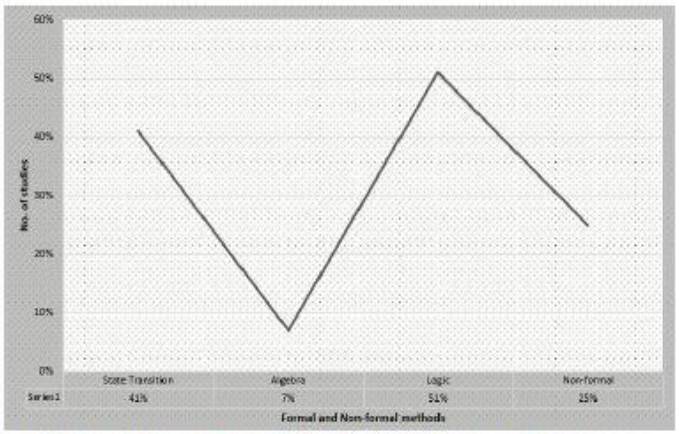
(RQ3): The third research question asked, "*What are the types of UML diagrams that have been used to manage consistency in each approach*?" The analysis reveals that in general, class, sequence and state chart diagrams are the most frequently used UML diagrams in the existing literature. We analyzed 55 most relevant studies regarding consistency management. We found that class diagram, sequence diagram and state chart diagram are considered in 53%, 53% and 46% studies respectively as shown in Figure 10(b).The other UML diagrams got less attention of researchers such as component diagram (0%), object diagram (4%), activity diagram (4%), deployment diagram (0%), use case diagram (14%), interaction diagram (0%), timing diagram (0%), profiling (0%), communication diagram (2%), compositional structural diagram (0%) and package diagram (0%).

Figure 10(c) represents the most frequently UML diagrams that used in combination the existing studies to represent the system at the different levels of abstraction. The results reveal that sequence, state chart and class diagrams are the most frequently used together to represent the structural and behavior of the system. Moreover, the techniques which are specifically designed to handle structural consistency mainly used class diagram for verification and validation. On the other hand, the techniques proposed for behavioral consistency management mainly used sequence and state chart diagrams. In order to represent both views of the system, different structural and behavioral diagrams are used. In the existing studies, we found few combinations of diagrams such as class+ sequence+ state diagrams (31%), class + sequence diagrams (26%) and class + state chart diagrams (15%) for a complete representation of a system.

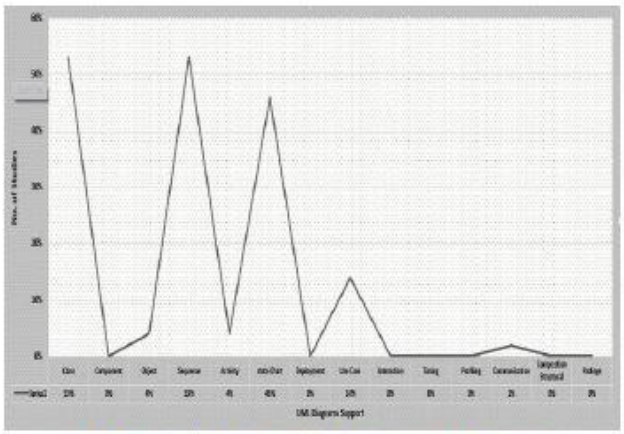
(RQ4): The fifth research question asked, "*What are the types of consistency problems that have been identified and resolved in each approach*?" The result reveals that there are four types of consistency problems that have been identified in UML based modeling: (i) syntactic consistency, (ii) semantic consistency, (iii) horizontal consistency, and (iv) vertical consistency. The horizontal consistency problems depend on overlap's elements and semantic relation of a model elements. On the other hand, vertical consistency problem arises due to software model transformation from one model to another model due to change in requirements. This transformation causes different versions of the same system, and these versions must be consistent with each other to get maximum benefits of MDA. Among all these four consistency issues, horizontal and syntactic consistency problems have given the most importance as compared to semantic and vertical consistency problems. Moreover, 61% state-of-the-art techniques focused on horizontal consistency management. The vertical consistency has not been studied in depth. Only 21% studies have been considered vertical consistency problems as shown in Figure 10(d). Moreover, the semantic consistency problem (20%) has been overlooked in the current state-of-the-art techniques because UML does not support semantic of a domain. For semantics representation in UML,other languages that measure the semantics of a domain are needed.

(RQ5): The fifth research question asked, "*What are the types of UML diagrams views that have been used in state-of-the-art consistency management approaches*" From the existing studies, we found that there are two types of UML diagram views that have been considered in consistency management techniques: (i) single-view, and (ii) multi-view. Single-view based diagram represents only one aspect of the system (i.e., static or dynamic), whereas multi-view based techniques use both static and dynamic views of a system. Single-view approaches has been used most frequently in existing state-of-the-art research (61%). On the other hand, only 38% studies considered multi-view UML diagrams for system representations. Moreover, 83% studies behavioral diagrams in single-view based techniques. In contrast, 16% structural diagrams have been considered in single-view based techniques as depicted in Figure 10(e). Therefore, it is concluded that single-view diagrams have been extensively explored in UML system

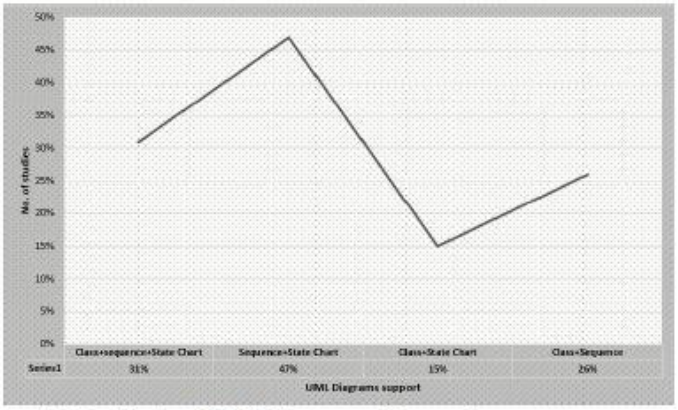
modeling and multi-view diagrams have not been explored in depth. There is a strong need for an automatic technique with the capability to represent a system by using multi-view diagrams in order to get a clear and overall picture of a system.



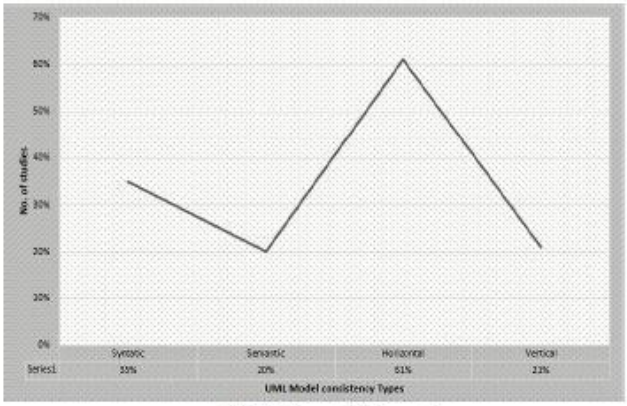
(a): Formal and non-formal consistency management techniques



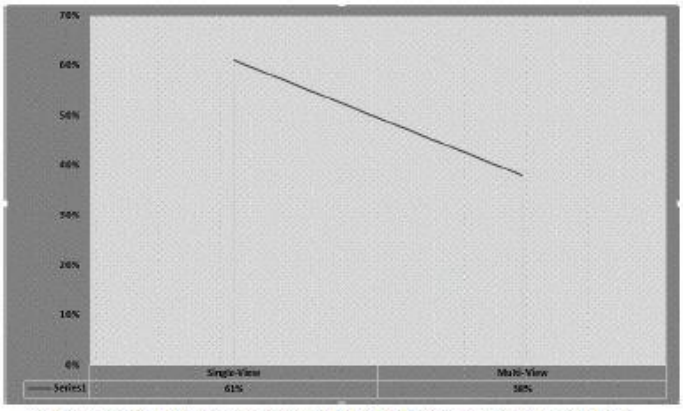
(b): UML diagrams support in consistency management



(c): Combination of UML diagrams used for consistency management



(d): UML model inconsistency types



(e) Single and multi-view aspects of the UML consistency techniques

Figure 10: Represents trends and hype in UML consistency management techniques

3.1. Challenges and Open Research issues

In this section, we highlight some important research issues and challenges that affect the performance and accuracy of model transformation and consistency management techniques.

- In agile development process (Chang, 2015), and internet of things (Li, Darema, & Chang, 2016), a software system evolves rapidly due to rapid change in requirement. To handle this change, UML based model has to change, which ultimately affects the associated diagrams on a different abstraction level. This problem is known as a vertical consistency problem, which may affect the overall model quality as well as system quality. The current state-of-the-art techniques mainly focused only on horizontal consistency problem and lack in vertical consistency management. Therefore, there is a strong need for fully automated techniques that are able to detect and resolve versions based (e.g., vertical) consistency problem without disturbing the behavior of the system.
- Another important issue is to manage a semantic consistency problem in a large and complex system. This issue arises when we wish to check for consistency between different UML diagrams defining the behavior of a software system. For instance, a state machine diagram describing the behavior of a class and has a relationship to a sequence diagram representing the interaction in a system. The relationship is the order in which the messages are received and sent on a lifeline (representing an object of the class) in a sequence diagram. This order must correspond to the order of triggers and effects placed on transitions that are possible to fire in a state machine diagram during the system execution. The state machine diagram, in this case, defines the behavior of the object represented by the lifeline. The described problem is recognized as the semantic consistency problem and is not trivial to detect.
- The third issue is about the applicability and scalability of existing state-of-the-art techniques. The current consistency management techniques are designed, specifically for one particular diagram or set of diagrams and cannot apply to other types of diagrams. For instance, a technique proposed for class diagram to handle consistency issues cannot be applied on sequence diagram for consistency management due to the different nature of representation. As a result, each technique is dependent on one specific diagram. Therefore, it is a strong need for approaches that can handle all types of UML diagrams within a single environment.
- The fourth challenge is how to prioritize the consistency problems. There is no study that identifies the impacts of consistency problems on model quality as well as the overall software quality. Impact measurement is important to provide help in measuring severity of the problem, which ultimately, helps to prioritize the problems based on their severity. It is strongly recommended that researchers explore the clear mapping between these consistency problems and quality factors such as maintainability, understandability, performance, correctness, etc. in order to evaluate the severity of the problems.

4. Conclusion and Future Work

UML offers various diagrams to model software systems. The main challenge is to ensure the consistency between various model related diagrams. UML-based model mainly suffers four types of consistency problems (i.e., syntactic, semantic, vertical and horizontal). In order to verify and validate consistency between various UML diagrams, the researchers have proposed numerous formal and non-formal techniques.

This paper provides a comprehensive overview of current state-of-the-art UML-based model consistency management techniques. In addition, we provided a classification of existing UML model consistency management based on five different criteria, including consistency problem classification, UML diagram support, view support, UML consistency management techniques, and UML version support. Finally, research trends and challenges have been identified to provide a baseline for future research in this domain of study.From the detailed analysis of each parameter of the taxonomy, we found that two main techniques have been employed to manage consistency, i.e. formal (75%) and non-formal (25%). Moreover, we observed that in formal techniques, the state transition (41%) and logic (51) got more attention compared to algebra (7%). Furthermore, class diagram (53%), sequence diagram (53%) and state chart diagram (46%) are the most frequently used UML diagrams for system modeling. In multi-view techniques, class+

state chart (47%) got more attention as compared to other diagrams. We also observed that vertical consistency (21%) and semantic consistency have not been explored in depth as compared to horizontal (61%) and syntactic consistency. As for the future work, we intend to develop a novel multi-view refactoring technique that enables software designers to select most optimized refactoring methods to maintain consistency and behavior within the software system. To this end, we will explore the relationship between various views of UML diagrams to identify the actual change impact on various diagrams as the same time. The approach would provide a multi-aspects of a system to evaluate the impact of structural or static change on associated behavioral diagrams.

Acknowledgements

This work is carried out within the framework of the research project supported by E-Science Fund with reference SF24-2013, funded by the Ministry of Science of Technology and Innovation, Malaysia and High impact research fund with reference UM.C/625/1/HIR/HOME/FCSIT/13 provided by the ministry of Higher Education, Malaysia.

References

Ahmad, R. W., Gani, A., Hamid, S. H. A., Shiraz, M., Yousafzai, A., & Xia, F. (2015). A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications, 52*, 11-25.

Ahmad, R. W., Gani, A., Hamid, S. H. A., Xia, F., & Shiraz, M. (2015). A Review on mobile application energy profiling: Taxonomy, state-of-the-art, and open research issues. *Journal of Network and Computer Applications, 58*, 42-59.

Alanazi, M. N. (2008). *Consistency checking in multiple UML state diagrams using super state analysis*: ProQuest.

Amálio, N., Stepney, S., & Polack, F. (2004). Formal proof from UML models *Formal Methods and Software Engineering* (pp. 418-433): Springer.

Amaya, P., Gonzalez, C., & Murillo, J. M. (2006). Towards a subject-oriented model-driven framework. *Electronic Notes in Theoretical Computer Science, 163*(1), 31-44.

Banerjee, A., Ray, S., Dasgupta, P., Chakrabarti, P., Ramesh, S., Vignesh, P., & Ganesan, V. (2012). A dynamic assertion-based verification platform for validation of UML designs. *ACM SIGSOFT Software Engineering Notes, 37*(1), 1-14.

Bellur, U., & Vallieswaran, V. (2006). *On OO design consistency in iterative development.* Paper presented at the Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on.

Berkenkötter, K. (2008). Reliable UML models and profiles. *Electronic Notes in Theoretical Computer Science, 217*, 203-220.

Bernardi, S., Donatelli, S., & Merseguer, J. (2002). *From UML sequence diagrams and statecharts to analysable petri net models.* Paper presented at the Proceedings of the 3rd international workshop on Software and performance.

Brambilla, M., Cabot, J., & Wimmer, M. (2012). Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering, 1*(1), 1-182.

Briand, L. C., Labiche, Y., & Sullivan, L. (2003). *Impact analysis and change management of UML models.* Paper presented at the Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on.

Cabot, J., Clarisó, R., & Riera, D. (2014). On the verification of UML/OCL class diagrams using constraint programming. *Journal of Systems and Software, 93*, 1-23.

Chang, V. (2015). A Cybernetics Social Cloud. *Journal of Systems and Software*.

Chen, R., Sgroi, M., Lavagno, L., Martin, G., Sangiovanni-Vincentelli, A., & Rabaey, J. (2003). Embedded system design using UML and platforms *System Specification & Design Languages* (pp. 119-128): Springer.

Chen, Z., & Motet, G. (2009). *A language-theoretic view on guidelines and consistency rules of UML.* Paper presented at the Model Driven Architecture-Foundations and Applications.

Chen, Z., & Zhenhua, D. (2011). *Specification and verification of UML2. 0 sequence diagrams using event deterministic finite automata.* Paper presented at the Secure Software Integration & Reliability Improvement Companion (SSIRI-C), 2011 5th International Conference on.

Chiorean, D., Paşca, M., Cârcu, A., Botiza, C., & Moldovan, S. (2004). Ensuring UML models consistency using the OCL Environment. *Electronic Notes in Theoretical Computer Science, 102*, 99-110.

Choppy, C., Klai, K., & Zidani, H. (2011). Formal verification of UML state diagrams: a petri net based approach. *ACM SIGSOFT Software Engineering Notes, 36*(1), 1-8.

Dam, H. K., Egyed, A., Winikoff, M., Reder, A., & Lopez-Herrejon, R. E. (2016). Consistent merging of model versions. *Journal of Systems and Software, 112*, 137-155.

Diethers, K., & Huhn, M. (2004). Vooduu: Verification of object-oriented designs using uppaal *Tools and Algorithms for the Construction and Analysis of Systems* (pp. 139-143): Springer.

Driss Allaki , M. D., ABDESLAM EN-NOUAARY. (2015). A NEW TAXONOMY OF INCONSISTENCIES IN UML MODELS WITH THEIR DETECTION METHODS FOR BETTER MDE *International Journal of Computer Science and Applications, Vol. 12*(No. 1), pp. 48 – 65.

Dzidek, W. J., Arisholm, E., & Briand, L. C. (2008). A realistic empirical evaluation of the costs and benefits of UML in software maintenance. *Software Engineering, IEEE Transactions on, 34*(3), 407-432.

Egyed, A. (2004). Consistent adaptation and evolution of class diagrams during refinement *Fundamental Approaches to Software Engineering* (pp. 37-53): Springer.

Egyed, A. (2007a). Fixing inconsistencies in UML design models. *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, 292-301.

Egyed, A. (2007b). *Fixing inconsistencies in UML design models.* Paper presented at the Software Engineering, 2007. ICSE 2007. 29th International Conference on.

Elaasar, M., Briand, L., & Labiche, Y. (2011). Domain-specific model verification with QVT *Modelling Foundations and Applications* (pp. 282-298): Springer.

Engels, G., Hausmann, J. H., Heckel, R., & Sauer, S. (2002). *Testing the consistency of dynamic UML diagrams.* Paper presented at the Proc. Sixth International Conference on Integrated Design and Process Technology (IDPT 2002).

Engels, G., Küster, J. M., Heckel, R., & Groenewegen, L. (2001). *A methodology for specifying and analyzing consistency of object-oriented behavioral models.* Paper presented at the ACM SIGSOFT Software Engineering Notes.

Fatolahi, A., Some, S. S., & Lethbridge, T. C. (2011). Model-driven web development for multiple platforms. *Journal of Web Engineering, 10*(2), 109.

Feng, T. H., & Vangheluwe, H. (2003). *Case study: Consistency problems in a UML model of a chat room.* Paper presented at the Workshop on" Consistency Problems in UML-based Software Development II.

Fernández-Sáez, A. M., Genero, M., Caivano, D., & Chaudron, M. R. (2016). Does the level of detail of UML diagrams affect the maintainability of source code?: a family of experiments. *Empirical Software Engineering, 21*(1), 212-259.

Fryz, L., & Kotulski, L. (2007). *Assurance of system consistency during independent creation of UML diagrams.* Paper presented at the Dependability of Computer Systems, 2007. DepCoS-RELCOMEX'07. 2nd International Conference on.

Girschick, M., & Darmstadt, T. (2006). Difference detection and visualization in UML class diagrams. *Technical University of Darmstadt Technical Report TUD-CS-2006-5*, 1-15.

Gogolla, M. (2004). Benefits and problems of formal methods *Reliable Software Technologies-Ada-Europe 2004* (pp. 1-15): Springer.

Graaf, B., & Van Deursen, A. (2007). *Model-driven consistency checking of behavioural specifications.* Paper presented at the Model-Based Methodologies for Pervasive and Embedded Software, 2007. MOMPES'07. Fourth International Workshop on.

Haesen, R., & Snoeck, M. (2005). *Implementing consistency management techniques for conceptual modeling.* Paper presented at the Consistency Problems in UML-Based Software Development.

Hausmann, J. H., Heckel, R., & Sauer, S. (2002). *Extended model relations with graphical consistency conditions.* Paper presented at the UML 2002 Workshop on Consistency Problems in UML-based Software Development.

Hennicker, R., & Koch, N. (2001). Systematic design of Web applications with UML. *Unified Modeling Language: Systems Analysis, Design and Development Issues*, 1-20.

Hilken, C., Seiter, J., Wille, R., Kühne, U., & Drechsler, R. (2014). *Verifying consistency between activity diagrams and their corresponding OCL contracts.* Paper presented at the Specification and Design Languages (FDL), 2014 Forum on.

Hoffmann, V., Lichter, H., Nyßen, A., & Walter, A. (2009). Towards the Integration of UML-and textual Use Case Modeling. *Journal of Object Technology, 8*(3), 85-100.

Holt, J. (2004). *UML for Systems Engineering: watching the wheels* (Vol. 4): IET.

Inverardi, P., Muccini, H., & Pelliccione, P. (2001). *Automated check of architectural models consistency using SPIN.* Paper presented at the Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on.

Kalibatiene, D., Vasilecas, O., & DUBAUSKAITE, R. (2013). Ensuring Consistency in Different IS Models–UML Case Study. *Baltic Journal of Modern Computing, 1*(1-2), 63-76.

Kaneiwa, K., & Satoh, K. (2006). *Consistency checking algorithms for restricted UML class diagrams.* Paper presented at the FoIKS.

Khan, A. H., & Porres, I. (2015). Consistency of UML class, object and statechart diagrams using ontology reasoners. *Journal of Visual Languages & Computing, 26*, 42-65.

Kim, S.-K., & Carrington, D. (2004). *A formal object-oriented approach to defining consistency constraints for UML models.* Paper presented at the Software Engineering Conference, 2004. Proceedings. 2004 Australian.

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering–a systematic literature review. *Information and Software Technology, 51*(1), 7-15.

Kotb, Y., & Katayama, T. (2005). *Consistency checking of UML model diagrams using the xml semantics approach.* Paper presented at the Special interest tracks and posters of the 14th international conference on World Wide Web.

Kukkala, P., Riihimaki, J., Hannikainen, M., Hamalainen, T. D., & Kronlof, K. (2005). *UML 2.0 profile for embedded system design.* Paper presented at the Proceedings of the conference on Design, Automation and Test in Europe-Volume 2.

Kuster, J. M., & Engels, G. (2004). Consistency management within model-based object-oriented development of components. *Lecture notes in computer science, 3188*, 157-176.

Lam, V. S., & Padget, J. (2005). *Consistency checking of sequence diagrams and statechart diagrams using the π-calculus.* Paper presented at the Integrated Formal Methods.

Larson, D., & Chang, V. (2016). A review and future direction of agile, business intelligence, analytics and data science. *International Journal of Information Management, 36*(5), 700-710.

Li, C.-S., Darema, F., & Chang, V. (2016). Distributed behavior model orchestration in cognitive internet of things solution. *International Journal of Information Management*.

Litvak, B., Tyszberowicz, S., & Yehudai, A. (2003). *Behavioral consistency validation of UML diagrams.* Paper presented at the Software Engineering and Formal Methods, 2003. Proceedings. First International Conference on.

Long, Q., Liu, Z., Li, X., & Jifeng, H. (2005). *Consistent code generation from UML models.* Paper presented at the Software Engineering Conference, 2005. Proceedings. 2005 Australian.

Lucas, F. J., Molina, F., & Toval, A. (2009). A systematic review of UML model consistency management. *Information and Software Technology, 51*(12), 1631-1645.

Malgouyres, H., & Motet, G. (2006). *A UML model consistency verification approach based on meta-modeling formalization.* Paper presented at the Proceedings of the 2006 ACM symposium on Applied computing.

Martínez, F. J. L., & Álvarez, A. T. (2005). *A precise approach for the analysis of the UML models consistency*: Springer.

Mens, T., Van Der Straeten, R., & D'Hondt, M. (2006). Detecting and resolving model inconsistencies using transformation dependency analysis *Model driven engineering languages and systems* (pp. 200-214): Springer.

Mens, T., Van Der Straeten, R., & Simmonds, J. (2003). *Maintaining consistency between UML models with description logic tools.* Paper presented at the ECOOP Workshop on Object-Oriented Reengineering.

Misbhauddin, M., & Alshayeb, M. (2015). UML model refactoring: a systematic literature review. *Empirical Software Engineering, 20*(1), 206-251.

Muskens, J., Bril, R. J., & Chaudron, M. R. (2005). *Generalizing consistency checking between software views.* Paper presented at the Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on.

Object, M. (2007). Group. OMG Unified Modeling Language (OMG UML), Infrastructure: V2. 1.2. Technical report.

OMG. (2011). OMG,UML,SuperstructureSpecification *Technical Report,ObjectManagementGroup*: Object Management Group.

Pakalnickiene, E., & Nemuraite, L. (2015). Checking of conceptual models with integrity constraints. *Information technology and control, 36*(3).

Pancham, J., & Millham, R. (2015). Design Phase Consistency: A Tool for Reverse Engineering of UML Activity Diagrams to Their Original Scenarios in the Specification Phase *Computational Science and Its Applications--ICCSA 2015* (pp. 655-670): Springer.

Pap, Z., Majzik, I., Pataricza, A., & Szegi, A. (2001). *Completeness and consistency analysis of UML statechart specifications.* Paper presented at the Proceedings IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop.

Rasch, H., & Wehrheim, H. (2003). Checking consistency in UML diagrams: Classes and state machines *Formal Methods for Open Object-Based Distributed Systems* (pp. 229-243): Springer.

Sapna, P., & Mohanty, H. (2007). *Ensuring consistency in relational repository of UML models.* Paper presented at the Information Technology,(ICIT 2007). 10th International Conference on.

Satoh, K., Kaneiwa, K., & Uno, T. (2006). *Contradiction finding and minimal recovery for UML class diagrams.* Paper presented at the Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on.

Shen, W., Wang, K., & Egyed, A. (2009). An efficient and scalable approach to correct class model refinement. *Software Engineering, IEEE Transactions on, 35*(4), 515-533.

Shinkawa, Y. (2006). *Inter-model consistency in UML based on CPN formalism.* Paper presented at the Software Engineering Conference, 2006. APSEC 2006. 13th Asia Pacific.

Shuja, J., Gani, A., Shamshirband, S., Ahmad, R. W., & Bilal, K. (2016). Sustainable Cloud Data Centers: A survey of enabling techniques and technologies. *Renewable and Sustainable Energy Reviews, 62*, 195-214.

Simmonds, J. (2003). Consistency maintenance of UML models with description logics.

Simmonds, J., Van Der Straeten, R., Jonckers, V., & Mens, T. (2004). Maintaining Consistency between UML Models Using Description Logic. *L'OBJET, 10*(2-3), 231-244.

Spanoudakis, G., & Zisman, A. (2001). Inconsistency management in software engineering: Survey and open research issues. *Handbook of software engineering and knowledge engineering, 1*, 329-380.

Van Der Straeten, R., Mens, T., Simmonds, J., & Jonckers, V. (2003). Using description logic to maintain consistency between UML models *«UML» 2003-The Unified Modeling Language. Modeling Languages and Applications* (pp. 326-340): Springer.

Van Der Straeten, R., Simmonds, J., & Mens, T. (2003). Detecting Inconsistencies between UML Models Using Description Logic. *Description Logics, 81*.

van Hee, K., Sidorova, N., Somers, L., & Voorhoeve, M. (2004). Consistency in model integration *Business Process Management* (pp. 1-16): Springer.

Wagner, R., Giese, H., & Nickel, U. (2003). *A plug-in for flexible and incremental consistency management.* Paper presented at the Proc. of the International Conference on the Unified Modeling Language.

Wahler, M., Basin, D., Brucker, A. D., & Koehler, J. (2010). Efficient analysis of pattern-based constraint specifications. *Software & Systems Modeling, 9*(2), 225-255.

Wang, H., Feng, T., Zhang, J., & Zhang, K. (2005). *Consistency check between behaviour models.* Paper presented at the Communications and Information Technology, 2005. ISCIT 2005. IEEE International Symposium on.

Yao, S., & Shatz, S. M. (2006). *Consistency checking of UML dynamic models based on petri net techniques.* Paper presented at the Computing, 2006. CIC'06. 15th International Conference on.

Zafar, N. A. (2016). Formal Specification and Verification of Few Combined Fragments of UML Sequence Diagram. *Arabian Journal for Science and Engineering*, 1-12.

Zapata, C. M., González, G., & Gelbukh, A. (2007). A rule-based system for assessing consistency between UML models *MICAI 2007: Advances in Artificial Intelligence* (pp. 215-224): Springer.

Zhao, X., Long, Q., & Qiu, Z. (2006). Model checking dynamic UML consistency *Formal methods and software engineering* (pp. 440-459): Springer.