Logical Approach: Consistency Rules between Activity Diagram and Class Diagram

Noraini Sulaiman[#1], Sharifah Sakinah Syed Ahmad[#2] Sabrina Ahmad[#3]

[#]Faculty of Information and Communication Technology, Universiti Teknikal Malaysia Melaka (UTeM), 76100 Melaka, Malaysia
*E-mail: [#1]sakinah@utem.edu.my; [*2]sulaiman_noraini@yahoo.com; [#3]sabrinaa@utem.edu.my*

***Abstract*— Requirements engineering (RE) is a fundamental in software development process. Requirements engineering encompasses activities ranging from requirements elicitation and analysis to specification, verification and validation. Poor requirements have been proved to be a major cause of software problems such as cost overruns, delivery delays, failure to meet expectation and degradation. Requirements validation especially models validation has gained quite an interest from a lot of researchers. In recent times, several researchers have expressed a great deal of interest in requirements validation, specifically models validation. The field of research related to consistency checking has undergone a considerable boom from time to time. Numerous methods, approaches and techniques have been recommended to address the requirements inconsistency issues, particularly in models validation. In the software development industry, UML modelling has been extensively used. The different forms of the UML model that characterise the system from various perspectives somehow establish a relation among the models to keep them inseparable from one another. This is the reason why the inconsistency becomes unavoidable. The inconsistency in the models arises when there is an overlap of the elements of the various models representing the different parts of the system and an absence of cooperation. In this paper, the emphasis is given on the consistency rules that exist between the two models. The focus is also on the class diagrams and activity, and the conversion of the rules into logical predicates, where the logical predicates are assessed with a sample case study that constitutes of the two models.**

***Keywords*— requirements engineering; UML modeling; logical approach; consistency rules; activity diagram.**

I. INTRODUCTION

Requirements engineering (RE) is the first phase of the software development process to develop software that is working perfectly and fulfill the client's needs. Requirements engineering encompasses activities ranging from requirements elicitation and analysis to specification, verification, and validation. Poor requirements have been proved a major cause of software problems such as cost overruns, delivery delays, failure to meet expectation and degradation. The requirements inconsistencies normally happen during requirements elicitation phase that makes customer's needs usually uncertain and sketchy. It could lead to an inadequate, incomplete, inconsistent, or ambiguous Software Requirements Specification (SRS). These drawbacks in SRS have a critical impact on the quality of software development. SRS is written in Natural Language (NL). This NL is prone to misunderstanding because of the lack of clarity. It is sometimes difficult to use language in a precise and ambiguous way without making the document wordy and difficult to read. Sometimes it leads to requirements confusion. The developer could not distinguish whether it is a functional requirement or non-functional requirement; sometimes several requirements may be expressed into a single requirement. Tools and techniques were introduced to translate this NL into logic statements by using logic and mathematical formulas [1].

The use of logic is theoretically proved effective to model the requirements by using Unified Modeling Language (UML). UML is a standard modeling language to represent the requirements of the system in diagrammatic notations in object-oriented development practices. The UML currently provides 14 diagrams to visualize the requirements of the system from different aspects [2]. For example, Use Case Diagram (UCD) models the functionalities of the system, Activity diagram (AD) describes the flows of activities and actions of the system, and Class diagram (CD) describes the structure of the system [3]. However, it may not always be possible to get consistent models. The more overwhelming a system is, the more its development obliges an accumulation of distinctive models. The vast scale modern system may include several software engineers taking a shot at many distinctive however related models speaking to parts of the entire system detail. Guaranteeing consistency between those models gets to be basic as even a minor inconsistency can prompt critical faults in the system [4].

Therefore, we need to do requirements validation, which is a concern with checking the requirements for consistency, completeness, and correctness (three Cs) Zowghi and Gervasi stated in their paper about the relationship between these three Cs [5]. To preserve the consistency in requirements, we often failed to preserve their completeness; therefore, it affects the correctness of the conditions because generally in an attempt to complete the requirements, we tend to add more requirements, which increase the possibility of inconsistency to happen. Hypothetically, the increasing of completeness will decrease the consistency and correctness of requirements.

Consistency checking rules can emerge from several sources such as (see Figure 1); Notation definitions; for example, in a strongly typed programming language, the notation requires that the use of each variable be consistent with its declaration. Development methods; for example, a method for designing distributed systems might require that for any pair of communicating subsystems, the data items to be communicated must be defined consistently in each subsystem interface. Development process models; a process model typically defines development steps, entry and exit conditions for those steps, and constraints on the products of each step. Local contingencies; sometimes a consistency relationship occurs between descriptions, even though the notation, method, or process model does not predetermine this relationship. For example, a particular timing constraint in requirement A must be the same as the timing constraint in requirement B. Application domains; many consistency rules arise from domain-specific constraints. For example, the telecommunication domain might impose constraints on the nature of a telephone call. Such constraints can be specified as consistency rules to be checked during development.

There are several techniques or approaches to validate the requirements such as requirements review, prototyping, model validation, requirements testing, etc. Different approaches and tools [6]–[9] have been proposed by the researchers in different ranging of inconsistency management, from diagnosing to handling the inconsistencies. Every researcher stated that how important it is to have good techniques to manage the inconsistencies in requirements regardless at any phase in software development it is being implemented.

In this research, we aim to justify the consistency checking rules for two commonly used UML models in software development, which are Activity diagram (AD) and Class diagram (CD) by using a logical approach. Previous studies are still lack of concerns on these two models, even though activity diagram is one of the top five most used UML diagrams in industry and the fact that the number one most used UML diagram is Class diagram are the reasons why we chose to focus on these two models [10]. The feedback we got from the questionnaire regarding the most used UML diagrams, which the respondents chose activity diagram as their most used UML diagram in their development also has convinced us to focus on these models. Activity diagrams are usually associated with a class as such; they model the operations flow inside the class. Nevertheless, the activity diagram also allows a hierarchical decomposition, with sub activity states, and so it can model several classes related to class aggregation. Using external events, we can even synchronize several activity diagrams. We then validated the rules by providing examples of models from a case study.

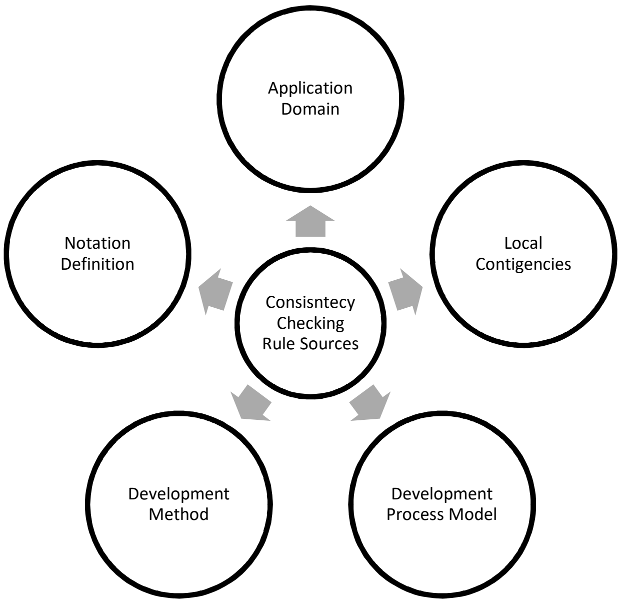


Fig. 1 Consistency checking rule sources.

II. Material and Methods

A. Background Studies

There are several approaches proposed by the researchers regarding consistency checking between UML models. A new approach called View Integra to use consistent transformation to detect the inconsistency by converts the source diagrams into targeted diagrams that need to be compared with (Egyed & Rey 2001). The converted diagrams are called “interpreted” diagrams. They presented a transformation framework for five UML diagrams; class, object, sequence, collaboration, and state machine but there are no consistency rules listed in their work. Shinkawa identified the consistency for UML inter-models using Colored Petri Net (CPN) formalism, where all the models are represented by a common notation [11]. They focused on four diagrams; use case, activity, state machine, and sequence. The drawback from this approach is, to get the consistent models, the original models need to be converted into CPN models then convert them back into their original states, which is taking quite a time to do that.

Sapna & Mohanty [12] chose to use direct approach by proposing the rules for structural inter-model consistency based on Object Constraint Language (OCL), which is primarily used to determine structural consistency rules and the relationship between the diagrams then transformed the rules into SQL triggers and applied the rules to diagrams saved in a repository. Their focused diagrams are a use case, activity, class, sequence and state machine. Kalibatiene et al. proposed a rule-based method to check consistency in UML diagrams [13]. The proposed method was assessed using comparative analysis and questionnaires. They elicited 50 consistency rules from 11 reviewed papers and from the 50 consistency rules; they evaluated the rules and removed the redundant rules.

Meanwhile, Torre [14] has successfully introduced 190 consistency rules for all 14 UML diagrams out of 619 consistency rules through empirical research. Compared to [13], which was focused on technique to identify the inconsistency and did not present any consistency rules, Torre has presented the whole collection of the rules in their paper.

Chanda et al. proposed a framework for models verification that composes syntactic correctness rules, consistency rules and traceability rules based on the relationship between the models [15]. By using a context-free grammar (CFG) and UML 2.0 standard, they have defined few rules of the syntactic correctness of the diagrams, diagram traceability, and consistency based on the common elements shared by the focused models. They have used Lex and YACC to validate the CFG. They have defined traceability rules to ensure the consistency between the models by mapping the common elements from use case to activity and from activity to class.

Ibrahim et al. proposed three structural consistency rules between use case diagram and activity diagram using logical approach [16]. They defined the elements of those two models gathered from other literature then formalized the elements to construct their proposed consistency rules. Khan proposed to check the consistency of UML by using logical reasoner [17]. The approach proposed the translations of the UML based designs into the form of logic facts such as predicate logic and then used an automatic logical reasoner to infer the logic facts. A reasoner performed the reasoning by checking the set of inferences rules (predicate logic) for their validation. The paper focused on checking the consistency of class diagrams by translating the diagram into Web Ontology Language (OWL 2) ontology and used the OWL 2 reasoner to reason the translated ontology. OWL 2 provides axioms to translate UML elements into OWL 2 semantics.

Ryndina and Jochen proposed an approach in their paper [18] to establish consistency between business process models and object life cycles using activity diagram and state machines diagram respectively. They defined two consistency notions for a process model and an object life cycle and expressed these in terms of conditions that must hold between the given life cycle and the life cycle that generated from the process model. Those consistency notions were transformed into predicate logic to form equivalence and refinement definitions.

In this paper, we try to justify consistency rules for between two models, activity and class diagrams since there is no research in justifying the consistency rules between these two models; activity and class diagram yet (refer Table 1). A software project mostly comprises of many designs that represent both static and behavior abstractions of the software. In [19], Spanoudakis stated, “Structural consistency rules define the relationship that should hold between the model elements regardless of the way they have been constructed”. The common elements shared by the two models to be identified and defined. The rules then will be justified using a logical approach before they will be tested using a case study that consists of both of the models.

TABLE I
SUMMARY OF CONSISTENCY RULES APPROACH

Articles	**Approach**	**Focused UML Diagram**							
		UCD	**AD**	**CD**	**SMD**	**SD**	**OD**	**COD**	**Others (eg: COMD, ID, DD, CSD, TD IOD, PSMD)**
[20]	transformational			✓	✓	✓	✓	✓	
[11]	transformational	✓	✓		✓	✓			
[12]	direct	✓	✓	✓	✓	✓			
[21]	knowledge base			✓	✓			✓	
[13]	rule-based	✓	✓	✓	✓	✓	✓	✓	✓
[2]	empirical research	✓	✓	✓	✓	✓	✓	✓	✓
[15]	logical	✓	✓	✓					
[16]	logical	✓	✓						
[17]	logical			✓					
[18]	logical		✓		✓				

The previous researches regarding the model's requirements consistency checking were mostly focused on techniques how to detect the inconsistencies between the two models and not the justification for consistency rules especially the rules between activity and class diagrams. Regarding that matter, we chose to identify the rules for these two models and try to justify the rules using a logical approach.

B. Consistency rules between Activity and Class diagrams

UML is a standard modeling language to represent the requirements of the system in diagrammatic notations in object-oriented development practices. The UML models represent the static structural and behavioral of the software system. The developers using class diagrams mostly describe the static structural and the behavioral of the system can be depicted by using activity diagrams or sequence diagrams or state diagrams. In short, the class diagram is used to understand the static structures of classes and activity diagram is used to understand the control flows of process or operation. The lack of researches regarding consistency rules for activity diagram even though activity diagram is one of the top five most used UML diagrams in the industry and the fact that the number one most used UML diagram is Class diagram are the reasons why we chose to focus on these two models [10].

Ohnishi proposed in their paper [22], that to ensure the consistency of these two models, we need to check for three things;

- Classes in AD and CD. The element of class in a class diagram is equal to the element of swim lane or partition in an activity diagram. Swim lane can be referred to a class in an activity diagram.
- Actions in AD and operations in CD and Element of action in activity diagram are equal to the element of operation in the class diagram.
- Control flows between classes in AD and associations in CD. The element of control flows between swim lanes is equal to the association between classes in a class diagram.

1) Rules Collection

Based on the literature reviews from other researchers [2], [12], [15], [16], there is a total of five rules between activity diagram and class diagram have been identified. Table 2 lists the rules between AD and CD.

TABLE III
RULES BETWEEN AD AND CD

No	Rules
1	A class name that appears in an activity diagram also appears in the class diagram.
2	Swim lanes/partition in Activity diagram (represented as class Name in activity state) must be present as a unique class in the class diagram.
3	Each activity in an activity diagram must have a corresponding operation in the class diagram.
4	An action that appears in an activity diagram must also appear in the class diagram as the operation of a class.

2) Rules Refinement

In this step, we removed the duplicates of UML consistency rules that are either identical to or are implied by another rule [23]. We do not need two or more rules that have the same meaning. For example; Rule 1 and 2 for AD and CD (refer to Table 2) are kind of have the same meaning.

a) Rule 1: A class name that appears in an activity diagram also appears in the class diagram.

b) Rule 2: Swim lanes/partition in Activity diagram (represented as class Name in activity state) must be present as a unique class in the class diagram. Both of Rule 1 and Rule 2 above give out the same meaning where the element of swim lane of an activity diagram also represented as class Name should appear as a class in a class diagram. Therefore, we can remove one of the rules or we can create another rule that has the same meaning as those two rules.

c) Rule 3: Each activity partition in an activity must have a corresponding class in the class diagram.

C. Formalize the Models

In this section, we described the formalization of the elements of these three models and then, the consistency rules between them could be shown [16].

Definition 1. A UML Model is defined as a set
Model = {<AD>, <CD>}
Where
$AD = \{ad_i \mid 1 \leq i \leq n\}$ is finite set of activity diagrams.
$CD = \{cd_{ad_i} \mid 1 \leq i \leq n\}$ is finite set of class diagrams for an activity.
Definition 1 descries a UML model that consists of at least one activity diagram and one class diagram.

1) Formalization of AD

The activity diagram (AD) consists of elements in term of;

- Activities or activity states represent the invocation of an operation, a step in a business process.
- Transitions or threads represent the flow of control from one activity to another through a link between the activities.
- Swim lanes represent a mechanism to group activities performed by the same organizational units.

Definition 2. Activity diagram, *ad* is defined as a set
ad = {<N>, <AE>, <C>},
where
$N = \{nodes_i \mid 1 \leq i \leq n\}$ *is a finite set of nodes,*
$AE = \{ae_i \mid 1 \leq i \leq n\}$ *is an edge that connected the nodes,*
$C = \{c_i \mid 1 \leq i \leq n\}$ *is a containment elements*
.

Definition 3. N is a collection of nodes in the AD,
N_{ad_i} = {<CN>, <ON>, <AC>}
where
$CN = \{cn_i \mid 1 \leq i \leq n\}$ *is a finite set of control nodes,*
$ON = \{on_i \mid 1 \leq i \leq n\}$ *is a finite set of object nodes,*
$AC = \{ac_i \mid 1 \leq i \leq n\}$ *is a finite set of action nodes.*

Definition 4. AE is an activity edges,
AE = {<CF>, <OF>}
where
$CF = \{cf_i \mid 1 \leq i \leq n\}$ *is a finite set of control flows,*
$ON = \{of_i \mid 1 \leq i \leq n\}$ *is a finite set of object flows.*

Definition 5. *C* is a containment element,
C = {<ACT>, <AP>}
where
$ACT = \{act_i \mid 1 \leq i \leq n\}$ *is a finite set of activities,*
$AP = \{ap_i \mid 1 \leq i \leq n\}$ *is a finite set of activity partitions.*

Definition 6. *CN* is set of control nodes and defined as disjoint set,
$I \cup AF \cup FF \cup DS \cup J \cup FK \cup M$
where
$I = \{i_i \mid 1 \leq i \leq n\}$ *is a finite set of initial nodes,*
$AF = \{af_i \mid 1 \leq i \leq n\}$ *is a finite set of activity final nodes,*
$FF = \{ff_i \mid 1 \leq i \leq n\}$ *is a finite set of flow final nodes,*
$DS = \{ds_i \mid 1 \leq i \leq n\}$ *is a finite set of decision nodes,*
$J = \{j_i \mid 1 \leq i \leq n\}$ *is a finite set of join nodes,*
$FK = \{fk_i \mid 1 \leq i \leq n\}$ *is a finite set of fork nodes,*
$M = \{m_i \mid 1 \leq i \leq n\}$ *is a finite set of merge nodes.*

2) Formalization of Class Diagram (CD)

The class diagram (CD) consists of elements in terms of;

- Objects grouped into classes

- Properties of classes that consist of attributes and operations
- Relationships between classes called associations

Definition 7. *cd* described a class diagram for an activity diagram ad_i and is defined as a finite set of class diagrams,
cd_{ad_i} =\{$cd_{ad_{i1}}, cd_{ad_{i2}}, \ldots, cd_{ad_{in}}$|$ad \in AD$\}
where $cd_{ad_i} \in CD$

Definition 8. *Let for each class diagram for an activity diagram,* cd_{ad_i} *is defined as*
cd_{ad_i} = {<Class>,<Rel>},
Where *Class* = {$class_i$| $1 \leq i \leq n$} *is a finite set of classes in* cd_{ad_i},
Definition 9. *A Class is a classifier, which describes a set of objects that share the same attributes and methods in* cd_{ad_i}
$Class_{cd_{ad_i}}$ = {<Name>, <Att>,<Operation> }
Where
Name = {$name_i$|$1 \leq i \leq n$ }*is a name of the class in* $class_i$
Att = {att_i|$1 \leq i \leq n$ }*is a finite set of attributes in* $class_i$
Operation = {op_i|$1 \leq i \leq n$ }*is a finite set of methods in* $class_i$

Each class is characterized by a name, which is unique for each one, and a set of properties called attributes and operations.

3) Formalization on Consistency between AD and CD

Rule 1: An activity partition in an activity diagram must have a corresponding class in a class diagram.

Proposition 1. If there is an activity partition in the activity diagram, then there exists a corresponding class for the activity partition.

Justification. Let given
C = {<ACT>, <AP>} *is a containment elements*
Where
AP = {ap_i|$1 \leq i \leq n$} *is a finite set of activity partitions*
Let cd_{ad_i} = {<Class>,<Rel>},
Where
Class = {$class_i$| $1 \leq i \leq n$} *is a finite set of classes in* cd_{ad_i},
cd_{class_i} = {$cd_{class_{i1}}, cd_{class_{i2}}, \ldots, cd_{class_{in}}$| $class \in CD$}
Therefore, $\forall\, ap_i \in AD : \exists cd_{class_{in}}, where\ cd_{class_{in}} \in CD$

Rule 3. An action in an activity diagram must have a corresponding method in a class diagram.
Proposition 3. If there is an action in an activity diagram, then there exists a method in a class in the class diagram.
Justification. Let given
N_{ad_i} = {<CN>, <ON>, <AC>} *is a finite set of nodes,*
Where *AC*= {ac_i|$1 \leq i \leq n$ } *is a finite set of action nodes in,*
Let $Class_{cd_{ad_i}}$ = {<Name>,<Att>,<Operation> },
Where Operation = { op_i|$1 \leq i \leq n$ }*is a finite set of operations in* $class_i$
And op_{class_i} = {$op_{class_{i1}}, op_{class_{i2}}, \ldots, op_{class_{in}}$| $class \in CD$}
Therefore, $\forall\, ac_i \in AD : \exists op_{class_i}, where\ op_{class_i} \in CD$

III. Result and Discussion

In this research, we use UML Models for Tour Management System (TMS) as a case study to discuss the application of our proposed method. TMS enables visitor requests for the scheme to check the availability of the desired tour package. This information is stored in the Tour Information System. The system will check whether the customer is existing or new. The new user will enter his personal and tour details for the reservation. In turn, he/she is provided with a system-generated unique ID and password for Login. When a customer is satisfied with the tour package, he/she will request for reservation of tour. Personal details of a new customer are stored in cust_info while the details regarding the tour selected by the particular customer are stored in tour info and the details regarding it would be restructured in Tour Information System. Existing customer can update his/her details in cust_info and cancel the reservation for a tour from tour_info and changes regarding it are reflected in Tour Information System. The requirements of TMS are captured and visualized using a use case diagram. The functionalities of each use case are then modeled using activity diagrams. To show how UML diagrams fulfilled our proposed consistency rules, we showed one activity diagram (Appendix 1) and a class diagram of the whole system (Appendix 2).

A. Consistency Rules between AD and CD

Rule 1. An activity partition in an activity diagram must have a corresponding class in a class diagram.

For "Tour Information System" activity partition in Appendix 1, there is a corresponding class in a class diagram Appendix 2, i.e.

$$ap_{Tour\ Information\ System} \in AD_{TMS}, then\ cd_{TIS}, where\ cd_{TIS} \in CD_{TMS}$$

Appendix 1 and Appendix 2 fulfilled Rule 1, i.e.,

$$\forall\, ap_{Tour\ Information\ System} \in AD_{TMS} : \exists cd_{TIS}, where\ cd_{TIS} \in CD_{TMS}$$

Rule 2. Each activity in an activity diagram must have a corresponding operation in a class diagram.

For "Select tour details" activity in Appendix 1, there is the corresponding operation in a class diagram Figure 3, i.e,

$$act_{Select\ Tour\ Details}\ AD_{TMS}, then\ \exists op_{Select\ Tour\ Details(\ )}, where\ op_{Select\ Tour\ Details()} \in CD_{TMS}$$

Appendix 1 and Appendix 2 fulfilled Rule 2, i.e,

$$act_{Select\ Tour\ Details} \in AD_{TMS} : \exists op_{Select\ Tour\ Details(\ )}, where\ op_{Select\ Tour\ Details()} \in CD_{TMS}$$

Rule 3. An action in an activity diagram must have a corresponding operation in a class diagram.

For "Get Tour Details" action in Appendix 1, there is the corresponding operation in a class diagram Appendix 2, i.e.,

$ac_{Get\ Tour\ Details}\ AD_{TMS}, then\ \exists op_{Get\ Tour\ Info(\ \ )},$
$where\ op_{Get\ Tour\ Info()}\ \in CD_{TMS}$

Appendix 1 and Appendix 2 fulfilled Rule 3, i.e.,

$ac_{Get\ Tour\ Details}\ \in AD_{TMS}$
$:\ \exists op_{Get\ Tour\ Info()}, where\ op_{Get\ Tour\ Info()}\ \in CD_{TMS}$

IV. CONCLUSIONS

A large number of UML consistency rules have been proposed by researchers to identify inconsistencies between UML models. However, no previous research has proposed the justification for consistency rules between two models; activity and class diagrams. This work presents results obtained by following a systematic protocol, whose aim was to identify and analyze UML Consistency rules from the literature. The set of UML Consistency rules compiled by Torre (2014) was analyzed and consistency rules between the two diagrams, class diagram, and activity diagram were extracted and transformed into predicate logics to justify the validation of the rules. The acquired predicated logics then have been validated against related UML models.

The results from the questionnaire survey confirmed the lack of requirements consistency checking practice within the software development industry. Even standard topics in requirements consistency research are new and unfamiliar to many companies. Most of the respondents said that they don't apply consistency checking because of the time constraint which consistency checking is normally will take time to be done, and their schedule will be left behind. Nevertheless, most of the companies need to improve their requirements consistency practices.

ACKNOWLEDGMENT

The authors would like to thank the Universiti Teknikal Malaysia Melaka for funding the study through PJP/2018/FTK (16A)/S01642. Besides, thank you to the Faculty of Information Technology and Communication for providing excellent research facilities.

REFERENCES

[1] V. Gervasi and D. Zowghi, "Reasoning about inconsistencies in natural language requirements," *ACM Trans. Softw. Eng. Methodol.*, vol. 14, no. 3, pp. 277–330, 2005.

[2] D. Torre, "A systematic identification of consistency rules for UML diagrams," 2015.

[3] H. Eriksson and M. Penker, *Business Modeling With UML: Business Patterns at Work*. John Wiley & Sons, Inc., 2000.

[4] X. Blanc, I. Mounier, A. Mougenot, and T. Mens, "Detecting model inconsistency through operation-based model construction," *Proc. 13th Int. Conf. Softw. Eng. - ICSE '08*, p. 511, 2008.

[5] D. Zowghi and V. Gervasi, "The Three Cs of Requirements: Consistency, Completeness, and Correctness," *Proc. 8th Int. Work. Require. Eng. Found. Softw. Qual.*, no. March, pp. 155–164, 2002.

[6] Z. Liang and G. Wu, "Consistency Checking of Multiviews Based on Agent." IEEE, Hubei, China, 2004.

[7] L. I. U. Hua-xiao, W. Shou-yan, and J. I. N. Ying, "A Tool to Verify the Consistency of Requirements Concern Model," 2013.

[8] M. Kamalrudin, "Automated Software Tool Support for Checking the Inconsistency of Requirements," *2009 IEEE/ACM Int. Conf. Autom. Softw. Eng.*, pp. 693–697, Nov. 2009.

[9] W. Li, "Toward consistency checking of natural language temporal requirements," *2011 26th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE 2011)*, pp. 651–655, Nov. 2011.

[10] G. Reggio, M. Leotta, F. Ricca, and D. Clerissi, "What are the used UML diagrams? A preliminary survey," in *CEUR Workshop Proceedings*, 2013, vol. 1078, pp. 3–12.

[11] Y. Shinkawa, "Inter-model consistency in UML based on CPN formalism," *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, pp. 411–418, 2006.

[12] P. G. Sapna and H. Mohanty, "Ensuring consistency in the relational repository of UML models," *Proc. - 10th Int. Conf. Inf. Technol. ICIT 2007*, pp. 217–222, 2007.

[13] D. Kalibatiene, O. Vasilecas, and R. Dubauskaite, "Ensuring Consistency in Different IS Models – UML Case Study," *Balt. J. Mod. Comput.*, vol. 1, no. 1–2, pp. 63–76, 2013.

[14] D. Torre, "On Collecting and Validating UML Consistency Rules : a Research Proposal," pp. 1–4, 2014.

[15] J. Chanda, a. Kanjilal, S. Sengupta, and S. Bhattacharya, "Traceability of requirements and consistency verification of UML use case, activity, and Class diagram: A Formal Approach," *2009 Proceeding Int. Conf. Methods Model. Comput. Sci.*, 2009.

[16] N. Ibrahim, R. Ibrahim, M. Z. Saringat, D. Mansor, and T. Herawan, "Consistency rules between UML use case and activity diagrams using logical approach," *Int. J. Softw. Eng. its Appl.*, vol. 5, no. 3, pp. 119–134, 2011.

[17] A. H. Khan, *Consistency of UML Based Designs Using Ontology Reasoners*, no. 168. 2013.

[18] K. Ryndina and M. K. Jochen, "Consistency of Business Process Models and Object Life Cycles."

[19] G. Spanoudakis and A. Zisman, "Inconsistency Management in Software Engineering : Survey and Open Research Issues," *Handb. Softw. Eng.*, pp. 329–380, 2001.

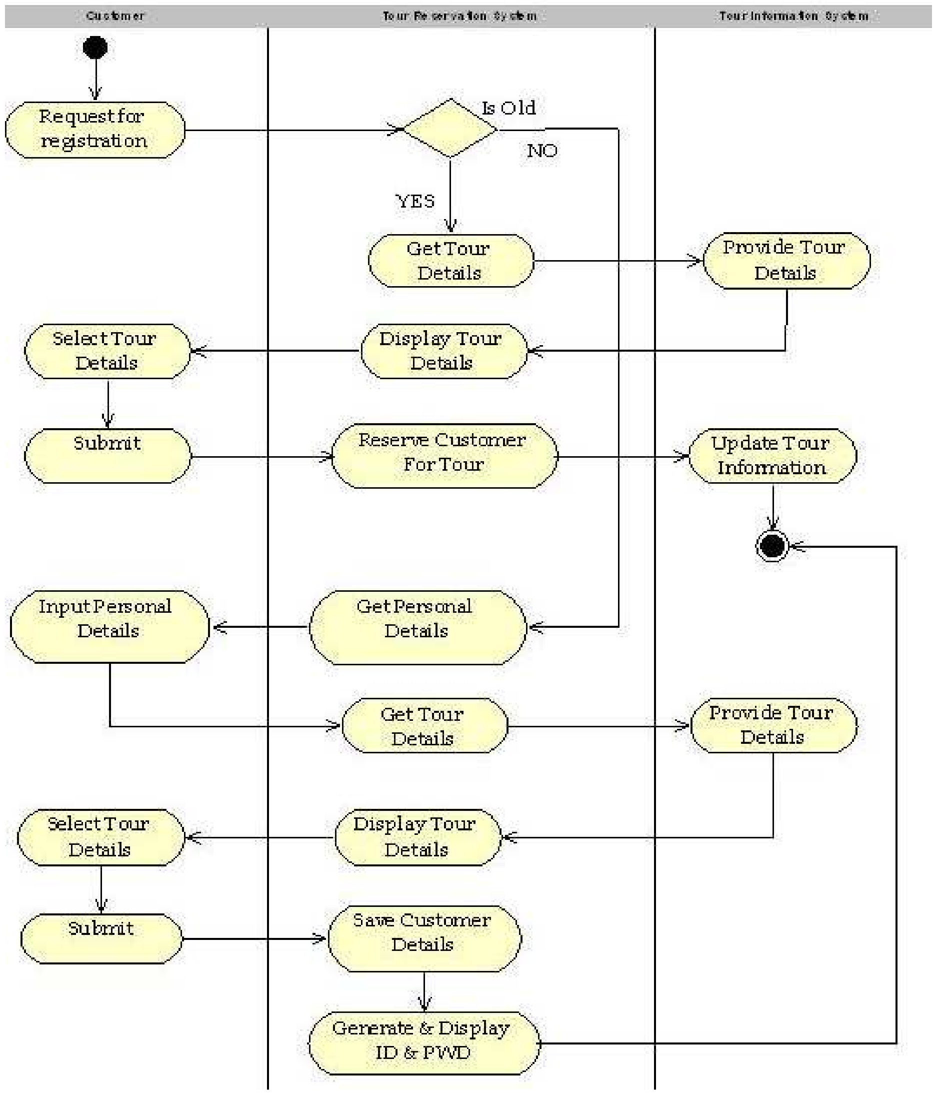
[20] A. Egyed and M. Del Rey, "Scalable Consistency Checking between Diagrams - The VIEWINTEGRA Approach," pp. 387–390, 2001.

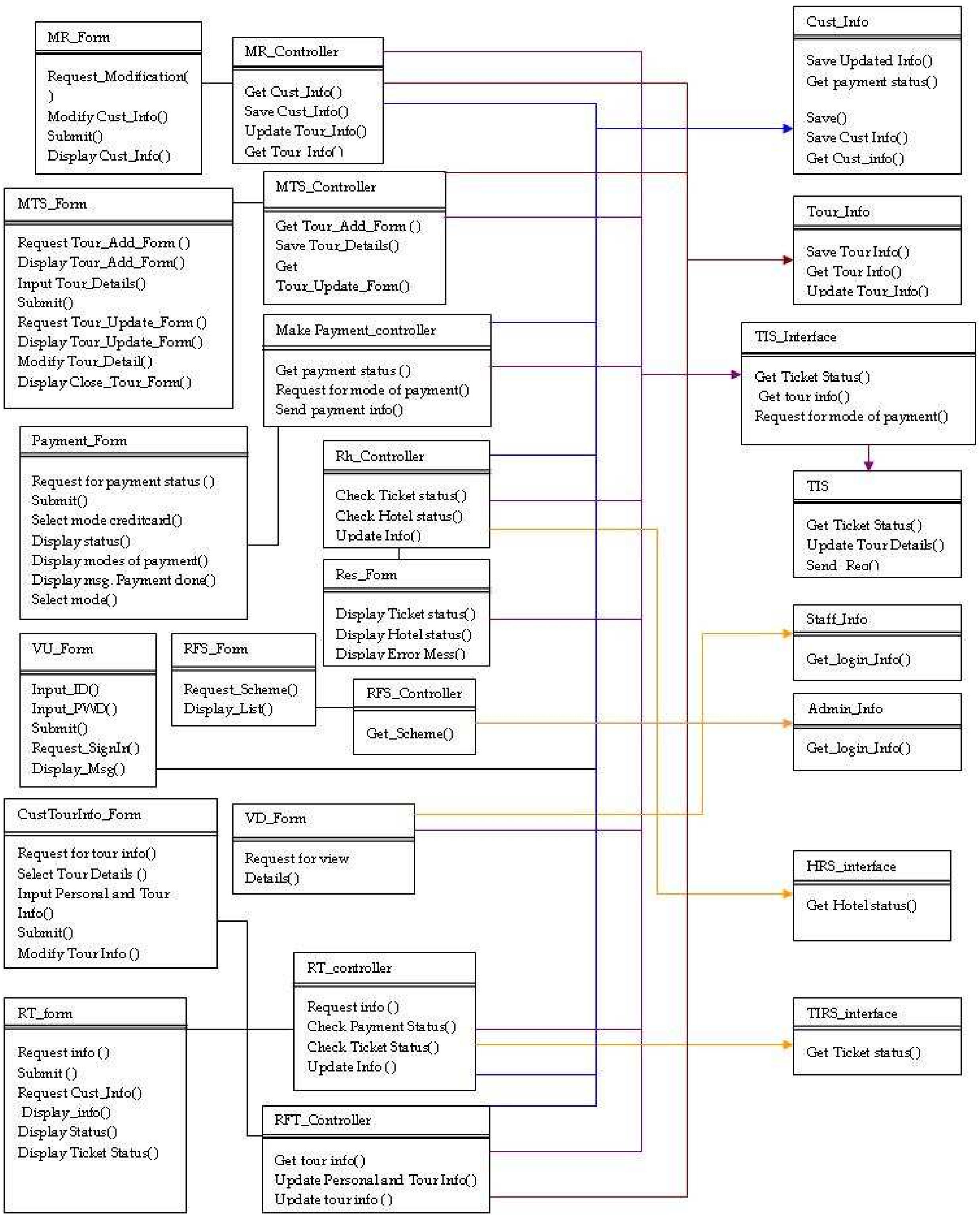
[21] A. Kozlenkov and A. Zisman, "Are their Design Specifications Consistent with our Requirements ?" 2002.

[22] A. Ohnishi, "Management and verification of the consistency among UML Models," no. September 2015.

[23] D. Torre and M. Genero, "UML Consistency Rules : A Systematic Mapping Study," no. January, pp. 1–28, 2014

APPENDIX I


Customer
Tour Reservation System
Tour Information System
Request for registration
Is Old
NO
YES
Get Tour Details
Provide Tour Details
Select Tour Details
Display Tour Details
Submit
Reserve Customer For Tour
Update Tour Information
Input Personal Details
Get Personal Details
Get Tour Details
Provide Tour Details
Select Tour Details
Display Tour Details
Submit
Save Customer Details
Generate & Display ID & PWD


MR_Form
Request_Modification( )
Modify Cust_Info()
Submit()
Display Cust_Info()
MR_Controller
Get Cust_Info()
Save Cust_Info()
Update Tour_Info()
Get Tour_Info()
Cust_Info
Save Updated Info()
Get payment status()
Save()
Save Cust Info()
Get Cust_info()
MTS_Form
Request Tour_Add_Form ()
Display Tour_Add_Form()
Input Tour_Details()
Submit()
Request Tour_Update_Form ()
Display Tour_Update_Form()
Modify Tour_Detail()
Display Close_Tour_Form()
MTS_Controller
Get Tour_Add_Form ()
Save Tour_Details()
Get Tour_Update_Form()
Tour_Info
Save Tour Info()
Get Tour Info()
Update Tour_Info()
Make Payment_controller
Get payment status ()
Request for mode of payment()
Send payment info()
TIS_Interface
Get Ticket Status()
Get tour info()
Request for mode of payment()
Payment_Form
Request for payment status ()
Submit()
Select mode creditcard()
Display status()
Display modes of payment()
Display msg. Payment done()
Select mode()
Rh_Controller
Check Ticket status()
Check Hotel status()
Update Info()
TIS
Get Ticket Status()
Update Tour Details()
Send Req()
Res_Form
Display Ticket status()
Display Hotel status()
Display Error Mess()
Staff_Info
Get_login_Info()
VU_Form
Input_ID()
Input_PWD()
Submit()
Request_SignIn()
Display_Msg()
RFS_Form
Request_Scheme()
Display_List()
RFS_Controller
Get_Scheme()
Admin_Info
Get_login_Info()
CustTourInfo_Form
Request for tour info()
Select Tour Details ()
Input Personal and Tour Info()
Submit()
Modify Tour Info ()
VD_Form
Request for view Details()
HRS_interface
Get Hotel status()
RT_form
Request info ()
Submit ()
Request Cust_Info()
Display_info()
Display Status()
Display Ticket Status()
RT_controller
Request info ()
Check Payment Status()
Check Ticket Status()
Update Info ()
TIRS_interface
Get Ticket status()
RFT_Controller
Get tour info()
Update Personal and Tour Info()
Update tour info ()