Consistency Rules between UML Use Case and Activity Diagrams Using Logical Approach*

Noraini Ibrahim¹, Rosziati Ibrahim¹, Mohd Zainuri Saringat¹, Dzahar Mansor² and Tutut Herawan³
¹Department of Software Engineering
Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia

²Microsoft Malaysia

³Faculty of Computer System and Software Engineering
Universiti Malaysia Pahang
Lebuhraya Tun Razak, Gambang 26300, Kuantan, Pahang, Malaysia
{noraini, rosziati, zainuri}@uthm.edu.my
dmansor@microsoft.com, tutut@ump.edu.my

Abstract

Consistency is one of the attributes in measuring the quality of UML model. It is the situation where two or more overlapping elements of different diagrams that describe behavior of system are jointly satisfiable. Although there are increasing researches on consistency management, there is still lack of researches of consistency driven by use case. Therefore, this paper proposes three consistency rules between use case and activity diagram. The elements of each diagrams and their consistency are described using logical approach. Based on an example of UML model consists of both diagrams, we show how the diagrams fulfilled our proposed consistency rules. Finally, the elements involved in the consistency rules are detected and formally reasoned.

Keywords: UML; Consistency rules; Use case diagram; Activity diagram; Logical approach.

1. Introduction

Any computer systems or software is developed according different software process or set of activities. Although there are various software processes, there are four essential activities in developing software. They are software specification, software design and implementation, software validation and software evolution [1]. Even though it depends on a specific software process whether the activities are sequential, incremental or iterative, in general the functionality and constraint of software must be specified. One of widely used technique to document the software specification is as a set of software model [1]. A software model is an abstract and graphical representation of software functionalities and constraints. It is constructed as a way to understand the software prior to building or modifying it. A model may consists of different diagrams [2]. In object oriented based system, requirements of the

software is visualized, captured and documented using Unified Modeling Language (UML). Currently, UML is represented by thirteen (13) diagrams that used to describe different views of a system. Each UML diagram is built of several notations/abstract syntax and described by their semantics. In UML standard [3], the abstract syntax and semantics are described by Object Constraint Language (OCL) and supported by natural language. However, limitation of OCL to express UML elements [4] and as not all elements are described in OCL [5] forces researches to give more precise definition to UML elements. The precise meaning of the elements is very important in order to have a common understanding of their meaning and make UML can be reason in terms for example the consistency of the diagram.

Consistency is the situation where two or more overlapping elements of different software models that describe the aspects of system are satisfy to joint [6]. Consistency between models is very important to ensure that implementation of the models are not getting trouble [7]. In general there are three (3) main activities in model consistency management. They are consistency specification, inconsistency detection and inconsistency handling [6; 8]. In consistency specification, consistency rules which must be respected by different diagrams in order for them to be consistent are described. If the consistency rules are abided, inconsistencies were aroused and they must be detected and handled. Even though the research on consistency between diagrams is rapidly increased, there is still lack of researches of consistency driven by use case. Shinkawa [9] specify consistency between use case, activity, sequence and statechart diagram using Colored Petri Net (CPN). Fryz and Kotulski [10] have defined consistency between use case and class diagram using graphs. While Sapna and Mohanty [11] describe consistency between use case, activity, sequence, class and statechart diagram. Consistency between use case, activity, sequence and class diagram also has been defined by Chanda et al, [12; 13] using Context Free Grammar (CFG). However, the definition of the consistency rules of the previous approaches [9-13] can be refined and extended.

Therefore, this paper will define elements of use case and activity diagrams, also consistency between them using logical approach. We refine and propose new consistency rules than other researchers [9-13]. Based on an example of UML model consists of the diagrams, we show how the diagrams fulfilled our proposed consistency rules. Furthermore, the elements involved in the consistency rules are detected and formally reasoned. Noted that UML standard [3] is referred in this paper.

The rest of this paper is organized as follows. The discussion on related works is in Section 2. Section 3 presents formalization of the elements of use case and activity diagrams. In this section, we also present definition of three (3) consistency rules. While on Section 4 is an example of a UML system model and elements involved in consistency problem, and Section 5 concludes the paper.

2. Preliminaries

This section describes on formal definition of use case and activity diagrams and consistency rules between them.

2.1 Formal Definition of UML Use Case Diagram (UCD) and Activity Diagram (AD) and Sequence Diagram (SD)

There are various researchers involved in giving formal definition to UCD and AD. However, they focus to several domains and using different technique. Shinkawa [9] defines actor, control flow and lifeline as places of CPN. He also defines use case, action and
execution occurrence as transitions. Fryz et al. [10] consider a use case diagram as user requirements and they described the diagram as a graph. While Sapna et al. [11] define elements of use case, activity and sequence diagrams using schema table. But the definitions just limited to elements of use case, actor, activity, message and object. Chanda et al. [12; 13] express elements of use case, activity and sequence diagram as CFG.

2.2 Consistency Rules between UML UCD and AD

There are no standard consistency rules described even in UML standard [3]. This forces researches to define several of consistency rules. Shinkawa [9] define each use case appears at least in one activity diagram. He defines each UML diagram as CPN model. While Sapna and Mohanty [11] propose each use case in use case diagram must have a corresponding activity diagram and for each actor in a use case diagram there must exist a matching class in the activity diagram. OCL is used to express the consistency rules. Chanda et al. [12] define an action/activity in activity diagram as an event of a use case in use case diagram. The formal syntax of each diagram are then used to reason the rules using CFG.

3. The Proposed Technique

In this section, we described the formalization of the elements of use case and activity diagram. Then, the consistency rules between them are shown.

**Definition 1.** A Model (or UML model) is defined as a set

\[ Model = \{<UCD>,<AD>\}, \]

where

\[ UCD = \{ucd_i \mid 1 \leq i \leq n\} \] is finite set of use case diagrams,

\[ AD = \{ad_{useCase_i} \mid 1 \leq i \leq n\} \] is finite set of activity diagrams for use case.

Definition 1 describes a UML model that consists of at least one use case diagram and one activity diagram.

3.1 Formalization of UCD

A use case diagram ucd may consists of actor (s), use case (s) and association between the actor (s) and use case (s), relationship between use cases, generalization of actors and generalization of use cases.

**Definition 2.** Use Case Diagram (UCD) is defined as a set

\[ UCD=\{<Actor>,<UseCase>,<Rel>\} \]

where

\[ Actor = \{actor_i \mid 1 \leq i \leq n\} \] is finite set of actors,

\[ UseCase = \{useCase_i \mid 1 \leq i \leq n\} \] is finite set of use cases,

\[ Rel = \{<Assoc>,<Include,<Extend>,<GenUC>,<GenAc>\} \] is finite set of relationship.
An actor is person or thing that interacts to system. Behavior offered by a system is represented by use cases. A use case is the specification of a set of actions performed by a system. Relationship between actor and use case (Assoc), between use cases (Include, Extend and GenUC) and between actors (GenAc) are represented by relationship.

Definition 3. An association between an actor and a use case is defined as a binary relation \( \text{assoc} = (\text{actor}, \text{useCase}) \), where \( \text{actor} \in \text{Actor} \) and \( \text{useCase} \in \text{UseCase} \).

Note that \( \text{assoc} \in \text{Assoc} \).

A use case may have relationship to other use case(s) through include and extend relationship.

When a use case include other use case, the \(<\text{include}>\) arrow is from the including use case to included use case. Once an including use case include the included use case, the included use case is not optional. Outcome of the included execution is always required for the including use case to accomplish properly. It means that, including use case depend on the result of the included use case. The included use case cannot include itself.

Definition 4. Let \( \text{UseCase} = \{\text{useCase}_i, 1 \leq i \leq n\} \). include is defined as a relationship between a use case, \( \text{useCase}_m \) including a use case, \( \text{useCase}_n \). Thus

\[
\text{include} = (\text{useCase}_m, \text{useCase}_n),
\]

where

\[
\text{useCase}_m, \text{useCase}_n \in \text{UseCase}
\]

and

\[
\text{useCase}_m \neq \text{useCase}_n,
\]

\( \text{useCase}_m \) is an including use case,

\( \text{useCase}_n \) is an included in use case,

\( \text{include} \in \text{Include} \).

While when a use case is extended, it means the behavior of the use case may be extended by other use case. The execution of the extended use case is not depending to the result of the extending use case. Extending use case may become extended use case. The arrow of \(<\text{extend relation}>\) is from extending use case to extended use case.

Definition 5. extend is defined as a relationship between a use case, \( \text{useCase}_p \) and extended by a use case, \( \text{useCase}_q \), i.e.

\[
\text{extend} = (\text{useCase}_p, \text{useCase}_q)
\]

where

\[
\text{useCase}_p, \text{useCase}_q \in \text{UseCase},
\]

\( \text{useCase}_p \) is an extended use case,
A use case also may generalize to other use case(s). It is not symmetric, not reflexive and transitive.

**Definition 6.** *GenUC* is defined as a generalization of use case, i.e.

\[
GenUC = \{ \text{useCase}_e, \text{useCase}_f \mid \text{useCase}_e, \text{useCase}_f \in \text{UseCase} \land \text{useCase}_e \neq \text{useCase}_f \land (\text{useCase}_f, \text{useCase}_e) \notin \text{GenUC} \}
\]

Let say, if \(GenUC = \{(\text{UseCase}_e, \text{UseCase}_f) \mid (\text{UseCase}_f, \text{UseCase}_g) \in \text{GenUC}\}\) then \((\text{UseCase}_e, \text{UseCase}_g) \in \text{GenUC}\)

An actor also may relate to other actor(s) through generalization. It is not symmetric, not reflexive and transitive.

**Definition 7.** *GenAc* is defined as a generalization of actor,

\[
GenAc = \{ \text{actor}_n, \text{actor}_i \mid \text{actor}_n, \text{actor}_i \in \text{Actor} \land \text{actor}_n \neq \text{actor}_i \land (\text{actor}_i, \text{actor}_n) \notin \text{GenAc} \}
\]

Let say, if \(GenAc = \{(\text{actor}_n, \text{actor}_i) \mid (\text{actor}_i, \text{actor}_j) \in \text{GenAc}\}\) then \((\text{actor}_n, \text{actor}_j) \in \text{GenAc}\)

### 3.2 Formalization of AD

**Definition 8.** "ad" describes an activity diagram for a use case, useCase, and is defined as a finite set of activity diagrams,

\[
ad_{useCase} = \big\{ \text{ad}_{useCase_1}, \text{ad}_{useCase_2}, \ldots, \text{ad}_{useCase_i} \mid \text{useCase} \in \text{UseCase} \big\}
\]

where \(ad_{useCase_i} \in \text{AD}\)

**Definition 9.** Let for each activity diagram for a use case, \(ad_{useCase_n}\), is defined as a triple, \(ad_{useCase_n} = \{< N >, < AE >, < C >\}\),

where

\[
N = \{ \text{nodes}_i \mid 1 \leq i \leq n \} \text{ is a finite set of nodes in } ad_{useCase_n},
\]

\[
AE = \{ \text{ae}_j \mid 1 \leq j \leq n \} \text{ is an edge that connected the nodes in } ad_{useCase_n},
\]

\[
C = \{ \text{c}_k \mid 1 \leq k \leq n \} \text{ is containment elements in } ad_{useCase_n}.
\]

**Definition 10.** \(N\) is a collection of nodes in \(ad_{useCase_n}\) and defined as a triple,

\[
N_{ad_{useCase_n}} = \{< CN >, < ON >, < AC >\}
\]
where

\[ CN = \{ cn_i \mid 1 \leq i \leq n \} \text{is a finite set of control nodes}, \]
\[ ON = \{ on_i \mid 1 \leq i \leq n \} \text{is a finite set of object nodes}, \]
\[ AC = \{ ac_i \mid 1 \leq i \leq n \} \text{is a finite set of action nodes}. \]

**Definition 11.** \( AE \) is an activity edges and defined as a triple,
\[ AE = \{ \langle CF \rangle, \langle OF \rangle \} \]
where
\[ CF = \{ cf_i \mid 1 \leq i \leq n \} \text{is a finite set of control flows}, \]
\[ OF = \{ of_i \mid 1 \leq i \leq n \} \text{is a finite set of object flows}. \]

**Definition 12.** \( C \) is an containment and defined as a tuple,
\[ C = \{ \langle ACT \rangle, \langle AP \rangle \} \]
where
\[ ACT = \{ act_i \mid 1 \leq i \leq n \} \text{is a finite set of activities}, \]
\[ AP = \{ ap_i \mid 1 \leq i \leq n \} \text{is a finite set of activity partitions}. \]

**Definition 13.** \( CN \) is set of control nodes and defined as disjoint set of
\[ I \cup AF \cup FF \cup DS \cup J \cup FK \cup M \]
where
\[ I = \{ i_i \mid 1 \leq i \leq n \} \text{is a finite set of initial nodes}, \]
\[ AF = \{ af_i \mid 1 \leq i \leq n \} \text{is a finite set of activity final nodes}, \]
\[ FF = \{ ff_i \mid 1 \leq i \leq n \} \text{is a finite set of flow final nodes}, \]
\[ DS = \{ ds_i \mid 1 \leq i \leq n \} \text{is a finite set of decisions nodes}, \]
\[ J = \{ j_i \mid 1 \leq i \leq n \} \text{is a finite set of join nodes}, \]
\[ FK = \{ fk_i \mid 1 \leq i \leq n \} \text{is a finite set of fork nodes}, \]
\[ M = \{ m_i \mid 1 \leq i \leq n \} \text{is a finite set of merge nodes}. \]

**Definition 14.** Let \( E_{in} \) be incoming activity edges for a node. A node may have one or more incoming activity edges.
\[ E_{in}(node) = \{ ae \mid ae \in AE, ae \text{ is an incoming edge of node where node} \in N \} \]

**Definition 15.** Let \( E_{out} \) be out going activity edges for a node. A node may have one or more outgoing activity edges.
\[ E_{out}(node) = \{ ae \mid ae \in AE, ae \text{ is an out going edge of node where node} \in N \} \]

**Definition 16.** An initial node has no incoming edges.
\[ |E_{in}(i)| = 0, \text{where } i \in I \text{ and } I \subseteq N. \]
where $|X|$ denotes the cardinality of a set $X$.

### 3.4 Formalization on consistency rules between UML UCD and UML AD

#### Consistency Rules 1.
Each use case has at least an activity diagram.

**Proposition 1.** If there is a use case, then there exists an associated activity diagram for the use case.

**Justification.** Let $$ucd = \{<\text{Actor}>,<\text{UseCase}>,<\text{Rel}>\}$$ and $$ad_{\text{useCase}_i} = \{ad_{\text{useCase}_{i1}}, ad_{\text{useCase}_{i2}}, \ldots, ad_{\text{useCase}_{i_n}} | \text{useCase} \in \text{UseCase}\}$$ Therefore, $\forall \text{useCase}_i \in \text{UseCase} : \exists ad_{\text{useCase}_i}, \text{where } ad_{\text{useCase}_i} \in AD$.

#### Consistency Rules 2.
An actor that associates to the use case will be an activity partition in the activity diagram.

**Proposition 2.** If an actor associates to a use case then the actor will be an activity partition in the associated activity diagram.

**Justification.** Let given $$\text{Actor} = \{\text{actor} | 1 \leq i \leq n\},$$ $$\text{UseCase} = \{\text{useCase} | 1 \leq i \leq n\},$$ $$\text{assoc} = \{(\text{actor}, \text{useCase}) | \text{actor} \in \text{Actor}, \text{useCase} \in \text{UseCase}\}$$ where $$\text{assoc} \in \text{Assoc}.$$

Let $$C = \{<\text{ACT}>,<\text{AP}>\}$$ where $$\text{ACT} = \{\text{act} | 1 \leq i \leq n\}$$ is a finite set of activities

and $$\text{AP} = \{\text{ap} | 1 \leq i \leq n\}$$ is a finite set of activity partitions.

Therefore, $\forall \text{actor}, (\text{actor}, \text{useCase}) \in \text{Assoc} : \exists \text{ap}_{\text{actor}}, \text{where } \text{ap}_{\text{actor}} \in \text{AP}_{\text{useCase}}$.

#### Consistency Rules 3.
If a use case, useCase$_i$ (including use case) include a use case, useCase$_j$ (included use case), then in the associated activity diagram for the including use case there exist action node that refer to the associated activity diagram for the included use case.

**Proposition 3.** If a use case, useCase$_i$ (including use case) include a use case, useCase$_j$ (included use case) in use case diagram, then in the associated activity diagram for the including use case there are a node that refer to the associated activity diagram for the included use case.
Justification. Let given
\[ \text{UseCase} = \{ \text{useCase}_i \mid 1 \leq i \leq n \} \]
\[ \text{ad}_{\text{useCase}_i} = \{ \text{ad}_{\text{useCase}_1}, \text{ad}_{\text{useCase}_2}, \ldots, \text{ad}_{\text{useCase}_n} \mid \text{useCase} \in \text{UseCase} \} \]
\[ \text{include} = \{ \text{useCase}_m, \text{useCase}_n \} \]
\[ N_{\text{ad}_{\text{useCase}_i}} = \{ < \text{CN}>, < \text{ON}>, < \text{AC}> \} \]
where
\[ \text{AC} = \{ ac_i \mid 1 \leq i \leq n \} \]
Therefore, \( \forall \text{useCase}, \text{include}(\text{useCase}_a, \text{useCase}_b) : \exists ac_{\text{AD}b}, \text{where} ac_{\text{AD}b} \in AC_{\text{useCase}} \).

4. Result and Discussion

This section shows an example of UML model for Lecture Assessment System (LAS).

4.1 UML Model for LAS

LAS is a web based system that provide effective application and to help the management of university to monitor the progress of all students. It enables lecturers to input student’s mark electronically through web based and to produce student’s grading based on total marks of assignments, tests, quizzes, laboratories and final examination. LAS also provide list of student attendance and enable to notify to the lecturer and produce the warning letters to the specific students. The system also produces several reports.

The requirements of LAS are captured and visualized using a use case diagram as shown in Figure 1. The functionalities of each use case in Figure 1 are then modelled using at least an activity diagram. For the purpose of showing how UML diagrams fulfilled our proposed consistency rules, we just show one activity diagram for Login use case (Figure 2) and three (3) activity diagrams for Retrieve Report use case (Figure 3 - Figure 5).

Based on Definition 1, we get
\[ \text{Model}_{\text{LAS}} = \{ \text{ucd}_1, \{ \text{ad}_{\text{Login}}, \text{ad}_{\text{RetrieveReport}}, \text{ad}_{\text{RetrieveReport}}, \text{ad}_{\text{RetrieveReport}} \} \} \].

4.2 UML Use Case diagram for LAS

Figure 1 shows a use case diagram for LAS.
Figure 1. Use Case Diagram of LAS

a. A use case diagram $\text{ucd}$ is consists of a set of actors $\text{Actor}$, a set of use cases $\text{UseCase}$, a set of association between actors and use cases $\text{Assoc}$, set of generalization of actors, $\text{GenActor}$ and a set of $\text{<<include>>}$, i.e.,

Based on Definition 2, we have the following

$$\text{ucd}_{\text{LAS}} = (\text{Actor}_{\text{LAS}}, \text{UseCase}_{\text{LAS}}, \text{Assoc}_{\text{LAS}}, \text{GenActor}_{\text{LAS}}, \text{Include}_{\text{LAS}})$$

where

$$\text{ucd}_{\text{LAS}} \in \text{UCD}_{\text{LAS}}$$

In the diagram, there are no generalization of use cases and $\text{<<extend>>}$ relationship between use cases.

b. Six actors named Lecturer, Academic Advisor, Head of Department, Academic Management Officer, System Admin and Student Information System, i.e.,

Based on Definition 2, we have the following

$$\text{Actor}_{\text{LAS}} = \{\text{actor}_{\text{Lecturer}}, \text{actor}_{\text{Academic Advisor}}, \text{actor}_{\text{Head of Department}}, \text{actor}_{\text{Academic Management Officer}}, \text{actor}_{\text{System Admin}}, \text{actor}_{\text{Student Information System}}\}$$

c. Academic Advisor is generalized to Lecturer, i.e.,

Based on Definition 7, we have the following

$$\text{genActor} = (\text{actor}_{\text{Lecturer}}, \text{actor}_{\text{Academic Advisor}}) \text{ where } \text{genActor} \in \text{GenActor}$$

d. Five use cases named Login, Manage Marks, Manage Attendance, Retrieve Report and Manage User, i.e.,

Based on Definition 2, we have the following

$$\text{UseCase}_{\text{LAS}} = \{\text{useCase}_{\text{Login}}, \text{useCase}_{\text{Manage Marks}}, \text{useCase}_{\text{Manage Student Attendance}}, \text{useCase}_{\text{Retrieve Report}}, \text{useCase}_{\text{Manage User}}\}$$

e. Manage Marks, Manage Attendance, Retrieve Report and Manage User use cases include Login use case, i.e.,

Based on Definition 4, we have the following

$$\text{Include}_{\text{LAS}} = \{\text{useCase}_{\text{Manage Marks}}, \text{useCase}_{\text{Login}}\} \cup \{\text{useCase}_{\text{Manage Student Attendance}}, \text{useCase}_{\text{Login}}\} \cup \{\text{useCase}_{\text{Retrieve Report}}, \text{useCase}_{\text{Login}}\} \cup \{\text{useCase}_{\text{Manage User}}, \text{useCase}_{\text{Login}}\}$$

f. Association of actors and use case, i.e.,
Based on **Definition 3**, we have the following

\[
\text{Assoc}_{\text{LAS}} = \{ \text{actor}_{\text{Lecturer}}, \text{useCase}_{\text{Manage Marks}} \} \cup \{ \text{actor}_{\text{Lecturer}}, \text{useCase}_{\text{Manage Student Attendance}} \} \\
\cup \{ \text{actor}_{\text{Academic Advisor}}, \text{useCase}_{\text{Retrieve Report}} \} \cup \{ \text{actor}_{\text{Academic Management Officer}}, \text{useCase}_{\text{Retrieve Report}} \} \\
\cup \{ \text{actor}_{\text{Head of Department}}, \text{useCase}_{\text{Retrieve Report}} \} \cup \{ \text{actor}_{\text{System Admin}}, \text{useCase}_{\text{Manage User}} \} \\
\cup \{ \text{actor}_{\text{Student Information System}}, \text{useCase}_{\text{Manage Marks}} \}
\]

### 4.3 UML Activity diagram for LAS

Figure 2 shows activity diagram for Login use case.

![Activity Diagram for Login Use Case](image)

**Figure 2. Activity Diagram for Login Use Case**

**a.** It is an activity diagram for Login use case, i.e.,

Based on **Definition 8**, we have the following

\[
ad_{\text{useCase}_{\text{Login}}} = \{ ad_{\text{Login}} \}
\]

**b.** An activity diagram \( ad_{\text{Login}} \) is consists of \( \{(N),\{AE\},\{C\}\} \)

Based on **Definition 9**, we have the following

\[
ad_{\text{Login}} = \{(N),\{AE\},\{C\}\}\n\]

Based on **Definition 10** – **Definition 13**, we have the following

\[
ad_{\text{Login}} = \{i_1, ac_1, ds_1, ac_2, ds_2, ac_3, ac_4, ac_5, ds_3, ac_6, ds_4, ac_7, ds_5, ac_8, ds_6\}
\]

**c.** An initial node of an activity diagram \( ad_{\text{Login}} \) has no incoming edges.

Based on **Definition 16**, we have the following

\[
|E_0(i)| = 0, \text{ where } i \in I \text{ and } I \subseteq N
\]
where $|X|$ denotes the cardinality of a set $X$.

Figure 3 - Figure 5 show activity diagrams for Retrieve Report use case.
4.4 Consistency Rules between UML UCD and UML AD

a. For Login use case in Figure 1, there are an associated activity diagrams Figure 2, i.e.

\[
useCase_{Login} \in UseCase_{LAS}, \text{ then } ad_{Login} \in AD_{LAS}
\]

Figure 1 and Figure 2 fulfilled Consistency Rules 1, i.e.

\[
useCase_{Login} \in UseCase_{LAS} : \exists ad_{Login} \in AD_{LAS}
\]

b. In Figure 1, Academic Advisor, Head of Department and Academic Management Officer are associate/ interact to Retrieve Report use case. They are appearing in the associated activity diagrams (Figure 3 - Figure 5) as activity partitions, i.e.

\[
AP_{LAS} = AP_{adAdvisor RetrieveReport} \cup AP_{adHeadDepartment RetrieveReport} \cup AP_{adAcademicManagementOfficer RetrieveReport} \cup AP_{adLogin}
\]

\[
= \{ ap_{AcademicAdvisor}, ap_{HeadOfDepartment}, ap_{AcademicManagementOfficer}, ap_{Login} \}
\]

\[(AcademicAdvisor, RetrieveReport), (HeadOfDepartment, RetrieveReport), (AcademicManagementOfficer, RetrieveReport) \in Assoc_{LAS}
\]

then

\[
ap_{AcademicAdvisor}, ap_{HeadOfDepartment}, ap_{AcademicManagementOfficer} \in AP_{RetrieveReport}
\]
Figure 1 and Figure 3 - Figure 5 fulfilled **Consistency Rules 2.**

d. In Figure 1, Retrieve Report use case has included Login use case. Therefore, in associated activity diagram for the Retrieve Report use case, there is an activity node “AD: Login” that refers to an activity diagram of login, i.e.

\[
\text{include(useCase}_{\text{Retrieve Report}} \text{, useCase}_{\text{Login}}) \in \text{Include}_{\text{LAS}}
\]

\[
\text{useCase}_{\text{Retrieve Report}} : \text{ad}_{\text{Retrieve Report}_1}, \text{ad}_{\text{Retrieve Report}_2}, \text{ad}_{\text{Retrieve Report}_3}
\]

In Figure 3,

\[
\text{ad}_{\text{Retrieve Report}_1} = \left( \text{N}_{\text{Retrieve Report}_1}, \text{AE}_{\text{Retrieve Report}_1}, \text{C}_{\text{Retrieve Report}_1} \right)
\]

\[
\text{N}_{\text{Retrieve Report}_1} = \left( \text{CN}_{\text{Retrieve Report}_1}, \text{ON}_{\text{Retrieve Report}_1}, \text{AC}_{\text{Retrieve Report}_1} \right)
\]

\[
\text{N}_{\text{Retrieve Report}_1} = \{ i_1, i_2, i_3, i_4, i_5 \} \cup \{ i_6, i_7, i_8, \ldots \}
\]

\[
\text{AD}_{\text{Retrieve Report}_1} = \{ \text{AC}_{\text{login}} \}
\]

\[
\text{AC}_{\text{Retrieve Report}_1} = \left( \text{ACT}_{\text{Retrieve Report}_1}, \text{AP}_{\text{Retrieve Report}_1} \right)
\]

\[
\text{ad}_{\text{Retrieve Report}_1} \text{ is an associated activity diagram for Retrieve Report use case. The Retrieve Report use case is including Login use case. Therefore, node } \text{ac}_{\text{AD: login}} \text{ is consist in } \text{ad}_{\text{Retrieve Report}_1}.
\]

Figure 1 and Figure 3 - Figure 5 fulfilled **Consistency Rules 3,** i.e.

\[
\text{useCase}_{\text{Retrieve Report}} \cup \text{include(useCase}_{\text{Retrieve Report}} \text{, useCase}_{\text{Login}}) \Rightarrow \exists \text{ac}_{\text{AD: Login}}, \text{where } \text{ac}_{\text{AD: Login}} \in \text{AC}_{\text{Retrieve Report}}
\]

5. **Conclusion**

UML is a popular modelling technique especially in object-oriented based software development. A UML model produced consists of different diagrams, views of different level of abstraction and contributes from different stakeholders and modellers. These open to consistency problems between diagrams. Even though the research in this issue is increased, there is still lack of research of consistency driven by use case. With this motivation, in this work, we have proposed formalization of use case and activity diagram elements. This formalization is then used to define new consistency rules between the diagrams using logical approach. We intend to use this approach as the platform to conduct the next activity of inconsistency management framework.
Acknowledgement

The authors would like to thank Ministry of Higher Education, Malaysia for supporting this research under the Scheme of Academic Training Initiative (SLAI).

References


Authors

Noraini Ibrahim received his Master of Science degree in Real-time Software Engineering from Universiti Teknologi Malaysia in 2003 and Bachelor of Science degree in Information Technology from Universiti Kebangsaan Malaysia in 2000. She is currently pursuing her study for Ph.D. degree in New Consistency Check for UML Diagrams using Logical Approach at Universiti Tun Hussein Onn Malaysia. His research interests are in software requirement specification, software development and software testing.

Rosziati Ibrahim obtained her PhD in Software Specification from the Queensland University of Technology (QUT), Brisbane and her
Master of Science and Bachelor of Science (Hons) Bachelor of Science degree in Computer Science and Mathematics from the University of Adelaide, Australia. Currently, she is a Dean of Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia. Her research areas cover software specification, software testing, operational semantics, formal methods, data mining, image processing, and object-oriented technology.

Mohd Zainuri Saringat received his Bachelor of Science and Master of Science degree in Computer Science from Universiti Teknologi Malaysia. He is currently pursuing his study for Ph.D. degree in Data Sanitization in Object Oriented Designing for Relational Database System at Universiti Tun Hussein Onn Malaysia. He is a lecturer at Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia. His research area includes system design, database design, and programming technique.

Dzahar Mansor received an Honors Degree in Computer Systems Engineering from Monash University in 1984, and completed his PhD in 1988 in La Trobe University. He is currently Microsoft National Technology Officer Malaysia. His technology and research interest include computer architectures (including multiprocessor, fault-tolerant, parallel and distributed computing), high performance real time systems, software design and engineering as well as machine learning and has published papers in all these areas.

Tutut Herawan received his Bachelor of Education and Master of Science degrees in Mathematics from Universitas Ahmad Dahlan and Universitas Gadjah Mada Yogyakarta, respectively. He obtained his Ph.D from Universiti Tun Hussein Onn Malaysia. Currently, he is a senior lecturer at Computer Science Program, Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang (UMP). He published more than 40 research papers in journals and conferences. His research area includes data mining and KDD, rough and soft set theories.