ClubUML

CSYE7945 Spring 2013

Merge Class Diagrams JSON Structure

Shen Yuanwu

Richard Do

3-23-2013

Revision: 1.1

Table of Contents

[Revision History 3](#_Toc351771316)

[Introduction 4](#_Toc351771317)

[Initial Merge\_Diagram1\_Diagram2 5](#_Toc351771318)

[Refresh 5](#_Toc351771319)

[Merge Sequence 6](#_Toc351771320)

[Add Sequence 8](#_Toc351771321)

[Complete Merge 9](#_Toc351771322)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Author(s)** | **Notes** |
| 1.0 | 3/22/13 | Shen Yuanwu | Initial version |
| 1.1 | 3/23/13 | Richard Do | Updated document with more specific structures |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Introduction

Based on the Sequence diagram for merging 2 class diagrams, this document describes the JSON structure for each step.

# Initial Merge\_Diagram1\_Diagram2

Client will request to merge 2 diagrams. The client-server interface will instantiate an XmiClassComparer which has 2 arguments to pass in diagrams. The client-server interface will then request a refresh shown below to send back to the client.

### Refresh

The state of the merging process is stored on the server, and the **Refresh** request will return the class level information. This **Refresh** request is used at any point the client wants the state of the classes.

#### Client request - Refresh

{

Request: Refresh,

}

#### 2. Server response – (Diagram\_Diagram2\_InitCompareResult and other operations)

*Diagram1* and *Diagram2* contain a list of classes that are unique to each diagram. *Same* contains a list of classes that were merged/consolidated by user or by server during initialization.

{

Response: Success,

Diagram1: [ClassA1, etc…],

Diagram2: [ClassA2, etc…],

Same: [Class1, Class2, etc…]

}

# **Merge Sequence**

Client request to merge 2 classes.

#### 3. Client request - Merge\_Class1A\_Class2C

{

Request: Compare,

Class1: Class1A,

Class2: Class2C

}

#### 4. Server response – Class1A\_Class2C\_Elements

Client interface can display this information and should be straight forward for attribute and operations.

{

Response: Success,

Class1: Class1A,

Class2: Class2C,

Attributes: {

Class1: [],

Class2: [],

Same: [],

},

Operations: {

Class1: [],

Class2: [],

Same: [],

},

Association: {

Class1: [],

Class2: [],

Same: [],

},

Generalization: {

Class1: [],

Class2: [],

Same: [],

}

]

#### 5. Client request - MergeElementResult

Client sends the user’s choices for each option to keep consolidate into a final class.

{

Request: Consolidate,

Class1: {

Class: Class1A,

Elements: {

Attributes: [],

Operations: [],

Association: [],

Generalization: []

},

},

Class2: {

Class: Class2C,

Elements: {

Attributes: [],

Operations: [],

Association: [],

Generalization: []

},

},

Name: NameOfClass, (user selected or default from one of the classes)

}

After consolidation is completed for the 2 classes, the **Refresh** server response (2) JSON is returned.

# Add Sequence

Adding a class is basically the same as a merge/consolidate in step (3), but only 1 class is passed in.

#### 6. Client request - Add\_Class1D

{

Request: Add,

Class1: Class1D,

}

#### 7. Server response - Class3D\_Elements.

List of elements for the class that is being added.

{

Response: Success,

Attributes: [],

Operations: [],

Association: [],

Generalization: []

}

#### 8. Client request - Refined\_Element

Only send elements the user wants to keep.

{

Request: Refined,

Attributes: [],

Operations: [],

Association: [],

Generalization: []

}

After adding is completed, the **Refresh** server response (2) JSON is returned.

# Complete Merge

#### 9. Done

Client presses complete and the merge data is passed into a merge processor for XMI.

#### 10. New\_PNG

After processor completes, a PNG is generated and the path to the PNG is passed back to the client to view.