Part V
Appendix
Appendix A

FGT SEQUENTIAL DEPENDENCY

A.1 Uni-Directional Sequential Dependencies

In the table below, all the uni-directional sequential dependencies between the different FGTs proposed in our approach are catalogued. The information in the table is a continuation of the discussion in section 4.3.2. Each row in the table represents the following sequential dependency: \( \text{FGT}_x \rightarrow \text{FGT}_y \), where \( \text{FGT}_y \) is sequentially dependent on \( \text{FGT}_x \).

Note that each numbered row in the table corresponds to a numbered arc in Figure 4.1 that represents a uni-directional sequential dependency.

<table>
<thead>
<tr>
<th>Number</th>
<th>Rule 1</th>
<th>Rule 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>changeODefType(P,C,M,PR,PLT,ObjT,<em>,ONewDT) ( \rightarrow ) changeODefType(P,C,M,PR,PLT,ObjT,ONewDT,</em>,_)</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>changeOAMode(P,C,M,PR,PLT,ObjT,<em>,ONewAM) ( \rightarrow ) changeOAMode(P,C,M,PR,PLT,ObjT,ONewAM,</em>,_)</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>deleteRelation(<em>,P,C,M,PR,PLT,Ftype,</em>,<em>,</em>,<em>,</em>,<em>,</em>) ( \rightarrow ) changeODefType(P,C,M,PR,PLT,Ftype,<em>,</em>)</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>deleteRelation(<em>,</em>,<em>,</em>,<em>,</em>,<em>,P,C,M,PR,PLT,Totype,</em>) ( \rightarrow ) changeODefType(P,C,M,PR,PLT,Totype,<em>,</em>)</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>deleteRelation(<em>,P,C,M,PR,PLT,Ftype,</em>,<em>,</em>,<em>,</em>,<em>,</em>) ( \rightarrow ) changeOAMode(P,C,M,PR,PLT,Ftype,<em>,</em>)</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>deleteRelation(<em>,P,C,M,PR,PLT,Ftype,</em>,<em>,</em>,<em>,</em>,<em>,</em>) ( \rightarrow ) deleteObject(P,C,M,PR,PLT)</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>deleteRelation(<em>,</em>,<em>,</em>,<em>,</em>,<em>,P,C,M,PR,PLT,Totype,</em>) ( \rightarrow ) deleteObject(P,C,M,PR,PLT)</td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>changeODefType(P,C,M,PR,PLT,ObjT,<em>,</em>) ( \rightarrow ) addRelation(<em>,P,C,M,PR,PLT,ObjT,</em>,<em>,</em>,<em>,</em>,<em>,</em>)</td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>changeODefType(P,C,M,PR,PLT,ObjT,<em>,</em>) ( \rightarrow ) addRelation(<em>,</em>,<em>,</em>,<em>,</em>,<em>,P,C,M,PR,PLT,ObjT,</em>,_)</td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td>changeOAMode(P,C,M,PR,PLT,ObjT,<em>,</em>) ( \rightarrow ) addRelation(<em>,P,C,M,PR,PLT,ObjT,</em>,<em>,</em>,<em>,</em>,<em>,</em>)</td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td>changeOAMode(P,C,M,PR,PLT,ObjT,<em>,</em>) ( \rightarrow ) addRelation(<em>,</em>,<em>,</em>,<em>,</em>,<em>,P,C,M,PR,PLT,ObjT,</em>,_)</td>
<td></td>
</tr>
<tr>
<td>14.</td>
<td>renameObject(P,C,M,PLT,attribute,X) ( \rightarrow ) addRelation(<em>,P,C,X,</em>,<em>,attribute,</em>,<em>,</em>,<em>,</em>,<em>,</em>)</td>
<td></td>
</tr>
<tr>
<td>15.</td>
<td>renameObject(P,C,M,PLT,method,X) ( \rightarrow ) addRelation(<em>,P,C,X,</em>,PLT,method,<em>,</em>,<em>,</em>,<em>,</em>,_)</td>
<td></td>
</tr>
<tr>
<td>16.</td>
<td>renameObject(P,C,X,<em>,class,X) ( \rightarrow ) addRelation(</em>,P,X,<em>,</em>,<em>,class,</em>,<em>,</em>,<em>,</em>,<em>,</em>)</td>
<td></td>
</tr>
<tr>
<td>17.</td>
<td>renameObject(P,C,M,PR,PLT,parameter,X) ( \rightarrow ) addRelation(<em>,</em>,<em>,</em>,<em>,</em>,P,C,M,X,PLT,parameter,<em>,</em>)</td>
<td></td>
</tr>
<tr>
<td>18.</td>
<td>renameObject(P,C,M,<em>,attribute,X) ( \rightarrow ) addRelation(</em>,<em>,</em>,<em>,</em>,<em>,P,C,X,</em>,<em>,attribute,</em>,_)</td>
<td></td>
</tr>
<tr>
<td>20.</td>
<td>renameObject(P,C,X,<em>,class,X) ( \rightarrow ) addRelation(</em>,<em>,</em>,<em>,</em>,<em>,P,X,</em>,<em>,</em>,class,<em>,</em>)</td>
<td></td>
</tr>
<tr>
<td>21.</td>
<td>renameObject(P,C,M,PR,PLT,PR1) ( \rightarrow ) deleteRelation(<em>,P,C,M,PR,PLT,parameter,</em>,<em>,</em>,<em>,</em>,<em>,</em>)</td>
<td></td>
</tr>
<tr>
<td>22.</td>
<td>renameObject(P,C,M,<em>,attribute,PR1) ( \rightarrow ) deleteRelation(</em>,P,C,PR1,<em>,</em>,attribute,<em>,</em>,<em>,</em>,<em>,</em>,_)</td>
<td></td>
</tr>
<tr>
<td>23.</td>
<td>renameObject(P,C,M,<em>,method,PR1) ( \rightarrow ) deleteRelation(</em>,P,PR1,<em>,</em>,method,<em>,</em>,<em>,</em>,<em>,</em>,_)</td>
<td></td>
</tr>
<tr>
<td>24.</td>
<td>renameObject(P,C,<em>,</em>,class,PR1) ( \rightarrow ) deleteRelation(<em>,P,PR1,</em>,<em>,</em>,class,<em>,</em>,<em>,</em>,<em>,</em>,_)</td>
<td></td>
</tr>
</tbody>
</table>
A.2 Bi-Directional FGTs Sequential Dependencies

In the table below, all the bi-directional sequential dependencies between the different FGTs proposed in this thesis are catalogued. The information in the table is a continuation of the discussion in section 4.3.3. Each row in the table represents the following sequential dependencies: FGT_x ↔ FGT_y where FGT_x is sequentially dependent on FGT_y and FGT_y is sequentially dependent on FGT_x. Note that each numbered row in the table corresponds to a numbered arc in Figure 4.1 that represents a bi-directional sequential dependency.

| A. | deleteRelation(Ca,P,C,M,PR,PLT,Ftype,P1,C1,M1,PR1,PLT,Totype,Ltype) ↔ addRelation(Ca,P,C,M,PR,PLT,Ftype,P1,C1,M1,PR1,PLT,Totype,Ltype) |
| E. | renameObject(P,C,_,_,_,class,X) ↔ renameObject(P,X,_,_,_,class,Y) |
| F. | renameObject(P,C,M,PR,PLT,parameter,PR1) ↔ deleteObject(P,C,M,PR,PLT,parameter) |
| G. | renameObject(P,C,M,_,attribute,M1) ↔ deleteObject(P,C,M,_,attribute) |
| I. | renameObject(P,C,_,_,_,class,M1) ↔ deleteObject(P,C,_,_,_,class) |
| K. | renameObject(P,C,M,_,attribute,PR1) ↔ deleteObject(P,C,M,_,attribute) |
| M. | renameObject(P,C,_,_,_,class,PR1) ↔ deleteObject(P,C,_,_,_,class) |
| N. | addObject(P,C,M,_,_,_,PLT,ObjT) ↔ deleteObject(P,C,M,PR,PLT,ObjT) |
| O. | addRelation(E,P,C,M,PR,PLT,ObjT,P1,C1,M1,PR1,PLT1,ObjT1,RT) ↔ renameRelation(E,P,C,M,PR,PLT,ObjT,P1,C1,M1,PR1,PLT1,ObjT1,RT) |
| P. | deleteRelation(E,P,C,M,PR,PLT,ObjT,P1,C1,M1,PR1,PLT1,ObjT1,RT) ↔ renameRelation(_E,P,C,M,PR,PLT,ObjT,P1,C1,M1,PR1,PLT1,ObjT1,RT,E) |
| Q. | renameRelation(E1,P,C,M,PR,PLT,ObjT,P1,C1,M1,PR1,PLT1,ObjT1,RT,E2) ↔ renameRelation(E2,P,C,M,PR,PLT,ObjT,P1,C1,M1,PR1,PLT1,ObjT1,RT,E3) |
Appendix B

PRIMITIVE REFACTORINGS AS FGT COLLECTIONS

This Appendix is a continuation of the discussion in chapter 5 that elaborates on the feasibility of representing primitive refactorings as a sequence of FGTs. Here we focus on primitive refactorings that map to a single FGT.

B.1 Add Element Refactorings

B.1.1 addClass\((ClassName, AccessMode)\)

Where \(ClassName\) has the following format: \(Pn.Cn\) (\(Pn\) is the name of the package and \(Cn\) is the name of the class).

Description

The refactoring creates a new class \(Cn\) with access mode \(AccessMode\) in the package \(Pn\), the created class will be empty and standalone (no members, super or subclasses).

Precondition Conjuncts

(1) The name of the new class \(Cn\) is distinct from those all classes declared already in the package \(Pn\).

(2) The access mode for the new class is a valid access mode for classes.

FGT-List

1. addObject\((Pn, Cn, _, _, _, AccessMode, _, class)\)

Note

Precondition conjuncts (1) and (2) are covered by precondition conjuncts of FGT 1 (section 4.2.1.1.A). There is no need to add precondition conjuncts at the refactoring-level.
B.1.2 addMethod(MethodName, ReturnDType, AccessMode, ParameterList)

Where

- MethodName has the following format: Pn.Cn.Methn
- ReturnDType has the following format: type(Type, Tname, Num)
- ParameterList has the following format: [(Prm1.type(Type1,Tname1,Num1)), (Prm2.type(Type2,Tname2,Num2)), …, (Prm_n.type(Type_n,Tname_n,Num_n))], where each item (Prm_i, type(Type_i, Tname_i, Num_i)) in the list represent information about a parameter defined in the method. The description of arguments of each item is as follows:
  - Prm_i is the name of the parameter.
  - Type_i is the definition type of the parameter (basic or complex).
  - Tname_i is the type name (int, float,…).
  - Num_i is the size of the array. (Zero if the parameter is not array).

Description

The refactoring creates a new method Methn with a list of parameters ParameterList in the class Pn.Cn. The method will have access mode AccessMode and return type ReturnDType.

Precondition Conjuncts

(1) The signature of the new method is distinct from those all methods declared already in the class Pn.Cn or any of its ancestors.

(2) Each parameter name is distinct from all other parameter's name in the parameter list ParameterList.

(3) The definition type of the return value of the method is valid and accessible.

(4) The access mode of the method is valid.

FGT-List

1. addObject(Pn, Cn, Methn, __, ReturnDType, AccessMode, ParameterList, method)

Note

Precondition conjuncts (1), (2), (3) and (4) are covered by precondition conjuncts of FGT 1 (section 4.2.1.1.B). There is no need to add precondition conjuncts at the refactoring-level.
B.1.3 **addAttribute**(AttributeName, AttributeDType, AccessMode)

Where

- **AttributeName** has the following format: *Pn.Cn.Attn*
- **AttributeDType** has the following format: *type(Type, Tname, Num)*

**Description**

The refactoring creates a new attribute *Attn* in the class *Pn.Cn* with access mode *AccessMode*. The definition type of the new attribute will be *AttributeDType*.

**Precondition Conjuncts**

(1) The name of the new attribute is distinct from those all attributes declared already in the class *Pn.Cn* or any of its ancestors.

(2) The definition type of the attribute is valid and accessible.

(3) The access mode of the attribute is valid.

**FGT-List**

1. addObject(*Pn, Cn, Attn,_,_, AttributeDType, AccessMode ,_, attribute*)

**Note**

The precondition conjuncts (1), (2) and (3) are covered by the set of precondition conjuncts of the FGT 1 (section 4.2.1.1.C). There is no need to add precondition conjuncts at the refactoring-level.

B.1.4 **addParameter**(Prmname, PrmDType, Index, MethTList)

Where

- **Prmname** has the following format: *Pn.Cn.Methn.Prmn*
- **PrmDType** has the following format: *type(Type, Tname, Num)*
- **MethTList**: [*Tname1, Tname2,….., TnameN*], where each item *Tnamei* in the list represent the name of the definition type (*int, float, ….*) of one of the parameters defined in the method *Methn* in the same order as defined in the method. The list is used in addition to the name of the method to specify the signature of the method *Methn*. 
Description

The refactoring declares a new parameter $Prmn$ in the method $Pn.Cn.Methn$ with $MethTList$. The type of the new parameter is defined by the variable $PrmDType$. The new parameter will be added at the index $Index$ of the list of the method parameters. If that $Index$ is occupied then all the parameters from the $Index$ will be shifted one-step to the right.

Precondition Conjuncts

(1) The parameter name is distinct in the method's parameters list.

(2) The produced method signature must be distinct from all those methods define in the class $Pn.Cn$ or any of its ancestors.

(3) The parameter definition type should be valid and accessible.

FGT-List

1. addObject($Pn, Cn, Methn, Prmn, Index, PrmDType, \_MethTList, parameter$)

Note

The precondition conjuncts (1), (2) and (3) are covered by the set of precondition conjuncts of the FGT 1 (section 4.2.1.1.D). There is no need to add precondition conjuncts at the refactoring-level.

B.2 Rename Element Refactorings

B.2.1 renameClass($ClassName, NewName$)

Where $ClassName$ has the following format: $Pn.Cn$

Description

The refactoring changes the name of the class $Pn.Cn$ to a new name $Pn.NewName$. The renameClass refactoring is a behaviour-preserving refactoring because changing the name of the class will not have any effect on the behaviour of the system.
Precondition Conjuncts

(1) The new name of the class NewName should not clash with any other class names declared in the package Pn.

FGT-List

1. renameObject(Pn, Cn, _, _, _, class, NewName)

Note

- Precondition conjunct (1) is covered by precondition conjuncts of the FGT 1 (section 4.2.1.2.A). There is no need to add precondition conjuncts at the refactoring-level.

B.2.2 renameMethod(MethodName, MethTList, NewName)

Where

- MethodName has the following format: Pn.Cn.Methn
- MethTList has the following format: [Tname1, Tname2, ..., Tname_n]

Description

The refactoring changes the name of the method Methn with parameter list MethTList defined in the class Pn.Cn to another name Pn.Cn.NewName. The renameMethod refactoring is a behaviour-preserving refactoring because changing the name of the method will not have any effect on the behaviour of the system.

Precondition Conjuncts

(1) The signature of the method with the new name should not clash with the signature of other methods declared in the class Pn.Cn or any of its ancestors.

FGT-List

1. renameObject(Pn, Cn, Methn, _, MethTList, method, NewName)

Note

- Precondition conjunct (1) is covered by precondition conjuncts of FGT 1 (section 4.2.1.2.B). There is no need to add precondition conjuncts at the refactoring-level.
B.2.3 renameAttribute(AttributeName, NewName)

Where AttributeName: Pn.Cn.Attn

Description

The refactoring changes the name of the attribute Attn declared in the class Pn.Cn to another name Pn.Cn.NewName. The renameAttribute refactoring is a behaviour-preserving refactoring because changing the name of the attribute will not have any effect on the behaviour of the system.

Precondition Conjuncts

(1) The new name of the attribute NewName should not clash with any other attributes names declared in the class Pn.Cn or any of its ancestors.

FGT-List

1. renameObject(Pn, Cn, Attn, _, _ attribute, NewName)

Note

A precondition (1) is covered by precondition conjuncts of FGT 1 (section 4.2.1.2.C). There is no need to add precondition conjuncts at the refactoring-level.

B.2.4 renameParameter(ParameterName, MethTList, NewName)

Where

- ParameterName has the following format: Pn.Cn.Methn.Prnn
- MethTList has the following format: [Tname1, Tname2,….., Tname_n]

Description

The refactoring changes the name of the parameter Prnn declared in the method Methn with parameter list MethTList to another name NewName. The renameParameter refactoring is a behaviour-preserving refactoring because changing the name of the parameter will not have any effect on the behaviour of the system.
**Precondition Conjuncts**

(1) The parameter's new name should not clash with the names of those parameters that are declared in the method $Pn.Cn.Methn$ with $MethTList$.

**FGT-List**

1. $\text{renameObject}(Pn, Cn, Methn, Prmn, MethTList, \text{parameter}, \text{NewName})$

**Note**

Precondition conjunct (1) is covered by precondition conjuncts of FGT 1 (section 4.2.1.2.D). There is no need to add precondition conjuncts at the refactoring-level.

**B.3 Change Characteristics Refactorings**

**B.3.1 changeClassAccess($ClassName$, $New\text{Access}$)**

*Where* $ClassName$ has the following format: $Pn.Cn$

**Description**

The refactoring changes the class $ClassName$ access mode.

**Precondition Conjuncts**

(1) In the case of changing the access mode of the class $Cn$ from a lower restriction access mode to a higher restriction one, all the references made by other object elements in the system to the class before the refactoring should be within the scope of the class after the refactoring. Since changing the access mode of the class does not affect any of the references to it, this refactoring will not change the behaviour of the system.

**FGT-List**

1. $\text{changeOAMode}(Pn, Cn, _\_\_\_, _\_\_\_, class, OOldAM, New\text{Access})$
Note

Precondition conjunct (1) is covered by precondition conjuncts of FGT 1 (section 4.2.1.3.B). There is no need to add precondition conjuncts at the refactoring-level. To retrieve the current access mode of the class we use the procedure \texttt{objectAMode}(Pn, Cn, class, OOldAM).

B.3.2 \texttt{changeMethodAccess} (MethodName, MethTList, NewAccess)

Where

- \textit{MethodName} has the following format: \texttt{Pn.Cn.Methodn}
- \textit{MethTList} has the following format: \texttt{[Tname1, Tname2,....., Tname_n]}

Description

The refactoring changes the access mode of the method.

Precondition Conjunctions

(1) In the case of changing the access mode of the method \textit{Methodn} from a lower restriction access mode to a higher restriction one, all the references made by other \texttt{object} elements in the system to the method before the refactoring should be within the scope of the method after the refactoring. Since changing the access mode of the method does not affect any of the references to it, this refactoring will not change the behaviour of the system.

FGT-List

1. \texttt{changeOAMode}(Pn, Cn, Methodn,\_, MethTList, method, OOldAM, NewAccess)

Note

The precondition of this refactoring is covered by precondition conjuncts of FGT 1 (section 4.2.1.3.B). There is no need to add precondition conjuncts at the refactoring-level. To retrieve the current access mode of the method we use the procedure \texttt{objectAMode}(Pn, Cn, Methodn, MethTList, method, OOldAM).

B.3.3 \texttt{changeAttributeAccess} (AttributeName, NewAccess)

Where \textit{AttributeName} has the following format: \texttt{Pn.Cn.Attn}
Description

The refactoring changes the access mode of the attribute.

Precondition Conjuncts

(1) In the case of changing the access mode of the attribute \textit{Attn} from a lower restriction access mode to a higher restriction one, all the references made by other object elements in the system to the attribute before the refactoring should be within the scope of the attribute after the refactoring. Since changing the access mode of the attribute does not affect any of the references to it, this refactoring will not change the behaviour of the system.

FGT-List

1. \textit{changeOAMode}(Pn, Cn, Attn, attribute, OOldAM, NewAccess)

Note

The precondition of this refactoring is covered by precondition conjuncts of FGT 1 (section 4.2.1.3.C). There is no need to add precondition conjuncts at the refactoring-level. To retrieve the current access mode of the attribute we use the procedure \textit{objectAMode}(Pn, Cn, Attn, OOldAM).

B.3.4 \textit{changeMethodReturnType}(Methodname, MethTList, NewRType)

Where

- \textit{MethodName} has the following format: \textit{Pn.Cn.Methn}
- \textit{MethTList} has the following format: [\textit{Tname}_1, \textit{Tname}_2, …, \textit{Tname}_n]
- \textit{NewRType} has the following format: \textit{type(Type, Tname, Num)}

Description

The refactoring changes the definition type of the return value of the method.

Precondition Conjuncts

(1) The method \textit{Pn.Cn.Methn} with \textit{MethTList} should be defined in the system.

(2) The \textit{NewRType} should be valid and accessible.
FGT-List

1. `changeODefType(Pn, Cn, Methn, _, MethTList, method, OldRType, NewRType)`

Note

Precondition conjuncts (1) and (2) of this refactoring are covered by precondition conjuncts of FGT 1 (section 4.2.1.4.A). There is no need to add precondition conjuncts at the refactoring-level. To retrieve the current definition type of the return value of the method we use the procedure `objectDType(Pn, Cn, Methn, MethTList, method, OldRType)`.

B.3.5 `changeAttributeDefType(AttributeName, NewDType)`

Where

- `AttributeName` has the following format: `Pn.Cn.Attn`
- `NewDType` has the following format: `type(Type, Tname, Num)`

Description

The refactoring changes the definition type of the attribute.

Precondition Conjuncts

(1) The attribute `Pn.Cn.Attn` should be defined in the system.

(2) The `NewDType` should be valid and accessible.

FGT-List

1. `changeODefType(Pn, Cn, Attn, _, _, attribute, OldDType, NewDType)`

Note

Precondition conjuncts (1) and (2) of this refactoring are covered by precondition conjuncts of FGT 1 (section 4.2.1.4.B). There is no need to add precondition conjuncts at the refactoring-level. To retrieve the current definition type of the return value of the method we use the procedure `objectDType(Pn, Cn, Attn, attribute, OldDType)`.
B.3.6 changeParameterDefType(ParameterName, MethTList, NewDType)

Where

- ParameterName has the following format: Pn.Cn.Methn.Prmn
- MethTList has the following format: [Tname1, Tname2, ..., Tname_n]
- NewDType has the following format: type(Type, Tname, Num)

Description

The refactoring changes the definition type of one of the parameters of the Methn.

Precondition Conjunctions

(1) The parameter Prmn should be declared in the method Pn.Cn.Methn with MethTList.

(2) The NewDType should be valid and accessible.

FGT-List

1. changeODefType(Pn, Cn, Methn, Prmn, MethTList, parameter, OldDType, NewDType)

Note

Precondition conjunctions (1) and (2) of this refactoring are covered by precondition conjunctions of FGT 1 (section 4.2.1.4.C). There is no need to add precondition conjunctions at the refactoring-level. To retrieve the current definition type of the return value of the method use the procedure objectDType(Pn, Cn, Methn, Prmn, MethTList, parameter, OldDType).

B.4 Delete Element Refactorings

B.4.1 deleteMethod(MethodName, MethTList)

Where

- MethodName has the following format: Pn,Cn, Methn
- MethTList has the following format: [Tname1, Tname2, ..., Tname_n]

Description
The refactoring deletes unreferenced method \textit{Methn} with the parameter list \textit{MethTList} from the class \textit{Pn.Cn}

\textbf{Precondition Conjuncts}

(1) The method \textit{Methn} with the parameter list \textit{MethTList} should be declared in the class \textit{Pn.Cn}.

(2) The method is unreferenced by any other \textit{object} elements.

(3) If the method is inherited by subclasses of the class \textit{Pn.Cn} then the method also should be unreferenced by any instances of these classes.

\textbf{FGT-List}

1. \texttt{deleteObject(Pn, Cn, Methn, \_ \_ MethTList, method)}

\textbf{Note}

- Precondition conjuncts (1), (2) and (3) are covered by the set of precondition conjuncts of the FGT 1 (section 4.2.1.1.B). There is no need to add precondition conjuncts at the refactoring-level of this refactoring.

\textbf{B.4.2 deleteAttribute(AttributeName)}

\textbf{Where} \textit{AttributeName} has the following format: \textit{Pn.Cn.Attn}

\textbf{Description}

The refactoring deletes unreferenced attribute \textit{Attn} from the class \textit{Pn.Cn}.

\textbf{Precondition Conjuncts}

(1) The attribute \textit{Attn} should be declared in the class \textit{Pn.Cn}.

(2) The attribute is unreferenced by any other \textit{object} elements.

(3) If the attribute \textit{Attn} is inherited by subclasses of the class \textit{Pn.Cn} then the attribute \textit{Attn} should not be referenced by any instances of these classes.

\textbf{FGT-List}

1. \texttt{deleteObject(Pn, Cn, Attn, \_ \_ attribute)
Note

The precondition conjuncts (1), (2) and (3) are covered by the set of precondition conjuncts of
the FGT 1 (section 4.2.1.5.C). There is no need to add precondition conjuncts at the
refactoring-level.

B.4.3 deleteParameter(Prmname, MethTList)

Where

- Prmname has the following format: $Pn.Cn.Methn.Prnn$
- MethTList has the following format: $[Tname_1, Tname_2, ..., Tname_n]$

Description

The refactoring removes the parameter Prmn from the parameter’s list of the method Methn.

This refactoring is beneficial when, for example, a method’s purpose is changed and there is a
need to remove (and perhaps later add) parameters from the method.

Precondition Conjuncts

(1) The parameter should be declared in the method.

(2) The produced method signature after removing the parameter should not be declared in the
class $Pn.Cn$ or in any of its ancestors.

FGT-List

1. deleteObject($Pn, Cn, Methn, Prmn, MethTList, parameter$)

Note

The precondition conjuncts (1) and (2) are covered by the set of precondition conjuncts of the
FGT 1 (section 4.2.1.5.D). There is no need to add precondition conjuncts at the refactoring-
level.
BIBLIOGRAPHY


[35] JTransformer homepage http://roots.iai.uni-bonn.de/research/jtransformer/


214


