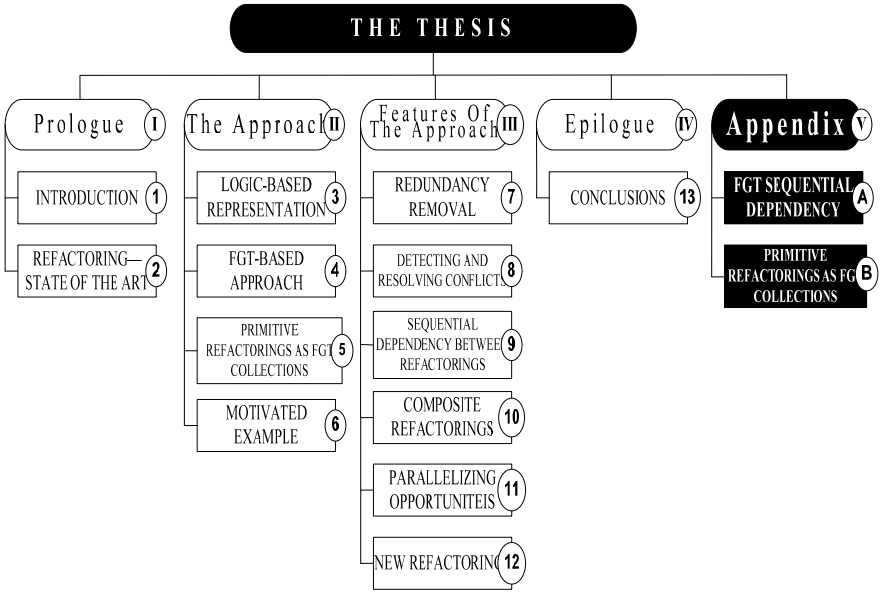
Part V

Appendix

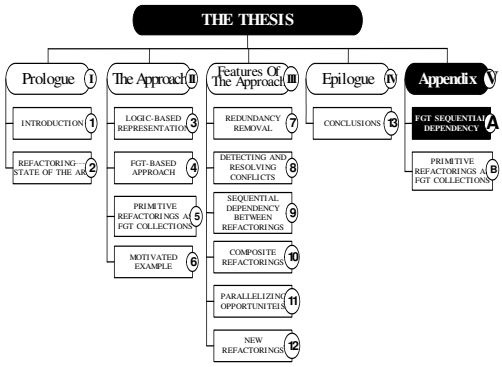


Appendix A

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: $FGT_x \rightarrow FGT_y$, where 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 uni-directional sequential dependency.



1.	changeODefType(*P,C,M,PR,PLT,ObjT,_,ONewDT*) → changeODefType(*P,C,M,PR,PLT, ObjT,ONewDT,_*)
2.	changeOAMode(*P,C,M,PR,PLT,ObjT,_,ONewAM*) → changeOAMode(*P,C,M,PR,PLT,ObjT,ONewAM,_*)
3.	deleteRelation(*_,P,C,M,PR,PLT,Ftype,_,_,_,_,_,_,_*) → changeODefType(*P,C,M,PR,PLT,Ftype,_,_*)
4.	deleteRelation(*_,_,_,_,_,_,_,P,C,M,PR,PLT,Totype,_*) → changeODefType(*P,C,M,PR,PLT,Totype ,_,_*)
5.	deleteRelation(*_,P,C,M,PR,PLT,Ftype,_,_,_,_,_,_,_*) → changeOAMode(*P,C,M,PR,PLT,Ftype,_,_*)
6.	deleteRelation(*_,_,_,_,_,_,_,P,C,M,PR,PLT,Totype,_*) → changeOAMode(*P,C,M,PR,PLT,Totype ,_,_*)
7.	deleteRelation(*_,P,C,M,PR,PLT,Ftype,_,_,_,_,_,_*) → deleteObject(*P,C,M,PR,PLT,Ftype*)
8.	deleteRelation(*_,_,_,_,_,_,_,P,C,M,PR,PLT,Totype,_*) → deleteObject(*P,C,M,PR,PLT,Totype*)
9.	changeODefType(*P,C,M,PR,PLT,ObjT,_,_*) → addRelation(*_,P,C,M,PR,PLT,ObjT,_,_,_,_,_,_,_*)
10.	changeODefType(*P,C,M,PR,PLT,ObjT,_,_*) → addRelation(*_,_,_,_,_,_,_,P,C,M,PR,PLT,ObjT ,_*)
11.	changeOAMode(*P,C,M,PR,PLT,ObjT,_,_*) → addRelation(*_,P,C,M,PR,PLT,ObjT,_,_,_,_,_,_,_*)
12.	changeOAMode(*P,C,M,PR,PLT,ObjT,_,_*) → addRelation*(_,_,_,_,_,_,_,P,C,M,PR,PLT,ObjT,_*)
13.	renameObject(*P,C,M,PR,PLT,parameter,X*) → addRelation(*_,P,C,M,X,PLT,parameter,_,_,_,_,_,_,_*)
14.	renameObject(*P,C,M,_,_,attribute,X*) → addRelation(*_,P,C,X,_,_,attribute,_,_,_,_,_,_,_*)
15.	renameObject(*P,C,M,_,PLT,method,X*) → addRelation(*_,P,C,X,_,PLT,method,_,_,_,_,_,_,_*)
16.	renameObject(*P,C,_,_,_,class,X*) → addRelation(*_,P,X,_,_,_,class,_,_,_,_,_,_,_*)
17.	renameObject(*P,C,M,PR,PLT,parameter,X*) → addRelation(*_,_,_,_,_,_,_,P,C,M,X,PLT,parameter,_*)
18.	renameObject(*P,C,M,_,_,attribute,X*) → addRelation(*_,_,_,_,_,_,_,P,C,X,_,_,attribute,_*)
19.	renameObject(*P,C,M,_,PLT,method,X*) → addRelation(*_,_,_,_,_,_,_,_,P,C,X,_,PLT,method,_*)
20.	renameObject(*P,C,_,_,_,class,X*) → addRelation(*_,_,_,_,_,_,_,_,P,X,_,_,_,class,_*)
21.	renameObject(*P,C,M,PR,PLT,parameter,PR1*) → deleteRelation(*_,P,C,M,PR1,PLT,parameter,_,_,_,_,_,_,_*)
22.	renameObject(*P,C,M,_,_,attribute,PR1*) → deleteRelation(*_,P,C,PR1,_,_,attribute,_,_,_,_,_,_,_*)
23.	renameObject(*P,C,M,_,PLT,method,PR1*) → deleteRelation(*_,P,C,PR1,_,PLT,method,_,_,_,_,_,_,_*)
24.	renameObject(*P,C,_,_,_,class,PR1*) → deleteRelation(*_,P,PR1,_,_,_,class,_,_,_,_,_,_,_*)

25. renameObject(*P,C,M,PR,PLT,parameter,PR1*) → deleteRelation(_,_,_,_,_,_,_,*P,C,M,PR1,PLT,parameter*,_)
26. renameObject(*P,C,M,PR*,_,*attribute,PR1*) → deleteRelation(_,_,_,_,_,_,_,*P,C,PR1*,_,_,*attribute*,_)
27. renameObject(*P,C,M,PR,PLT,method,PR1*) → deleteRelation(_,_,_,_,_,_,_,*P,C,PR1*,_,*PLT,method*,_)
28. renameObject(*P,C*,_,_,_,*class,PR1*) → deleteRelation(_,_,_,_,_,_,_,*P,PR1*,_,_,_,*class*,_)
29. renameObject(*P,C,M,PR,PLT,parameter,X*) → changeODefType(*P,C,M,X,PLT,parameter*,_,_)
30. renameObject(*P,C,M*,_,_,*attribute,X*) → changeODefType(*P,C,X*,_,_,*attribute*,_,_)
31. renameObject(*P,C,M*,_,*PLT,method,X*) → changeODefType(*P,C,X*,_,*PLT,method*,_,_)
32. renameObject(*P,C*,_,_,_,*class,X*) → changeODefType(*P,X*,_,_,_,*class*,_,_)
33. renameObject(*P,C,M,PR,PLT,parameter,X*) → changeOAMode(*P,C,M,X,PLT,parameter*,_,_)
34. renameObject(*P,C,M*,_,_,*attribute,X*) → changeOAMode(*P,C,X*,_,_,*attribute*,_,_)
35. renameObject(*P,C,M*,_,*PLT,method,X*) → changeOAMode(*P,C,X*,_,*PLT,method*,_,_)
36. renameObject(*P,C*,_,_,_,*class,X*) → changeOAMode(*P,X*,_,_,_,*class*,_,_)
37. renameObject(*P,C*,_,_,_,*class,PR1*) → addObject(*P,PR1,M*,_,_,_,_,_,*attribute*)
38. renameObject(*P,C*,_,_,_,*class,PR1*) → addObject(*P,PR1,M*,_,_,_,_,*[],method*)
39. renameObject(*P,C,M*,_,*PLT,method,PR1*) → addObject(*P,C,PR1,PR*,_,_,_,*PLT,parameter*)
40. renameObject(*P,C*,_,_,_,*class,PR1*) → changeOAMode(*P,PR1,M*,_,_,*attribute*,_,_)
41. renameObject*(P,C*,_,_,_,*class,PR1*) → changeOAMode(*P,PR1,M*,_,*PLT,method*,_,_)
42. renameObject(*P,C,M*,_,*PLT,method,PR1*) → changeOAMode(*P,C,PR1,PR,PLT,parameter*,_,_)
43. renameObject(*P,C*,_,_,_,*class,PR1*) → changeODefType(*P,PR1,M*,_,_,*attribute*,_,_)
44. renameObject(*P,C*,_,_,_,*class,PR1*) → changeODefType(*P,PR1,M*,_,*PLT,method*,_,_)
45. renameObject(*P,C,M*,_,*PLT,method,PR1*) → changeODefType(*P,C,PR1,PR,PLT,parameter*,_,_)
46. renameObject(*P,C*,_,_,_,*class,PR1*) → deleteObject(*P,PR1,M*,_,_,*attribute*)
47. renameObject(*P,C*,_,_,_,*class,PR1*) → deleteObject(*P,PR1,M*,_,*PLT,method*)
48. renameObject(*P,C,M*,_,*PLT,method,PR1*) → deleteObject(*P,C,PR1,PR,PLT,parameter*)
49. renameObject(*P,C*,_,_,_,*class,PR1*) → addRelation(_,*P,PR1,M*,_,_,*attribute*,_,_,_,_,_,_,_)
50. renameObject(*P,C*,_,_,_,*class,PR1*) → addRelation(_,_,_,_,_,_,*P1,PR1,M*,_,_,*attribute*,_)
51. renameObject(*P,C*,_,_,_,*class,PR1*) → addRelation(_,*P,PR1,M*,_,*PLT,method*,_,_,_,_,_,_,_)
52. renameObject(*P,C*,_,_,_,*class,PR1*) → addRelation(_,_,_,_,_,_,*P1,PR1,M*,_,*PLT,method*,_)
53. renameObject(*P,C,M*,_,*PLT,method,PR1*) → addRelation(_,*P,C,PR1,PR,PLT,parameter*,_,_,_,_,_,_,_)
54. renameObject(*P,C,M*,_,*PLT,method,PR1*) → addRelation(_,_,_,_,_,_,*P,C,PR1,PR,PLT,parameter*,_)
55. renameObject(*P,C*,_,_,_,*class,PR1*) → deleteRelation(_,*P,PR1,M*,_,_,*attribute*,_,_,_,_,_,_,_)
56. renameObject(*P,C*,_,_,_,*class,PR1*) → deleteRelation(_,_,_,_,_,_,*P1,PR1,M*,_,_,*attribute*,_)
57. renameObject(*P,C*,_,_,_,*class,PR1*) → deleteRelation(_,*P,PR1,M*,_,*PLT,method*,_,_,_,_,_,_,_)
58. renameObject(*P,C*,_,_,_,*class,PR1*) → deleteRelation(_,_,_,_,_,_,*P1,PR1,M*,_,*PLT,method*,_)
59. renameObject(*P,C,M*,_,*PLT,method,PR1*) → deleteRelation(_,*P,C,PR1,PR,PLT,parameter*,_,_,_,_,_,_,_)
60. renameObject(*P,C,M*,_,*PLT,method,PR1*) → deleteRelation(_,_,_,_,_,_,*P,C,PR1,PR,PLT,parameter*,_)
61. addObject(*P,C*,_,_,_,_,_,*PLT,class*) → addObject(*P,C,M*,_,_,_,_,*PLT,attribute*)
62. addObject(*P,C*,_,_,_,_,_,*PLT,class*) → addObject(*P,C,M*,_,_,_,_,*PLT,method*)
63. addObject(*P,C,M*,_,_,_,_,*PLT,method*) → addObject(*P,C,M,PR*,_,_,_,*PLT,parameter*)
64. addObject(*P,C,M,PR*,_,_,_,*PLT,ObjT*) → addRelation(_,*P,C,M,PR,PLT,ObjT*,_,_,_,_,_,_)
65. addObject(*P1,C1,M1,PR1*,_,_,_,*PLT,ObjT*) → addRelation(_,_,_,_,_,_,*P1,C1,M1,PR1,PLT,ObjT*,_)
66. addObject(*P,C,M,PR,Oldtype*,_,_,*PLT,ObjT*) → changeODefType(*P,C,M,PR,PLT,ObjT,Oldtype,Newtype*)
67. addObject(*P,C,M,PR*,_,_,*Oldmode,PLT,ObjT*) → changeOAMode(*P,C,M,PR,PLT,ObjT,OldMd, NewMd*)
68. deleteObject(*P,C,M,PR,PLT,parameter*) → deleteObject(*P,C,M*,_,*PLT,method*)
69. deleteObject(*P,C,M*,_,*PLT,method*) → deleteObject(*P,C*,_,_,*PLT,class*)
70. deleteObject(*P,C,M*,_,*PLT,attribute*) → deleteObject(*P,C*,_,_,*PLT,class*)
71. renameObject(*P,C,M,PR,PLT,parameter,X*) → renameRelation(_,*P,C,M,X,PLT,parameter*,_,_,_,_,_,_,_,_)
72. renameObject(*P,C,M*,_,_,*attribute,X*) → renameRelation (_,*P,C,X*,_,_,*attribute*,_,_,_,_,_,_,_,_)
73. renameObject(*P,C,M*,_,*PLT,method,X*) → renameRelation (_,*P,C,X*,_,*PLT,method*,_,_,_,_,_,_,_,_)
74. renameObject(*P,C*,_,_,_,*class,X*) → renameRelation (_,*P,X*,_,_,_,*class*,_,_,_,_,_,_,_,_)

75.	renameObject(*P,C,M,PR,PLT,parameter,X*) → renameRelation (*_,_,_,_,_,_,_,P,C,M,X,PLT,parameter,_,_*)
76.	renameObject(*P,C,M,_,_,attribute,X*) → renameRelation (*_,_,_,_,_,_,P,C,X,_,_,attribute,_,_*)
77.	renameObject(*P,C,M,_,PLT,method,X*) → renameRelation (*_,_,_,_,_,_,_,P,C,X,_,PLT,method,_,_*)
78.	renameObject(*P,C,_,_,_,class,X*) → renameRelation (*_,_,_,_,_,_,_,P,X,_,_,_,class,_,_*)
79.	* deleteObject(*P1,C1,M,_,_,attribute*) → addObject(*P2,C2,M,_,_,_,_,_,attribute*)
80.	* deleteObject(*P1,C1,M,_,PLT,method*) → addObject(*P2,C2,M,_,_,_,_,PLT,method*)
81.	* deleteObject(*P1,C1,M,_,_,attribute*) → renameObject(*P2,C2,X,_,_,attribute,M*)
82.	* deleteObject(*P1,C1,M,_,PLT,method*) → renameObject(*P2,C2,X,_,PLT,method,M*)
83.	* renameObject(*P1,C1,M,_,_,attribute,X*) → addObject(*P2,C2,M,_,_,DefType, AMode,_,attribute*)
84.	* renameObject(*P1,C1,M,_,PLT,method,X*) → addObject(*P2,C2,M,_,_,DefType, AMode,PLT,method*)

** Note: Assume P1.C1 is one of the ancestor's of P2.C2*

A.2 Bi-Directional FGTs Sequential Dependencies

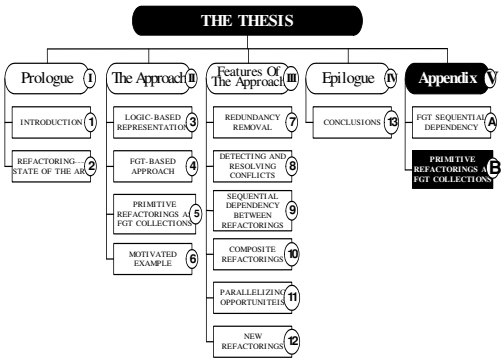
In the table below, all the bi-directional sequential dependencies between the different FGTs proposed in this thesis are catalogued. The information in the table is a continuation of the discussion in section 4.3.3. Each row in the table represents the following sequential dependencies: $FGT_x \leftrightarrow FGT_y$ where FGT_y is sequentially dependent on FGT_x and FGT_x is sequentially dependent on FGT_y. 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*)
B.	renameObject(*P,C,M,PR,PLT,parameter,X*) ↔ renameObject(*P,C,M,X,PLT,parameter,Y*)
C.	renameObject(*P,C,M,_,_,attribute,X*) ↔ renameObject(*P,C,X,_,_,attribute,Y*)
D.	renameObject(*P,C,M,_,PLT,method,X*) ↔ renameObject(*P,C,X,_,PLT,method,Y*)
E.	renameObject(*P,C,_,_,_,class,X*) ↔ renameObject(*P,X,_,_,_,class,Y*)
F.	renameObject(*P,C,M,PR,PLT,parameter,PR1*) ↔ deleteObject(*P,C,M,PR1,PLT,parameter*)
G.	renameObject(*P,C,M,_,_,attribute,M1*) ↔ deleteObject(*P,C,M1,_,_,attribute*)
H.	renameObject(*P,C,M,_,PLT,method,M1*) ↔ deleteObject(*P,C,M1,_,PLT,method*)
I.	renameObject(*P,C,_,_,_,class,M1*) ↔ deleteObject(*P,M1,_,_,_,class*)
J.	renameObject(*P,C,M,PR,PLT,parameter,PR1*) ↔ addObject(*P,C,M,PR,_,_,_,PLT,parameter*)
K.	renameObject(*P,C,M,_,_,attribute,PR1*) ↔ addObject(*P,C,M,_,_,_,_,_,attribute*)
L.	renameObject(*P,C,M,_,PLT,method,PR1*) ↔ addObject(*P,C,M,_,_,_,_,PLT,method*)
M.	renameObject(*P,C,_,_,_,class,PR1*) ↔ addObject(*P,C,_,_,_,_,_,_,class*)
N.	addObject(*P,C,M,PR,_,_,_,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 (*_,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: [$(Prm_1,type(Type_1,Tname_1,Num_1))$, $(Prm_2,type(Type_2,Tname_2,Num_2))$, …., $(Prm_n,type(Type_n,Tname_n,Num_n))$], 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(*AttibuteName, 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: [*$Tname_1$*,* $Tname_2$*,…..,* $Tname_n$*]*, where each item $Tname_i$ 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 muse 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: *[Tname*$_1$*, Tname*$_2$*,…., 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.Prmn*
- *MethTList* has the following format: $[Tname_1, Tname_2, \ldots, Tname_n]$

Description

The refactoring changes the name of the parameter *Prmn* 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. renameObject(*Pn, Cn, Methn, Prmn, MethTList, parameter, 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, NewAcces*)

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. changeOAMode(*Pn, Cn, _, _, _, class, OOldAM, NewAcces*)

Note

Precondition conjunct (1) is covered by precondition conjuncts of FGT 1 (*section 4.2.1.3.A*). 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 **objectAMode**(*Pn, Cn, class, OOldAM*).

B.3.2 changeMethodAccess(*Methname, MethTList, NewAccess*)

Where

- *MethodName* has the following format: *Pn.Cn.Methn*
- *MethTList* has the following format: *[Tname$_1$, Tname$_2$,…., Tname$_n$]*

Description

The refactoring changes the access mode of the method.

Precondition Conjuncts

(1) In the case of changing the access mode of the method *Methn* from a lower restriction access mode to a higher restriction one, all the references made by other *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. changeOAMode(*Pn, Cn, Methn,_, 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 **objectAMode**(*Pn, Cn, Methn, MethTList, method, OOldAM*).

B.3.3 changeAttributeAccess(*AttributeName, NewAccess*)

Where *AttributeName* has the following format: *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 *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. 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 **objectAMode**(*Pn, Cn, Attn, OOldAM*).

B.3.4 changeMethodReturnType(*Methodname, MethTList, NewRType*)

Where

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

Description

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

Precondition Conjuncts

(1) The method *Pn.Cn.Methn* with *MethTList* should be defined in the system.

(2) The *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: *[Tname$_1$, Tname$_2$,…., 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 Conjuncts

(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 conjuncts (1) and (2) of this refactoring are covered by precondition conjuncts of FGT 1 (*section 4.2.1.4.C*). 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 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: *[Tname$_1$, Tname$_2$,…., Tname$_n$]*

Description

The refactoring deletes unreferenced method *Methn* with the parameter list *MethTList* from the class *Pn.Cn*

Precondition Conjuncts

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

(2) The method is unreferenced by any other *object* elements.

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

FGT-List

1. deleteObject(*Pn, Cn, Methn, _, MethTList, method*)

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.

B.4.2 deleteAttribute(*AttributeName*)

Where *AttributeName* has the following format: *Pn.Cn.Attn*

Description

The refactoring deletes unreferenced attribute *Attn* from the class *Pn.Cn*.

Precondition Conjuncts

(1) The attribute *Attn* should be declared in the class *Pn.Cn*.

(2) The attribute is unreferenced by any other *object* elements.

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

FGT-List

1. 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.Prmn*
- *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

[1] Arnold, R. (1986). An introduction to software restructuring. *In Tutorial on Software Restructuring*, Robert S. Arnold, Ed. IEEE.

[2] Astels, D. (2002). Refactoring with UML. *In Proc. Int'l Conf. eXtreme Programming and Flexible Processes in Software Engineering* (pp. 67-70).

[3] Back, R. (2002). Software Construction by Stepwise Feature Introduction. *In Didier Bert et. Al., editor, ZB 2002: Formal Specification and Development in Z and B*, volume 2272 of Lecture Notes in Computer Science, pages 162-183.

[4] Banerjee, J., and Kim, W. (1987). Semantics and implementation of schema evolution in object-oriented databases. *In Proc. SIGMOD Conf.*, ACM.

[5] Binkley, D. and Gallagher, K. (1996). Program slicing. *Advances of Computing 43*, pp. 150.

[6] Boger, P., Sturm, T., and Fragemann, P. (2002). Refactoring Browser for UML. *In Proc. 3rd Int'l Conf. on eXtreme Programming and Flexible Processes in Software Engineering*, pages 77-81, Alghero, Sardinia, Italy.

[7] Bottoni, P., Parisi-Presicce, F., and Taentzer, G. (2002). Coordinated distributed diagram transformation for software evolution. *Electronic Notes in Theoretical Computer Science* 72(4).

[8] Cinne'ide, M. (2000). *Automated Application of Design Patterns: A Refactoring Approach*. PhD thesis, Department of Computer Science – Trinity College – Dublin.

[9] Coleman, D. , Ash, D., Lowther, B., and Oman, P. (1994). Using metrics to evaluate software system maintainability. *IEEE Computer*, vol. 27, no. 8, pp. 44–49.

[10] Corradini, A., Ehrig, H., Kreowski, H., and Rozenberg, G. (2002). Graph Transformation. *Lecture Notes in Computer Science 2505*, Springer-Verlag.

[11] Cuny, J., Ehrig, H., Engels, G., and Rozenberg, G. (1996). Graph Grammars and Their Application to Computer Science. *Lecture Notes in Computer Science* 1073, Springer-Verlag.

[12] D'Hondt, M. (1998). *Managing Evolution of Changing Software Requirements*. Dissertation, Department of Computer Science, Vrije Universiteit Brussel.

[13] Demeyer, S. *et al*. (2005). The LAN-simulation: A Refactoring Teaching Example. *Int. Workshop on Principles of Software Evolution (IWPSE)*. pages 123-134.

[14] Deursen, V., and Kuipers, T. (1999). Identifying objects using clusterand concept analysis. *In Proceedings of the 21st International Conference on Software Engineering (ICSE 1999)*, pages 246–255. IEEE Computer Society.

[15] Ecklund E., Lois, M., and Freiling, M. (1996). Change cases: Use cases that identify future requirements. *Proceedings of OOPSLA '96, ACM SIGPLAN Notices*, 31(10), pp. 342-358, ACM Press.

[16] Edwards, W. (1997). Flexible Conflict Detection and Management in Collaborative Applications. *Proc. Symp. User Interface Software and Technology*.

[17] Eetvelde, N., and Janssens, D. (2003). A hierarchical program representation for refactoring. *In Proc. of UniGra'03 Workshop.*

[18] Ehrig, H., Engels, G., Kreowski, J., & Rozenberg, G. (2000). Theory and Application to Graph Transformations. *Lecture Notes in Computer Science* 1764,Springer-Verlag.

[19] Engels, G., Hartmut, E., & Rozenberg, G. (1996). Special Issue on Graph Transformations. *Fundamenta Informaticae* 26 (3,4), IOS Press.

[20] Fanta, R. and Rajlich, V. (1999). Restructuring legacy C code into C++. *In Proc. Int'l Conf. Software Maintenance*, pp. 77-85.

[21] Feather, M. (1989). Detecting Interference when Merging Specification Evaluations. *Proc. Fifth Int'l Workshop Software Specification and Design*, pp. 169-176.

[22] Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

[23] France, R., and James M., (2001). Multi-View Software Evolution: A UML based Framework for Evolving Object-Oriented Software. *In Proceedings of The IEEE Interna- tional Conference on Software Maintenance*.

[24] Ganter, B., and Wille, R. (1999). Formal Concept Analysis: mathematical foundations. *(Translated from the German by Cornelia Franzke) Springer-Verlag*, Berlin-Heidelberg.

[25] Glass, R. (1998). Maintenance: Less is not more. *IEEE Software* 15(4): 67-68.

[26] Gorp, P., Stenten, H., Mens, T., and Demeyer, S. (2003). Towards automating source consistent UML refactorings. *In Proc. UML 2003, vol. 2863 of Lecture Notes in Computer Science*, pp. 144–158, Springer-Verlag.

[27] Griswold, W. (1991). *Program Restructuring as an Aid to Software Maintenance*. PhD thesis, University of Washington.

[28] Griswold, W., and Notkin, D. (1993). Automated assistance for program restructuring. *Trans. Software Engineering and Methodology*, vol. 2, no. 3, pp. 228–269, ACM.

[29] Guimaraes, T. (1983). Managing application program maintenance expenditure. *Comm. ACM*, vol. 26, no. 10, pp. 739–746.

[30] Hannemann, J., and Kiczales, G. (2001). Overcoming the prevalent decomposition of legacy code. *Workshop on Advanced Separation of Concerns at the International Conference on Software Engineering (ICSE), Toronto.* http://www.cs.ubc.ca/jan/papers/ICSE2001-OvercomingDecomposition.pdf .

[31] Heckel, R. (1995). *Algebraic graph transformations with application conditions.* M.S. thesis, TU Berlin.

[32] Hunt, J., and Mclllroy, M. (1976). An Algorithm for Differential File Comparison. *Technical Report 41*, AT&T Bell Laboratories Inc.

[33] Jacobson, I., Christerson, M., Jonsson , P., and Övergaard, G. (1992). *Object Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley.

[34] Jahnke, J., and Zundorf, A. (1997). Rewriting poor design patterns by good design patterns. *In S. Demeyer and H. Gall,editors. Proc. of ESEC/FSE '97 Workshop on Object-Oriented Reengineering*, Technical University of Vienna,Technical Report TUV-1841-97-10.

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

[36] Kempen, M., Chaudron, M., Kourie, D., and Boake, A. (2005). Towards proving preservation of behaviour of refactoring of UML models. *Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, SAICSIT; Vol. 150, South Africa

[37] Kerievsky, J. (2002). *Refactoring To Patterns*. Technical report, Industrial Logic. http://www.industriallogic.com/xp/ refactoring/.

[38] Kniesel, G. & Koch, H. (2004). Static composition of refactorings. *Science of Computer Programming*, 52:9-51.

[39] Kniesel, G. (2006). A logic foundation for conditional program transformations. *Technical report no IAI-TR-2006-01*,ISSN 0944-8535,CS Dept III.

[40] Komondoor, R. and Horwitz, S. (2000). Semantics-preserving procedure extraction. *Technical report, Computer Sciences Department*, University of Wisconsin Madison.

[41] Kramer, J., and Magee J., (1998). Analysing Dynamic Change in Software Architectures. *Proceedings of IWPSE98, International Workshop on Principles of Software Evolution*, Kyoto, Japan.

[42] Lakhotia, A. and Deprez, J. (1998). Restructuring programs by tucking statements into functions. *In: M. Harman and K. Gallagher, editors, Special Issue on Program Slicing, Information and Software Technology 40, Elsevier*, pp. 677-689.

[43] Lanubile, F., and Visaggio, G., (1997). Extracting reusable functions by flow graph-based program slicing. *IEEE Transactions on Software Engineering* 23 4, pp. 246–259.

[44] Leblang, D. and Chase, R. (1984). Computer-Aided Software Engineering in a Distributed Workstation Environment. *Proc. SIGPLAN/SIGSOFT Software Eng. Symp. Practical Software Development Environments*, ACM SIGPLAN Notices, vol. 19, no. 5, pp. 104-112.

[45] Leblang, D., Chase, R. and Spilke, H. (1988). Increasing Productivity with a Parallel Configuration Manager. *Proc. Int'l Workshop Software Version and Configuration Control*. pp. 21-38.

[46] Lehman, M., Belady, L. (1985). Program Evolution, *P*rocesses of software change. *Academic Press Professional, Inc.,* San Diego, CA.

[47] Lientz, B., and Swanson, E. (1980). *Software maintenance management: a study of the maintenance of computer application software in 487 data processing organizations*, Addison-Wesley.

[48] Lubkin, D. (1991). Heterogeneous Configuration Management with DSEE. *Proc. Third Int'l Workshop Software Configuration Management*, pp. 153-160.

[49] Markovic', S. (2004). Composition of UML Described Refactoring Rules. *OCL and Model Driven Engineering, UML 2004 Conference Workshop*. Lisbon, Portugal, Octavian Patrascoiu (Ed.), University of Kent, pp. 45-59.

[50] Mens, T. (2002). A State-of-the-Art Survey on Software Merging. *IEEE Transactions on Software Engineering,* vol. 28 n.5, p. 449-462.

[51] Mens, T. (1999). *A Formal Foundation for Object-Oriented Software Evolution.* PhD Thesis, Dept. Computer Science, Vriji Univ. Brussel, Belgium.

[52] Mens, T. (2001). Transformational Software Evolution by Assertions. Formal Foundations for the Evolution of Hypermedia Systems. *5th European Conference on Software Maintenance and Reengineering, Workshop on FFSE*. IEEE Press. Lisbon, Portugal, pp. 67-74.

[53] Mens, T. (2005). On the use of graph transformations for model refactoring. *In Generative and transformational techniques in software engineering* (J. V. Ralf Lämmel,Joao Saraivaed.), pp. 67–98, Departamento di Informatica,Universidade do Minho.

[54] Mens, T., Demeyer, S., and Janssens, D. (2002). Formalising behaviour preserving program transformations. *In Graph Transformation, Lecture Notes in Computer Science,* vol. 2505, pp. 286-301, Springer-Verlag.

[55] Mens, T., Kniesel, G., and Runge, O. (2006). Transformation dependency analyis: A comparison of two approaches. *Proceedings of Langages et Modèles à Objects* (LMO2006).

[56] Mens, T., Lucas, P., and Steyaert, P. (1998). Supporting reuse and evolution of UML models. *In P.-A. Muller and J. Bézivin, editors, Proceeding of <<UML>>'98 International Workshop*, Mulhouse, France, pages 341-350.

[57] Mens, T., Taentzer, G., and Runge, O. (2005). Detecting structural refactoring conflicts using critical pair analysis. *Electronic Notes in Theoretical Computer Science*, vol. 127, n° 3, p. 113-128

[58] Mens, T., Taentzer, G., and Runge, O. (2006). Analyzing Refactoring Dependencies Using Graph Transformation. *Software and Systems Modeling*.

[59] Mens, T., Tourwe', T. (2004). A survey of software refactoring. *IEEE Transactions on Software Engineering,* vol. 30 n.2, p. 126-139.

[60] Mens, T., Van Eetvelde, N., Demeyer, S., & Janssens, D. (2005). Formalizing refactorings with graph transformations. *Journal on Software Maintenance and Evolution* 17(4), 247–276, Wiley.

[61] Mens, T., Van Gorp, P., ,Varró, D., & Karsai, G. (2005). Applying a model transformation taxonomy to graph transformation technology. *In Proc. Int'l* (GraMoT2005).

[62] Munson, J. and Dewan, P. (1994). A Flexible Object Merging Frame-work. *Proc. ACM Conf. Computer Supported Collaborative Work*, pp. 231-241.

[63] Nagl, M., Sch¨urr, A., and M¨unch, M. (2000). Applications of Graph Transformations with Industrial Relevance. *Lecture Notes in Computer Science,* volume 1779. Springer-Verlag.

[64] Object Management Group (2005). Unified Modeling Language: Infrastructure version 2.0. *Formal/2005-07-05*.

[65] Opdyke, W. (1992). *Refactoring object-oriented frameworks*. Ph.D. thesis. University of Illinois at Urbana-Champaign.

[66] Opdyke, W., & Johnson R. (1993). Creating abstract superclasses by refactoring. *Proceedings ACM Computer Science Conference. ACM Press*, pp. 66-73.

[67] Philipps, J. and Rumpe, B. (1997). Refinement of information flow architectures. *In Proc. ICFEM'97*. IEEE Computer Society.

[68] Porres, I. (2003). Model refactorings as rule-based update transformations. *Proceedings of UML 2003 Conference*, pages 159-174.

[69] Proceedings of International Workshop on Principles of Software Evolution, Kyoto, Japan, 1998. ACM SIG Publication, ACM Press, 1999.

[70] Roberts, D. (1999). *Practical Analysis for Refactoring*. PhD thesis, University of Illinois at Urbana-Champaign.

[71] Roberts, D., Brant, J., & Johnson, R. (1997). A refactoring tool for smalltalk. *Theory and Practice of Object Systems,* vol. 3, no. 4, pp. 253-263.

[72] Russo, A., Nuseibeh, B., and Kramer, J. (1998). Restructuring requirements specifications for managing inconsistency and change: A case study. *In Proc. Int'l Conf. Requirements Engineering*, pp. 51-61, Colorado Spring, USA.

[73] Saadeh, E., Kourie, D. & Boake, A. (2008). Model Refactorings as Logic-Based Fine-Grain Transformations. *Proceedings of the 9th African Conference on Research in Computer Science and Applied Mathematics*, 703-710, ISBN: 2-7261-1299-4, Rabat, Morocco.

[74] Saadeh, E., Kourie, D., & Boake, A. (2008). An Algorithm for Ordering Refactorings Based on Fine-Grained Model Transformations. *7th International Conference on Software Methodologies, Tools and Techniques*, 225-243,ISSN:0922-6389,SOMET08, Sharjah, UAE.

[75] Saadeh, E., Kourie, D., and Boake, A. (2009). Fine-grain Transformations to Refactor UML Models. *Proceedings of The Warm Up Workshop for ACM/IEEE ICSE 2010*, 45-51, ACM ISBN: 978-1-60558-565-9, Cape Town, South Africa.

[76] Simon, F., Steinbruckner, F., and Lewerentz, C. (2001). Metrics based refactoring. *Proc European Conf Software Maintenance and Reengineering*, pages 30-38.

[77] Snelting, G., and Tip, F. (1998). Reengineering class hierarchies using concept analysis. *In Proc. Foundations of Software Engineering (FSE-6), SIGSOFT Software Engineering Notes 23(6)*, pp. 99-110.

[78] Soley, R. (2000). Model Driven Architecture. *OMG Document omg*.

[79] Sommerville, (2004). *Software evolution and reengineering*. Software Engineering 7th Edition (Chp 21) Addison-Wesley.

[80] Steyaert, P., Lucas, C., Mens, K. and D'Hondt, T. (1996). Reuse Contracts: Managing the Evolution of Reusable Assets. *Proc. OOPSLA '96, ACM SIGPLAN Notices*, vol. 31, no. 10, pp. 268-286.

[81] Sunyé, G., Pollet, D., Le Traon, Y., & Jézéquel, J.-M. (2001). Refactoring UML models. *In Proc. Int'l Conf. Unified Modeling Language* (pp. 134-138), LNCS 2185, Springer.

[82] Tip, F.(1995). A survey of program slicing techniques. *Journal of Programming Languages 3(3)*, pp. 121-189.

[83] Tokuda, L., and Batory, D. (2001). Evolving object-oriented designs with refactorings. *Automated Software Engineering*, vol. 8, no.1, pp. 89–120.

[84] Tonella, P. (2001). Concept analysis for module restructuring. *Trans. Software Engineering 27(4)*, pp. 351-363.

[85] Tourwe, T. and Mens, T. (2003). Identifying refactoring opportunities using logic meta programming. *Proc 7th European Conf Software Maintenance and Re-engineering* (CSMR2003), IEEE Computer Society Press, pages 91-100.

[86] Ward, M., and Bennett, K. (1995). Formal methods to aid the evolution of software. *Int'l Journal of Software Engineering and Knowledge Engineering*, vol. 5, no. 1, pp. 25–47.

[87] Wermelinger, M. (1998). Software architecture evolution and the chemical abstract machine. *In International Workshop on the Principles of Software Evolution*, pages 93–97, Kyoto, Japan.

[88] Wiels, V., and Easterbrook, S. (1998). Management of Evolving Specifications Using Category Theory. *Proceedings of Automated Software Engineering Conference '98*, pp. 1221, IEEE Press.