University of Zurich[UZH]

Cédric Jeanneret
Martin Glinz
Benoit Baudry

Footprinting Operations Written in Kermeta

TECHNICAL REPORT – No. IFI-2011.0002

April 2011

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

C. Jeanneret
M. Glinz
B. Baudry: Footprinting Operations Written in Kermeta
Technical Report No. IFI-2011.0002, April 2011
Requirements Engineering
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.ifi.uzh.ch/rerg

Footprinting Operations Written in Kermeta

Cédric Jeanneret
University of Zurich, Switzerland
jeanneret@ifi.uzh.ch

Martin Glinz
University of Zurich, Switzerland
glinz@ifi.uzh.ch

Benoit Baudry
IRISA, France
bbaudry@irisa.fr

Abstract

When performed on a model, a set of operations (e.g., queries or model transformations) rarely uses all the information present in the model. Unintended underuse of a model can indicate various problems: the model may contain more detail than necessary or the operations may be immature or erroneous. Analyzing the footprints of the operations — i.e., the part of a model actually used by an operation — is a simple technique to diagnose and analyze such problems. In this report, we present the implementation of static and dynamic footprinting approaches for operations written in Kermeta. We also present the models and model operations used for the comparison of these approaches.

1. Introduction

Software engineers use many kinds of models to develop software. In our work, we are interested in models used for analysis and generation purposes. Such models are mainly used as input to various model operations like queries, simulations, views extractions or model transformations. Increasingly, these operations are automated to improve both the productivity of engineers and the reliability of the analysis.

When executed on a model, a particular model operation rarely touches all elements contained in its input models. The set of elements touched during its execution forms its footprint. In [1], we explain how analyzing footprints can help in detecting problems with a model's scope or level of detail or the presence of faults in an operation. We presented two approaches for footprinting operations: dynamic footprinting reveals the actual footprint of a model operation after its execution, while static footprinting estimates the footprint of an operation before its execution.

To implement and evaluate these ideas, we chose Kermeta[1] [2] as language for the definition of model operations. Kermeta is an executable metamodeling language. It extends EMOF with a statically typed action language. Thus, a metamodel written in Kermeta has two main components: a static part, describing the abstract syntax of a modeling language, and a behavioral part, describing its operational semantics.

This technical report presents the material (implementation of algorithms, models and model operations) that lead to the results presented in [1]. It does not contain a comprehensive introduction to footprints, neither a detailed presentation of our findings.

The remainder of this report is organized as follows. In the next section, we present the implementation of both footprinting approaches: Section 2.1 describes the implementation of dynamic footprinting, while Section 2.2 presents the realization of static footprinting. Section 3 presents the experimental material. More precisely, Section 3.1 focuses on the models and Section 3.2 describes the model operations.

2. Implementation

In [1], we present two approaches for footprinting model operations: Dynamic footprinting requires the execution of the operation (see left part of Figure 1), while static footprinting analyzes its definition (right part of Figure 1). In this section, we present the implementation of both footprinting approaches for model operations written in Kermeta.

1. http://www.kermeta.org

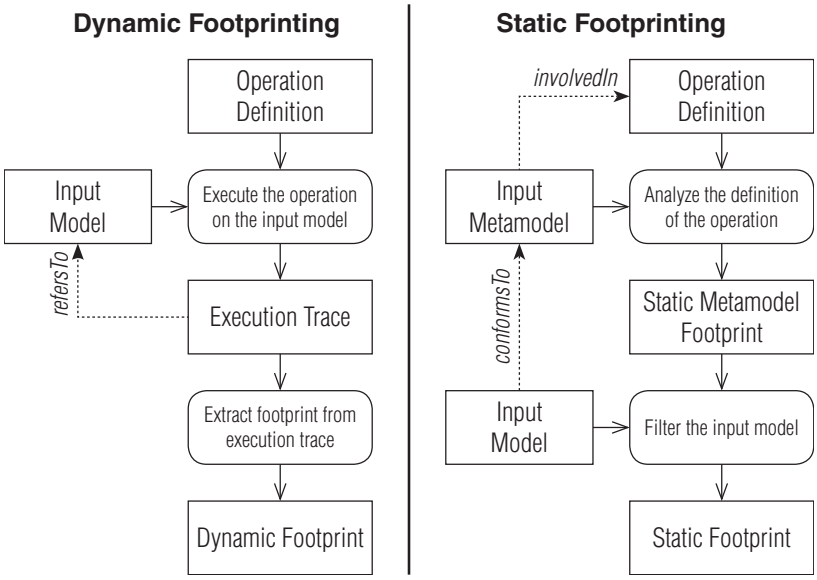

Figure 1. Dynamic (left) and static (right) footprinting

2.1. Dynamic Footprinting

As explained in [1], dynamic footprinting consists of executing the operation on the model while recording a trace of this execution — a set of events that occurred during the execution of an operation. The dynamic (actual) model footprint of an operation can then be established from its execution trace with a simple book-keeping algorithm.

2.1.1. Recording Execution Traces. To compute the dynamic footprint of an operation, we are interested in two kinds of execution events: accesses to the state of objects and invocations of operations on objects. These events occur when Kermeta interprets a `CallFeature` object in the abstract syntax tree of the transformation. To record these events, we modify the class `org.kermeta.interpreter.internal.CallFeatureInterpreter` to store, for every `CallFeature` object being interpreted, the model object targeted by the `CallFeature` and the property or operation being called.

This information can be stored in the `RuntimeMemory` object of the interpreter. We extend this class to store trace events in an EMF model, whose metamodel is shown in Figure 2. Note that this metamodel contains references to both Ecore and Kermeta metamodels. Once the execution of the Kermeta transformation is terminated, the interpreter (an implementation of the `AbstractKInterpreter` class) saves the execution trace (an EMF model) into a file.

2.1.2. Extracting Dynamic Footprints. Extracting dynamic footprints requires an array in two dimensions to keep track of relevant objects and settings. Each row of this array represents an object, while each column represents a structural feature of the input metamodel. The array is initially empty. For every invocation of an operation *op* on an object *obj* in the execution trace, we create a row for *obj* (unless such a row already exists). Similarly, for every access to a structural feature f of an object *obj*, we create a row for *obj* (unless such a row already exists) and a column for f (unless such a column already exists) and we mark the cell at the intersection of the row and the column. Additionally, if f is a reference, we insert all referenced objects into the table. Once the trace has been fully analyzed, the dynamic footprint is the set of objects present in the array and the set of settings marked in the array.

This algorithm and the underlying data structure was implemented in Java. The source code is listed in Appendix B. *RecordTable* implements the data structure and *DynAnalysis* the algorithm.

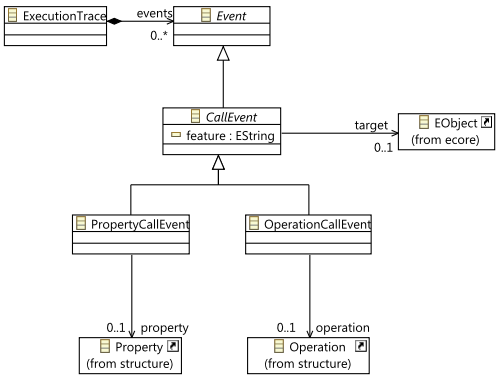

Figure 2. Metamodel for Kermeta execution traces

2.2. Static Footprinting

Static footprinting is achieved in two steps [1]: First, the definition of the operation is analyzed to establish its static metamodel footprint, that is, the set of all metamodel elements involved in its definition. Then, model footprints are obtained by filtering models, selecting only elements that are instances of elements in the static metamodel footprint.

2.2.1. Analyzing Operations written in Kermeta. To establish the static metamodel footprint of an operation, we collect all metamodel elements (types and features) along its control flow graph — the set of all its possible execution paths. Image 3 displays an excerpt of Kermeta's abstract syntax. Among others, `CallFeature` is especially interesting, as it represents the access to an object's state or the invocation of an operation on an object. `CallFeature` objects can be found in 3 contexts within the abstract syntax tree (AST) of an operation:

- the body of a constraint (invariant of a class, contract of an operation)
- the body of an operation
- the body of a getter/setter of a derived property

For every reachable `CallFeature` object in the AST of the operation, we add the following elements to the static metamodel footprint:

- the (static) type of the target object
- the feature being invoked
- the type of the property or the parameters and return types of the operation

For every type being added to the static metamodel footprint, we include all its subtypes as well, to make subsequent model filtering easier.

The abstract syntax of Kermeta has been defined in Kermeta. Therefore, the analysis has been implemented in Kermeta too, as a visitor on Kermeta ASTs (i.e., *km* file). Two aspects are first woven to the Kermeta abstract syntax (or Kermeta metamodel):

- *EnhancedKermeta* augments Kermeta metamodel with usage annotations and some "shortcuts" to make the analysis easier.
- *VisitableKermeta* implements a visitor pattern for the Kermeta metamodel.

The creation of a static metamodel footprint (and a static footprint metamodel) is achieved in 5 phases:

1) *PrepareTransformation* prepares an operation written in Kermeta for analysis.
2) *AnalyzeTransformation* analyzes the operation by collecting metamodel elements while navigating through the control flow graph.
3) *CreateStaticFootprintMetamodel* creates a static footprint metamodel by removing irrelevant constructs from the input metamodel.
4) *DisplayStaticMetamodelFootprint* displays the static metamodel footprint and stores usage annotations in tags (so that they are available in Ecore models for the filtering phase).

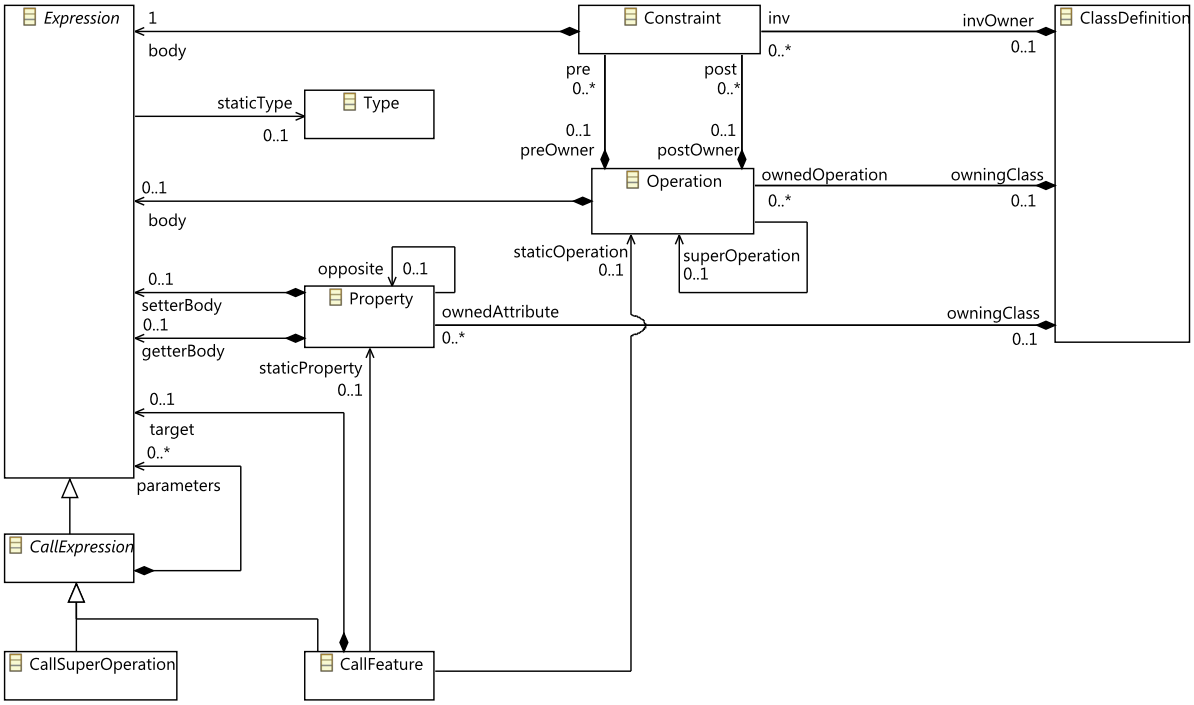

Figure 3. Excerpt of Kermeta's abstract syntax

5) *CompleteStaticMetamodelFootprint* completes a static metamodel footprint so that it becomes a valid metamodel footprint (by adding classes whose feature are part of the metamodel footprint).

Main loads a the AST of an operation written in Kermeta and applies these 5 steps to the given AST, creating a static footprint metamodel for it. The source code for analyzing Kermeta operations can be found in Appendix C.

2.2.2. Model Filtering. The filtering of a model through a static metamodel footprint is straightforward. First, we remove model objects whose metaclass is not part of the static metamodel footprint. Then, for every remaining object, we unset its features that are not part of the static metamodel footprint. The result of this filtering is a model conforming to the static footprint metamodel. We have implemented this filtering in Java: *MetamodelMatcher* establishes mappings between the original metamodel and the static metamodel footprint, to detect which classes and features should be filtered out. *ModelFilter* actually filters models according to these mappings. The source code for the model filtering can be found in Appendix C.

3. Experimental Material

To evaluate static footprinting, we compared the precision of static footprints (with respect to dynamic ones) and compared the cost of both approaches. In this section, we present the model and the operations we used for this evaluation. Results are presented and discussed in [1].

3.1. Models

For the evaluation, we decided to use metamodels (a specific kind of models) because there are many real-world metamodels available for researchers in the public domain. We have selected and prepared 75 models written in Ecore that were packaged in Eclipse plugins (for their implementation or provided as examples) or available in

online repositories such as AtlanMod[2]. Due to a limitation of our implementation, metamodels must be contained in a single EMF resource (a single Ecore file). Thus, for every metamodel having dependencies to other metamodels, we have modified it according to the following rules:

- if the metamodel uses some external data types, we added these data types into the metamodel's resource.
- if the metamodel extends or refers to some external classes, we added the complete external metamodel to the metamodel's resource.

Table 1 lists these models with their origin, their size (in terms of objects and settings, counted after modifications) and the modifications that where carried on them.

3.2. Model Operations

Typical usage on metamodels include code generation and documentation. Thus, we defined 5 operations for Ecore models in Kermeta:

- E2KV generates Kermeta code implementing a visitor pattern for the input metamodel.
- E2HTML creates an HTML document presenting the metamodel (as does Javadoc for Java code).
- E2SQL creates a SQL schema for storing models expressed in the input metamodel.
- E2DOT creates a DOT file to visualize the metamodel (DOT files can be rendered with GraphViz[3]).
- E2GEN creates an Genmodel from an Ecore model (as does EMF[4] tools to generate code from a metamodel).

The source code and the static metamodel footprints of these operations can be found in Appendix A.

4. Conclusion

In this report, we have presented the material required to produce the results presented in [1]. We discussed the implementation of both dynamic and static footprinting and described the models and operations we used for the evaluation.

References

[1] C. Jeanneret, M. Glinz, and B. Baudry, "Estimating footprints of model operations," in *33rd International Conference on Software Engineering (ICSE 2011)*, 2011.

[2] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel, "Weaving executability into object-oriented meta-languages," in *8th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2005)*, ser. LNCS, vol. 3713, 2005, pp. 264–278.

2. http://www.emn.fr/z-info/atlanmod/index.php/Zoos
3. http://www.graphviz.org
4. http://www.eclipse.org/emf

Table 1. List of Models

Model Name	Objects	Settings	Source	Modifications
ACME	85	1'227	AtlanMod	
Ant	333	4'675	AtlanMod	
ATL	466	6'826	AtlanMod	
AWKPrograms	57	874	AtlanMod	
BibTeX	106	1'565	AtlanMod	
BPEL	1'974	24'539	Implementation of BPEL Project	Includes WSDL, XSD and some datatypes from Ecore.
BPMN	1'158	9'265	Implementation of BPMN Modeler (STP)	Includes Ecore.
Bugzilla	202	2'108	AtlanMod	
C	89	1'312	AtlanMod	
Change	531	5'810	Implementation of EMF	Includes Ecore.
COBOL	105	1'370	AtlanMod	
CORBAComponent	75	1'142	AtlanMod	
CPP	87	1'214	AtlanMod	
CSharp	77	1'076	AtlanMod	
CWMCore	73	1'018	AtlanMod	
diff	711	7'910	Implementation of EMF Compare (EMFT)	Includes match and Ecore.
DocBook	23	353	AtlanMod	
DOT	202	2'852	AtlanMod	
DTD	87	1'158	AtlanMod	
Ecore	441	4'584	Implementation of EMF	
extlibrary	113	1'419	Example of EMF	Includes some datatypes from Ecore.
FeatureDiagrams	62	880	AtlanMod	
filesystem	44	479	Example of Epsilon (GMT)	Includes some datatypes from Ecore.
flowchart	45	481	Example of Epsilon (GMT)	Includes some datatypes from Ecore.
friends	33	249	Example of Epsilon (GMT)	Includes some datatypes from Ecore.
fsmStatic	59	535	Example of Kermeta	
Gantt	97	1'385	AtlanMod	
GenModel	792	9'126	Implementation of EMF	Includes Ecore.
gmfgraph	545	6'556	Implementation of GMF	Includes some datatypes from Ecore.
gmfmap	1'655	17'607	Implementation of GMF	Includes gmfgraph, tooldef and Ecore.
GraphML	89	1'162	AtlanMod	
GraphVizDot	65	852	AtlanMod	
HTML	330	4'679	AtlanMod	
HTTP	1'342	16'065	Implementation of WSDL Editor (WTP)	
ImperativeOCL	2'272	20'308	Implementation of Operational QVT (M2M)	Includes Ecore, OCL and OCLEcore.
instance	794	7'600	Implementation of JEM (VEP)	Includes Ecore.
java	777	7'802	Implementation of JEM (VEP)	Includes Ecore.
kermeta_java	780	7'140	Implementation of Kermeta	
KM3	64	905	AtlanMod	
kompose	87	1'237	Implementation of Kompose (Kermeta)	
LaTeX	138	2'112	AtlanMod	
logoCSM	112	1'598	Example of Kermeta	Includes some datatypes from Ecore.
Make	43	636	AtlanMod	
match	490	5'314	Implementation of EMF Compare (EMFT)	Includes Ecore.
MavenMaven	368	5'187	AtlanMod	
MavenProject	80	1'113	AtlanMod	
MIME	1'363	16'294	Implementation of WSDL Editor (WTP)	Includes WSDL, XSD and some datatypes from Ecore.
mindmap	83	1'059	Example of GMF	Includes some datatypes from Ecore.
SpreadsheetMLBasicDef	233	3'334	AtlanMod	
WordprocessingMLBasicDef	300	4'306	AtlanMod	
DatadiagramMLBasicDef	436	6'358	AtlanMod	
mtl	2'184	19'704	Implementation of Acceleo (M2T)	Includes Ecore, OCL and OCLEcore.
MySQL	47	663	AtlanMod	
notation	939	10'699	Implementation of GMF	Includes Ecore.
OCL	1'361	10'709	Implementation of OCL (MDT)	Includes some datatypes from Ecore.
OCLCST	342	4'342	Implementation of OCL (MDT)	Includes some datatypes from Ecore.
OCLEcore	2'045	17'724	Implementation of OCL (MDT)	Includes OCL and Ecore.
OCLUML	13'849	104'691	Implementation of OCL (MDT)	Includes UML, Ecore and OCL.
PluginEclipse	111	1'521	AtlanMod	
QVT	797	11'447	AtlanMod	
QVTOperationalCST	793	10'820	Implementation of Operational QVT (M2M)	Includes OCLCST and some datatypes from Ecore.
RequisitePro	53	737	AtlanMod	
RSS-2.0	154	2'110	AtlanMod	
SOAP	1'411	16'928	Implementation of WSDL Editor (WTP)	Includes WSDL, XSD and some datatypes from Ecore.
STS	537	5'992	Implementation of Sintaks (Kermeta)	Includes Ecore.
SVG	217	3'070	AtlanMod	
taipan	82	1'122	Example of GMF	Includes some datatypes from Ecore.
tooldef	148	1'799	Implementation of GMF	Includes some datatypes from Ecore.
UML	12'269	91'675	Implementation of UML2 (MDT)	Includes Ecore.
WSDL	1'315	15'679	Implementation of WSDL Editor (WTP)	Includes XSD and some Ecore datatypes.
XHTML	1'035	13'947	AtlanMod	
XML	30	403	AtlanMod	
XQuery	100	1'503	AtlanMod	
XSD	668	8'752	Implementation of EMF	
XSLT	65	925	AtlanMod	

The source code of E2* operations are given in Appendix A. The implementation of dynamic footprinting is given in Appendix B while Appendix C lists the source code for static footprinting. The source code presented in these appendices can be found online at http://www.ifi.uzh.ch/rerg/research/miria/footprints.

Appendix A.
E2* Operations

```
@mainClass "ch::uzh::ifi::rerg::e2kv::E2KV"
@mainOperation "main"


package ch::uzh::ifi::rerg::e2kv;

require kermeta

require "Ecore.kmt"

using kermeta::utils
using kermeta::standard
using kermeta::language::structure
using kermeta::language::behavior
using kermeta::persistence
using ecore

class E2KV
{
  operation main(inputURI: String, outputURI: String): Void is do
    var pkgs: OrderedSet<EPackage> init loadModel(inputURI)
    var visitableMU: ModelingUnit init createVisitableMetamodel(pkgs)
    saveKermetaModel(visitableMU, outputURI)
  end

  operation loadModel(mmURI: String): OrderedSet<EPackage> is do
    var rep : EMFRepository init EMFRepository.new
    var res : EMFResource
    res ?= rep.getResource(mmURI)
    result := OrderedSet<EPackage>.new
    res.select{e | e.isKindOf(EPackage)}.each{p |
        result.add(p.asType(EPackage))}
  end

  operation saveKermetaModel(mUnit: ModelingUnit, uri: String): Void is do
    var rep : EMFRepository init EMFRepository.new
    var res : EMFResource
    res ?= rep.createResource(uri,
        "http://www.kermeta.org/kermeta/1_2_0//kermeta")
    res.add(mUnit)
    res.save()
  end

  operation createVisitableMetamodel(pkgs: OrderedSet<EPackage>):
      ModelingUnit is do
```

```
    result := ModelingUnit.new
    pkgs.each{p |
      var vp: VisitorPattern init VisitorPattern.new
      result.packages.add(vp.createVisitablePackage(p)) }

    var kermetaFrameworkRequire: Require init Require.new
    result.requires.add(kermetaFrameworkRequire)
    kermetaFrameworkRequire.uri := "kermeta"
  end
}

class VisitorPattern
{

  reference visitorClass: ClassDefinition
  reference visitableClass: ClassDefinition
  reference defaultAcceptOp: Operation
  reference defaultVisitOp: Operation

  reference acceptOps: Hashtable<EClass, Operation>
  reference visitOps: Hashtable<EClass, Operation>
  reference packages: Hashtable<EPackage, Package>
  reference classDefs: Hashtable<EClass, ClassDefinition>

  reference packageOrdering: Sequence<EPackage>
  reference classOrdering: Sequence<EClass>


  operation createVisitablePackage(root: EPackage): Package is do

    var mmName: String init root.name
    packages := Hashtable<EPackage, Package>.new
    acceptOps := Hashtable<EClass, Operation>.new
    visitOps := Hashtable<EClass, Operation>.new
    classDefs := Hashtable<EClass, ClassDefinition>.new
    packageOrdering := Sequence<EPackage>.new
    classOrdering := Sequence<EClass>.new

    // Phase 1 - Build Aspect Unit skeleton


    visitorClass := ClassDefinition.new
    visitorClass.name := mmName + "Visitor"
    visitorClass.isAbstract := true

    createPackage(root)
    // Phase 2 - Set it up

    result := packages.getValue(root)
    result.ownedTypeDefinition.add(visitorClass)


    visitableClass := ClassDefinition.new
```

```
    visitableClass.name := mmName + "VisitableElement"
    visitableClass.isAbstract := true

    defaultVisitOp := Operation.new
    defaultAcceptOp := Operation.new

    result.ownedTypeDefinition.add(visitableClass)


    setUpVisitOp(visitableClass, defaultVisitOp)
    setUpAcceptOp(visitableClass, defaultVisitOp, defaultAcceptOp)

    // Watch out for order
    packageOrdering.each{p | setUpPackage(p)}
    classOrdering.each{cl |
      setUpClassDef(cl)
      var vOp: Operation init visitOps.getValue(cl)
      if not vOp.isVoid
      then
        var vAs: ClassDefinition init classDefs.getValue(cl)

        setUpVisitOp(vAs, vOp)

        var aOp: Operation init acceptOps.getValue(cl)
        if not aOp.isVoid
        then
          setUpAcceptOp(vAs, vOp, aOp)
        end
      end
      }

    // Phase 3 - Finishing



  end

  /*
   * First Phase: Create Aspect Unit Skeleton
   */

  operation createPackage(p: EPackage): Void is do
    var pack: Package init Package.new
    packages.put(p,pack)
    packageOrdering.add(p)
    p.eSubpackages.each{sp | createPackage(sp)}
    p.eClassifiers.select{c | c.isInstanceOf(EClass)}.each{c |
        createAspect(c.asType(EClass))}
  end


  operation createAspect(cl: EClass): Void is do
    classDefs.put(cl,ClassDefinition.new)
```

```
    classOrdering.add(cl)
    if not(cl.~abstract)
    then
      acceptOps.put(cl, Operation.new)
      visitOps.put(cl, Operation.new)
    end
  end

  /*
   * Second Phase: Setup Aspect Unit
   */

  operation setUpPackage(p: EPackage): Void is do
    var pack: Package init packages.getValue(p)
    pack.name := p.name

    if not p.eSuperPackage.isVoid
    then
      var parent: Package init packages.getValue(p.eSuperPackage)
      parent.nestedPackage.add(pack)
    end

  end

  operation setUpClassDef(cl: EClass): Void is do
    var classDef: ClassDefinition init classDefs.getValue(cl)
    classDef.isAspect := true
    classDef.name := cl.name
    packages.getValue(cl.ePackage).ownedTypeDefinition.add(classDef)
    classDef.isAbstract := cl.~abstract

    cl.eSuperTypes.each{ecl |
      var sup: ClassDefinition init classDefs.getValue(ecl)
      if not sup.isVoid
      then
        addSuperClass(classDef, sup)
      end }

    addSuperClass(classDef,visitableClass)


  end

  operation setUpVisitOp(classDef: ClassDefinition, visitOp: Operation): Void
      is do
    var targetParam: Parameter init Parameter.new
    targetParam.name := "arg"
    setType(targetParam, createClassType(classDef))
    setMulti(targetParam)
    visitOp.ownedParameter.add(targetParam)
    visitOp.name := "visit"+classDef.name
    setType(visitOp, VoidType.new)
    setMulti(visitOp)
```

```
    visitOp.body := Block.new
    setType(visitOp.body, visitOp.type)
    visitorClass.ownedOperation.add(visitOp)

    if classDef.superType.isEmpty
    then
      //Default Visit Op

      //TODO: Add some behavior here... A warning? An exception? Make it abstract?
    else
      // Call default visit op

      var cv: CallVariable init CallVariable.new
      cv.name := targetParam.name
      setType(cv, targetParam.type)

      var selfExp: SelfExpression init SelfExpression.new
      setType(selfExp,createClassType(classDef))

      var cf: CallFeature init CallFeature.new
      cf.name := defaultVisitOp.name
      cf.target := selfExp
      cf.parameters.add(cv)
      cf.staticOperation := defaultVisitOp

      setType(cf,createFunctionType(createClassType(visitableClass), VoidType.new))

      visitOp.body.asType(Block).statement.add(cf)
    end
  end

  operation setUpAcceptOp(classDef: ClassDefinition, visitOp: Operation, acceptOp: Operation): Void is do
    acceptOp.name := "accept"
    setMulti(acceptOp)
    setType(acceptOp, VoidType.new)

    var visitorParam: Parameter init Parameter.new
    visitorParam.name := "v"
    setType(visitorParam, createClassType(visitorClass))
    setMulti(visitorParam)
    acceptOp.ownedParameter.add(visitorParam)

    if classDef.superType.isEmpty
    then
      //Default Accept Op
      tagObject(acceptOp, "Overloadable", "true")
    else
      acceptOp.superOperation := defaultAcceptOp
    end

```

```
    // Note: a lot of type objects created here...
    // Should we optimize it a bit?

    acceptOp.body := Block.new
    setType(acceptOp.body, acceptOp.type)

    var cv: CallVariable init CallVariable.new
    cv.name := visitorParam.name
    setType(cv, visitorParam.type)

    var selfExp: SelfExpression init SelfExpression.new
    setType(selfExp,createClassType(classDef))

    var cf: CallFeature init CallFeature.new
    cf.name := visitOp.name
    cf.target := cv
    cf.parameters.add(selfExp)
    cf.staticOperation := visitOp

    setType(cf,createFunctionType(createClassType(classDef), VoidType.new))

    acceptOp.body.asType(Block).statement.add(cf)

    classDef.ownedOperation.add(acceptOp)
  end

  /**
   * Utility Methods
   */

  operation tagObject(obj: Object, name: String, val: String): Void is do
    var tag: Tag init Tag.new
    tag.name := name
    tag.~value := val
    obj.tag.add(tag)
    obj.ownedTags.add(tag)
  end

  operation setMulti(me: MultiplicityElement): Void is do
    me.lower := 0
    me.upper := 1
    me.isUnique := true
    me.isOrdered := true
  end

  operation setType(tc: TypeContainer, type: Type): Void is do

    if type.typeContainer.isVoid
    then
      tc.containedType.add(type)
    end

    var te: TypedElement
```

```
    te ?= tc
    if not te.isVoid
    then
      te.type := type
    end

    var exp: Expression
    exp ?= tc
    if not exp.isVoid
    then
      exp.staticType := type
    end
  end

  operation createClassType(cd: ClassDefinition): Class is do
    result := Class.new
    result.typeDefinition := cd
  end

  operation createFunctionType(argT: Type, resT: Type): FunctionType is do
    result := FunctionType.new

    var prodT: ProductType init ProductType.new

    if argT.typeContainer.isVoid
    then
      prodT.containedType.add(argT)
    end

    prodT.type.add(argT)

    result.left := prodT
    result.right := resT

    if resT.typeContainer.isVoid
    then
      result.containedType.add(resT)
    end
    result.containedType.add(prodT)
  end

  operation addSuperClass(sub: ClassDefinition, sup: ClassDefinition): Void
      is do
    var superType: Type init createClassType(sup)
    sub.superType.add(superType)
    sub.containedType.add(superType)
  end
}
```

Listing 1. ModelOperations/E2KV.kmt

```
@mainClass "ch::uzh::ifi::rerg::e2html::E2HTML"
@mainOperation "main"
```

```
3
4 package ch::uzh::ifi::rerg::e2html;
5
6 require kermeta
7
8 require "Ecore.kmt"
9 require "HTML.kmt"
10
11 using kermeta::standard
12 using kermeta::utils
13 using kermeta::persistence
14 using kermeta::io
15 using ecore
16 using html
17
18
19 class E2HTML
20 {
21   operation main(inputURI: String, outputURI: String): Void is do
22     var pkgs: OrderedSet<EPackage> init loadModel(inputURI)
23     var doc: HTML init createDoc(pkgs)
24     saveDocText(doc, outputURI)
25   end
26
27   attribute modelURI: String
28
29   operation loadModel(mmURI: String): OrderedSet<EPackage> is do
30     var rep : EMFRepository init EMFRepository.new
31     var res : EMFResource
32     res ?= rep.getResource(mmURI)
33     result := OrderedSet<EPackage>.new
34     res.select{e | e.isKindOf(EPackage)}.each{p |
         result.add(p.asType(EPackage))}
35     modelURI := mmURI
36   end
37
38   operation saveDocModel(doc: HTML, uri: String): Void is do
39     var rep : EMFRepository init EMFRepository.new
40     var res : EMFResource
41     res ?= rep.createResource(uri, "http://www.ifi.uzh.ch/rerg/html")
42     res.add(doc)
43     res.save()
44   end
45
46   operation saveDocText(doc: HTML, uri: String): Void is do
47     var file : FileIO init FileIO.new
48     file.writeTextFile(uri,doc.toHTML())
49   end
50
51   reference ctx: HTMLElement
52
53   operation createDoc(pkgs: OrderedSet<EPackage>): HTML is do
54     result := HTML.new
```

```
    result.title := "Metamodel Documentation"
    ctx := result
    pkgs.each{p | visitPackage(p)}
  end

  operation visitPackage(pkg: EPackage): Void is do
    ctx.createH1(getPackageID(pkg)).createText("EPackage: " + pkg.name)
    insertDoc(pkg,ctx)
    pkg.eClassifiers.select{c | c.isInstanceOf(EClass)}.each{c |
        visitClass(c.asType(EClass))}
    pkg.eClassifiers.select{c | c.isInstanceOf(EEnum)}.each{c |
        visitEnum(c.asType(EEnum))}
    pkg.eClassifiers.select{c | c.isInstanceOf(EDataType) and (not
        (c.isInstanceOf(EEnum)))}.each{c |
        visitPrimitiveType(c.asType(EDataType))}
    pkg.eSubpackages.each{p | visitPackage(p)}
  end

  operation visitClass(cl: EClass): Void is do
    var h2: H2 init ctx.createH2(getClassifierID(cl))
    h2.createText("EClass: ")

    if cl.~abstract
    then
      h2.createText(cl.name).makeEM()
    else
      h2.createText(cl.name)
    end

    insertDoc(cl, ctx)

    if not cl.eSuperTypes.isEmpty
    then
      ctx.createH3().createText("Inherits from")
      var ul: UL init ctx.createUL()
      cl.eSuperTypes.each{scl |
          ul.createLI().createText(scl.name).makeA("#"+getClassifierID(scl))}
    end

    if not cl.eStructuralFeatures.isEmpty
    then
      ctx.createH3().createText("Structural Features")
      var oldCtx: HTMLElement init ctx
      ctx := ctx.createTABLE()
      cl.eStructuralFeatures.each{feat | visitTypedElement(feat)}
      ctx := oldCtx
    end

    if not cl.eOperations.isEmpty
    then
      ctx.createH3().createText("Operations")
      var oldCtx: HTMLElement init ctx
      ctx := ctx.createTABLE()
```

```
      cl.eOperations.each{op | visitOperation(op)}
      ctx := oldCtx
    end
  end

  operation visitTypedElement(te: ETypedElement): Void is do
    var oldCtx: HTMLElement init ctx
    ctx := ctx.createTR()
    var typeTD: TD init ctx.createTD()

    if isTypeInDocument(te.eType)
    then
      typeTD.createText(te.eType.name).makeA("#" + getClassifierID(te.eType))
    else
      typeTD.createText("?")
    end
    ctx.createTD().createText(getMultiplicity(te))
    ctx.createTD().createText(te.name)
    insertDoc(te,ctx.createTD())
    ctx := oldCtx
  end

  operation visitOperation(op: EOperation): Void is do
    var oldCtx: HTMLElement init ctx
    ctx := ctx.createTR()

    var typeTD: TD init ctx.createTD()

    if isTypeInDocument(op.eType)
    then
      typeTD.createText(op.eType.name).makeA("#" + getClassifierID(op.eType))
    else
      typeTD.createText("?")
    end


    var nameTD: TD init ctx.createTD()
    nameTD.createText(op.name)
    insertDoc(op,nameTD)


    var paramTD: TD init ctx.createTD()
    if not op.eParameters.isEmpty
    then
      ctx := paramTD.createTABLE()
      op.eParameters.each{p | visitTypedElement(p)}
    else
      paramTD.createText("-")
    end



    ctx := oldCtx
```

```
  end

  operation isTypeInDocument(type: EClassifier): Boolean is do
    result := false
    if not type.isVoid
    then
      if not type.containingResource.isVoid
      then
        if type.containingResource.uri == modelURI
        then
          result := true
        end
      end
    end
  end

  operation visitEnum(en: EEnum): Void is do
    ctx.createH2(getClassifierID(en)).createText("EEnum: " + en.name)
    insertDoc(en,ctx)
    var oldCtx: HTMLElement init ctx
    ctx := ctx.createUL()
    en.eLiterals.each{l | visitEnumLiteral(l)}
    ctx := oldCtx
  end

  operation visitEnumLiteral(el: EEnumLiteral): Void is do
    var li: LI init ctx.createLI()
    li.createText(el.~value.toString + ": ")
    li.createText(el.name).makeEM()
    insertDoc(el,li)
  end

  operation visitPrimitiveType(pt: EDataType): Void is do
    ctx.createH2(getClassifierID(pt)).createText("EDataType: " + pt.name)
    insertDoc(pt,ctx)
    if not pt.instanceTypeName.isVoid
    then
      var p: P init ctx.createP()
      p.createText("Mapped to: ")
      p.createText(pt.instanceTypeName).makeEM()
    end
  end


  operation getClassifierID(cl: EClassifier): String is do
    result := getPackageID(cl.ePackage) + "." + cl.name
  end

  operation getPackageID(pkg: EPackage): String is do
    if pkg.eSuperPackage.isVoid
    then
      result := pkg.name
    else
```

```
      result := getPackageID(pkg.eSuperPackage) + "." + pkg.name
    end
  end

  operation getMultiplicity(feat: ETypedElement): String is do
    var sb: StringBuffer init StringBuffer.new
    sb.append("[")
    sb.append(feat.lowerBound.toString)
    sb.append(",")
    if feat.upperBound == -1
    then
      sb.append("*")
    else
      sb.append(feat.upperBound.toString)
    end
    sb.append("]")
    result := sb.toString
  end

  operation insertDoc(elem: EModelElement, par: HTMLElement): Void is do

    var an: EAnnotation init
        elem.getEAnnotation("http://www.eclipse.org/emf/2002/GenModel")
    if not an.isVoid
    then
      var det: EStringToStringMapEntry init an.details.detect { a |
          "documentation".equals(a.key)}
      if not det.isVoid
      then
        par.createPRE().createText(det.~value)
      end
    end
  end
}
```

Listing 2. ModelOperations/E2HTML.kmt

```
@mainClass "ch::uzh::ifi::rerg::e2sql::E2SQL"
@mainOperation "main"

package ch::uzh::ifi::rerg::e2sql;

require kermeta

require "Ecore.kmt"
require "DBSchema.kmt"

using kermeta::standard
using kermeta::utils
using kermeta::persistence
using kermeta::io
using ecore
using dbschema
```

```
class E2SQL
{
  operation main(inputURI: String, outputURI: String): Void is do
    var pkgs: OrderedSet<EPackage> init loadModel(inputURI)
    var schema: Schema init createSchema(pkgs)
    saveSchemaText(schema, outputURI)
  end

  operation loadModel(mmURI: String): OrderedSet<EPackage> is do
    var rep : EMFRepository init EMFRepository.new
    var res : EMFResource
    res ?= rep.getResource(mmURI)
    result := OrderedSet<EPackage>.new
    res.select{e | e.isKindOf(EPackage)}.each{p |
        result.add(p.asType(EPackage))}
  end

  operation saveSchemaModel(schema: Schema, uri: String): Void is do
    var rep : EMFRepository init EMFRepository.new
    var res : EMFResource
    res ?= rep.createResource(uri, "http://www.ifi.uzh.ch/rerg/dbschema")
    res.add(schema)
    res.save()
  end

  operation saveSchemaText(schema: Schema, uri: String): Void is do
    var file : FileIO init FileIO.new
    file.writeTextFile(uri,schema.toSQL())
  end


  reference tables: Hashtable<EClassifier, Table>
  reference connectionTables: Hashtable<EStructuralFeature, Table>
  reference schema: Schema

  reference attributes: Bag<EAttribute>
  reference references: Bag<EReference>

  operation createSchema(pkgs: OrderedSet<EPackage>): Schema is do
    schema := Schema.new
    result := schema
    schema.name := "Schema" // TODO: Something Better here?

    tables := Hashtable<EClassifier, Table>.new
    connectionTables := Hashtable<EStructuralFeature,Table>.new
    attributes := Bag<EAttribute>.new
    references := Bag<EReference>.new


    // First, create a table per class / enumeration
    pkgs.each{p | createTablesForClassifiers(p)}

```

```
    // Then, proceed with attributes / references
    attributes.each{att | proceedWithAttribute(att)}
    references.each{ref | proceedWithReference(ref)}
  end

  operation createTablesForClassifiers(pkg: EPackage): Void is do
    pkg.eSubpackages.each{sp | createTablesForClassifiers(sp)}
    pkg.eClassifiers.select{c | c.isInstanceOf(EClass)}.each{c |
        createTableForClass(c.asType(EClass))}
    pkg.eClassifiers.select{c | c.isInstanceOf(EEnum)}.each{c |
        createTableForEnum(c.asType(EEnum))}
  end

  operation createTableForEnum(en: EEnum): Table is do
    result := tables.getValue(en)
    if result.isVoid
    then
      result := schema.createTable(getClassifierID(en))
      tables.put(en,result)
      result.addColumnToPrimaryKey(result.createColumn("value", "Integer",
          true))
      result.createColumn("label", "String", true)
    end
  end

  operation createTableForClass(cl: EClass): Table is do
    result := tables.getValue(cl)
    if result.isVoid
    then
      result := schema.createTable(getClassifierID(cl))
      tables.put(cl,result)
      if cl.eSuperTypes.isEmpty
      then
        result.createArtificialPrimaryKey()
      else
        cl.eSuperTypes.each{scl |
          var supTable: Table init createTableForClass(scl)
          result.createReferenceToTable(supTable,scl.name,
              true).columns.each{c | result.addColumnToPrimaryKey(c)}}
      end
      attributes.addAll(cl.eAttributes.select{att| not att.transient})
      references.addAll(cl.eReferences.select{ref| not ref.transient})
    end
  end

  operation createConnexionTableForFeature(feature: EStructuralFeature):
      Table is do
    result := schema.createTable(feature.name)
  end

  operation proceedWithAttribute(att: EAttribute): Void is do
    var table: Table init tables.getValue(att.eContainingClass)
    if not att.many
```

```
    then
      if att.eAttributeType.isInstanceOf(EEnum)
      then
        var en: EEnum init att.eAttributeType.asType(EEnum)
        table.createReferenceToTable(createTableForEnum(en), att.name,
            att.required)
      else
        table.createColumn(att.name,att.eAttributeType.name,att.required)
      end
    else
      var conTable: Table init schema.createTable(att.eContainingClass.name +
          "_" + att.name)
      connectionTables.put(att,conTable)
      conTable.createArtificialPrimaryKey()
      conTable.createReferenceToTable(table, att.eContainingClass.name, true)

      if att.eAttributeType.isInstanceOf(EEnum)
      then
        var en: EEnum init att.eAttributeType.asType(EEnum)
        conTable.createReferenceToTable(createTableForEnum(en), att.name,
            true)
      else
        conTable.createColumn(att.name,att.eAttributeType.name,true)
      end
    end
  end

  operation proceedWithReference(ref: EReference): Void is do
    var opRef: EReference init ref.eOpposite
    var selfTable: Table init tables.getValue(ref.eContainingClass)
    var opTable: Table init tables.getValue(ref.eReferenceType)

    if not opTable.isVoid
    then
      if not ref.many
      then
        selfTable.createReferenceToTable(opTable, ref.name, ref.required)
      else
        if opRef.isVoid
        then // create connection table
          var conTable: Table init
              schema.createTable(ref.eContainingClass.name + "_" + ref.name)
          connectionTables.put(ref,conTable)
          conTable.createArtificialPrimaryKey()
          conTable.createReferenceToTable(selfTable,
              ref.eContainingClass.name, true)
          conTable.createReferenceToTable(opTable, ref.name, true)
        else
          if opRef.many
          then //M:N relationship
            var conTable: Table init connectionTables.getValue(opRef)
            if conTable.isVoid
            then
```

```
                var refName: String init ref.eContainingClass.name + "_" +
                    ref.name
                var opRefName: String init opRef.eContainingClass.name + "_" +
                    opRef.name
                conTable := schema.createTable(refName + "_" + opRefName )
                connectionTables.put(ref,conTable)

                conTable.createArtificialPrimaryKey()
                conTable.createReferenceToTable(selfTable, opRef.name, true)
                conTable.createReferenceToTable(opTable, ref.name, true)
              else
                connectionTables.put(ref,conTable)
              end


            else //Nothing to do - the eOpposite's table has a pointer to this
                table
            end
          end
        end
      end
    end

    operation getClassifierID(cl: EClassifier): String is do
      result := getPackageID(cl.ePackage) + "_" + cl.name
    end

    operation getPackageID(pkg: EPackage): String is do
      if pkg.eSuperPackage.isVoid
      then
        result := pkg.name
      else
        result := getPackageID(pkg.eSuperPackage) + "_" + pkg.name
      end
    end
}
```

Listing 3. ModelOperations/E2SQL.kmt

```
@mainClass "ch::uzh::ifi::rerg::e2dot::E2DOT"
@mainOperation "main"

package ch::uzh::ifi::rerg::e2dot;

require kermeta

require "Ecore.kmt"
require "DOT.kmt"

using kermeta::standard
using kermeta::utils
using kermeta::persistence
using kermeta::io
```

```
using ecore
using dot

class E2DOT
{
  operation main(inputURI: String, outputURI: String): Void is do
    var pkgs: OrderedSet<EPackage> init loadModel(inputURI)
    var graph: Graph init createGraph(pkgs)
    saveGraphText(graph, outputURI)
  end

  operation loadModel(mmURI: String): OrderedSet<EPackage> is do
    var rep : EMFRepository init EMFRepository.new
    var res : EMFResource
    res ?= rep.getResource(mmURI)
    result := OrderedSet<EPackage>.new
    res.select{e | e.isKindOf(EPackage)}.each{p |
        result.add(p.asType(EPackage))}
    modelURI := mmURI
  end

  operation saveGraphModel(graph: Graph, uri: String): Void is do
    var rep : EMFRepository init EMFRepository.new
    var res : EMFResource
    res ?= rep.createResource(uri, "http://www.ifi.uzh.ch/rerg/dot")
    res.add(graph)
    res.save()
  end

  operation saveGraphText(graph: Graph, uri: String): Void is do
    var file : FileIO init FileIO.new
    file.writeTextFile(uri,graph.toDOT())
  end

  reference nodes: Hashtable<EClassifier, Node>
  reference edges: Hashtable<EStructuralFeature, Arc>
  reference root: Graph
  reference subClasses: OrderedSet<EClass> // Only used to preserve ordering
      in DOT file
  reference references: OrderedSet<EReference>
  reference modelURI: String

  operation createGraph(pkgs: OrderedSet<EPackage>): Graph is do
    root := Graph.new
    result := root
    root.rankDir := RankDir.TB
    root.id := "Metamodel" // TODO: Something Better here?

    nodes := Hashtable<EClassifier, Node>.new
    edges := Hashtable<EStructuralFeature, Arc>.new
    references := OrderedSet<EReference>.new
    subClasses := OrderedSet<EClass>.new

```

```
    // First, create a node per class / enumeration
    pkgs.each{p | createNodesForClassifiers(p)}

    // Then, proceed with references
    references.each{ref | proceedWithReference(ref)}

    // Finally, proceed with inheritance
    subClasses.each{cl | proceedWithInheritance(cl)}
  end

  operation createNodesForClassifiers(pkg: EPackage): Void is do
    var oldCtx : Graph init root
    var newCtx : SubGraph init SubGraph.new
    newCtx.id := getPackageID(pkg)
    newCtx.label := SimpleLabel.new
    newCtx.label.content := pkg.name
    newCtx.cluster := true
    oldCtx.elements.add(newCtx)

    root := newCtx
    pkg.eSubpackages.each{sp | createNodesForClassifiers(sp)}
    pkg.eClassifiers.each{c | createNodeForClassifier(c)}

    root := oldCtx
  end

  operation createNodeForClassifier(cl: EClassifier): Void is do
    var node: Node init Node.new
    node.id := getClassifierID(cl)
    node.shape := PlaintextShape.new

    var en: EEnum
    en ?= cl

    var cls: EClass
    cls ?= cl

    var label: StringBuffer init StringBuffer.new

    label.append("<table border=\"0\" cellborder=\"1\" cellspacing=\"0\"
        cellpadding=\"5\">")
    label.append("<tr>")
    label.append("<td balign=\"CENTER\" align=\"CENTER\">")

    if (not cls.isVoid)
    then
      if cls.~abstract
      then
        label.append("<font face=\"Times-Italic\">")
      end
    end

    label.append(cl.name)
```

```
118
119     if (not cls.isVoid)
120     then
121       if cls.~abstract
122       then
123         label.append("</font>")
124       end
125       if not cls.eSuperTypes.isEmpty
126       then
127         subClasses.add(cls)
128       end
129     end
130
131     label.append("</td>")
132     label.append("</tr>")
133
134     if not en.isVoid
135     then
136       label.append("<tr>")
137       label.append("<td balign=\"LEFT\" align=\"LEFT\">")
138       en.eLiterals.each{el | label.append(el.name) label.append("<br/>")}
139       label.append("</td>")
140       label.append("</tr>")
141     end
142
143     if not cls.isVoid
144     then
145       references.addAll(cls.eReferences)
146       // attributes
147       label.append("<tr>")
148       label.append("<td balign=\"LEFT\" align=\"LEFT\">")
149       cls.eAttributes.each{att | label.append(printAttribute(att))
             label.append("<br/>")}
150       label.append("</td>")
151       label.append("</tr>")
152       // operations
153       label.append("<tr>")
154       label.append("<td balign=\"LEFT\" align=\"LEFT\">")
155       cls.eOperations.each{op | label.append(printOperation(op))
             label.append("<br/>")}
156       label.append("</td>")
157       label.append("</tr>")
158     end
159
160     label.append("</table>")
161
162     node.label := HTMLLabel.new
163     node.label.content := label.toString
164     root.elements.add(node)
165     nodes.put(cl, node)
166   end
167
168   operation proceedWithReference(ref: EReference): Void is do
```

```
169     var opRef: EReference init ref.eOpposite
170     var selfNode: Node init nodes.getValue(ref.eContainingClass)
171     var opNode: Node init nodes.getValue(ref.eReferenceType)
172
173     if isTypeInGraph(ref.eReferenceType)
174     then
175
176       if opRef.isVoid
177       then
178         var edge: Arc init Arc.new
179         edge.toNode := opNode
180         edge.fromNode := selfNode
181         root.elements.add(edge)
182         edge.headLabel := SimpleLabel.new
183         edge.headLabel.content := printReference(ref)
184         edge.arrowHead := createReferenceArrow()
185
186         if ref.containment
187         then
188           edge.arrowTail := createDiamondArrow()
189         else
190           edge.arrowTail := createNoArrow()
191         end
192       else
193         var edge: Arc init edges.getValue(opRef)
194         if edge.isVoid
195         then
196           edge := Arc.new
197           edge.toNode := opNode
198           edge.fromNode := selfNode
199           root.elements.add(edge)
200
201           edge.headLabel := SimpleLabel.new
202           edge.headLabel.content := printReference(ref)
203
204           edge.tailLabel := SimpleLabel.new
205           edge.tailLabel.content := printReference(opRef)
206
207           if opRef.containment
208           then
209             edge.arrowHead := createDiamondArrow()
210           else
211             edge.arrowHead := createNoArrow()
212           end
213
214           if ref.containment
215           then
216             edge.arrowTail := createDiamondArrow()
217           else
218             edge.arrowTail := createNoArrow()
219           end
220
221           edges.put(opRef, edge)
```

```
        end
        edges.put(ref,edge)
      end
    end
  end

  operation proceedWithInheritance(cl: EClass): Void is do
    var selfNode: Node init nodes.getValue(cl)
    cl.eSuperTypes.each{scl |
      var superNode: Node init nodes.getValue(scl)
      var edge: Arc init Arc.new
      edge.toNode := superNode
      edge.fromNode := selfNode
      edge.arrowHead := createInheritanceArrow()
      edge.arrowTail := createNoArrow()
      root.elements.add(edge)}
  end

  operation getClassifierID(cl: EClassifier): String is do
    result := getPackageID(cl.ePackage) + "_" + cl.name
  end

  operation getPackageID(pkg: EPackage): String is do
    if pkg.eSuperPackage.isVoid
    then
      result := pkg.name
    else
      result := getPackageID(pkg.eSuperPackage) + "_" + pkg.name
    end
  end

  operation printReference(ref: EReference): String is do
    result := ref.name + "\\n" + printMultiplicity(ref)
  end

  operation printMultiplicity(te: ETypedElement): String is do
    var sb: StringBuffer init StringBuffer.new
    if(te.lowerBound == te.upperBound)
    then
      sb.append(te.lowerBound.toString)
    else
      sb.append(te.lowerBound.toString)
      sb.append("..")
      if(te.upperBound == -1)
      then sb.append("*")
      else sb.append(te.upperBound.toString)
      end
    end
    result := sb.toString
  end

  operation printAttribute(att: EAttribute): String is do
    var sb: StringBuffer init StringBuffer.new
```

```
    sb.append(att.name)
    sb.append("[")
    sb.append(printMultiplicity(att))
    sb.append("]")
    if isTypeInGraph(att.eType)
    then
      sb.append(": ")
      sb.append(att.eType.name)
    end
    result := sb.toString
  end

  operation printOperation(op: EOperation): String is do
    var sb: StringBuffer init StringBuffer.new
    sb.append(op.name)
    sb.append("(")
    from var i: Integer init 0
    until i == op.eParameters.size()
    loop
      sb.append(printParam(op.eParameters.elementAt(i)))
      if i != op.eParameters.size()
      then
        sb.append(",")
      end
      i := i+1
    end
    sb.append(")")
    if isTypeInGraph(op.eType)
    then
      sb.append(":")
      sb.append(op.eType.name)
      sb.append("[")
      sb.append(printMultiplicity(op))
      sb.append("]")
    end

    result := sb.toString
  end

  operation printParam(param: EParameter): String is do
    var sb: StringBuffer init StringBuffer.new
    sb.append(param.name)
    if isTypeInGraph(param.eType)
    then
      sb.append(":")
      sb.append(param.eType.name)
      sb.append("[")
      sb.append(printMultiplicity(param))
      sb.append("]")
    end
    result := sb.toString
  end

```

```
  operation createInheritanceArrow(): Arrow is do
    result := Arrow.new
    result.shape := ArrowShape.normal
    result.plain := false
    result.clip := ArrowClip.none
  end

  operation createReferenceArrow(): Arrow is do
    result := Arrow.new
    result.shape := ArrowShape.vee
    result.plain := false
    result.clip := ArrowClip.none
  end

  operation createNoArrow(): Arrow is do
    result := Arrow.new
    result.shape := ArrowShape.none
    result.plain := true
    result.clip := ArrowClip.none
  end

  operation createDiamondArrow(): Arrow is do
    result := Arrow.new
    result.shape := ArrowShape.diamond
    result.plain := true
    result.clip := ArrowClip.none
  end

  operation isTypeInGraph(type: EClassifier): Boolean is do
    result := false
    if not type.isVoid
    then
      if not type.containingResource.isVoid
      then
        if type.containingResource.uri == modelURI
        then
          result := true
        end
      end
    end
  end
}
```

Listing 4. ModelOperations/E2DOT.kmt

```
@mainClass "ch::uzh::ifi::rerg::e2gen::E2GEN"
@mainOperation "main"

package ch::uzh::ifi::rerg::e2gen;

require kermeta

require "Ecore.kmt"
```

```
require "GenModel.kmt"

using kermeta::standard
using kermeta::utils
using kermeta::persistence
using kermeta::io
using ecore
using genmodel

class E2GEN
{
  operation main(inputURI: String, outputURI: String): Void is do
    var pkgs: OrderedSet<EPackage> init loadModel(inputURI)
    var gen: GenModel init createGen(pkgs, inputURI)
    saveGenModel(gen, outputURI)
  end

  operation loadModel(mmURI: String): OrderedSet<EPackage> is do
    var rep : EMFRepository init EMFRepository.new
    var res : EMFResource
    res ?= rep.getResource(mmURI)
    result := OrderedSet<EPackage>.new
    res.select{e | e.isKindOf(EPackage)}.each{p |
        result.add(p.asType(EPackage))}
  end

  operation saveGenModel(gen: GenModel, uri: String): Void is do
    var rep : EMFRepository init EMFRepository.new
    var res : EMFResource
    res ?= rep.createResource(uri, "http://www.eclipse.org/emf/2002/GenModel")
    res.add(gen)
    res.save()
  end

  operation createGen(pkgs: OrderedSet<EPackage>, inputURI: String): GenModel
      is do
    result := GenModel.new
    pkgs.each{p | result.genPackages.add(visitPackage(p))}
    result.modelDirectory := "/ch.uzh.ifi.rerg.cs.output/src"
      result.modelPluginID := "ch.uzh.ifi.rerg.cs.output"
      result.foreignModel.add(inputURI)
      result.modelName := getModelName(inputURI)
      result.importerID := "org.eclipse.emf.importer.ecore"
      result.complianceLevel := GenJDKLevel.JDK60
      result.copyrightFields := false
  end

  operation visitPackage(p: EPackage): GenPackage is do
    result := GenPackage.new
    result.ecorePackage := p
    result.disposableProviderFactory := true
    result.prefix := firstLetterUpperCase(p.name)
    result.basePackage := getBasePkgName(p)
```

```
    p.eSubpackages.each{sp | result.nestedGenPackages.add(visitPackage(sp))}
    p.eClassifiers.select{cl | cl.isKindOf(EClass)}.each{cl |
        result.genClasses.add(visitClass(cl.asType(EClass)))}
    p.eClassifiers.select{cl | cl.isKindOf(EEnum)}.each{cl |
        result.genEnums.add(visitEnum(cl.asType(EEnum)))}
    p.eClassifiers.select{cl | cl.isKindOf(EDataType) and (not
        cl.isKindOf(EEnum))}.each{cl |
        result.genDataTypes.add(visitDatatype(cl.asType(EDataType)))}
  end

  operation visitClass(c: EClass): GenClass is do
    result := GenClass.new
    result.ecoreClass := c
    result.image := not c.~abstract
    c.eStructuralFeatures.each{sf | result.genFeatures.add(visitFeature(sf))}
    c.eOperations.each{op | result.genOperations.add(visitOperation(op))}
    c.eTypeParameters.each{tp |
        result.genTypeParameters.add(visitTypeParameter(tp))}
  end

  operation visitEnum(en: EEnum): GenEnum is do
    result := GenEnum.new
    result.ecoreEnum := en
    result.typeSafeEnumCompatible := false
    en.eLiterals.each{el | result.genEnumLiterals.add(visitEnumLiteral(el))}
    en.eTypeParameters.each{tp |
        result.genTypeParameters.add(visitTypeParameter(tp))}
  end

  operation visitDatatype(dt: EDataType): GenDataType is do
    result := GenDataType.new
    result.ecoreDataType := dt
    dt.eTypeParameters.each{tp |
        result.genTypeParameters.add(visitTypeParameter(tp))}
  end

  operation visitOperation(op: EOperation): GenOperation is do
    result := GenOperation.new
    result.ecoreOperation := op
    op.eParameters.each{p | result.genParameters.add(visitParameter(p))}
  end

  operation visitParameter(p: EParameter): GenParameter is do
    result := GenParameter.new
    result.ecoreParameter := p
  end

  operation visitEnumLiteral(el: EEnumLiteral): GenEnumLiteral is do
    result := GenEnumLiteral.new
    result.ecoreEnumLiteral := el
  end
```

```
106   operation visitFeature(sf: EStructuralFeature): GenFeature is do
107     result := GenFeature.new
108     result.ecoreFeature := sf
109
110     var ref: EReference
111     ref ?= sf
112
113     if (sf.changeable)
114     then
115       result.~property := GenPropertyKind.Editable
116
117     else
118       result.~property := GenPropertyKind.Readonly
119     end
120
121     result.propertySortChoices := false
122
123     if not ref.isVoid
124     then
125
126       if ref.containment
127       then
128         result.~property := GenPropertyKind.None
129       else
130         if not ref.eOpposite.isVoid
131         then
132           if ref.eOpposite.containment
133           then
134             result.~property := GenPropertyKind.None
135           end
136         end
137       end
138
139       result.children := ref.containment
140       result.createChild := ref.containment and ref.changeable
141       result.notify := ref.containment
142       result.propertySortChoices := (result.~property ==
            GenPropertyKind.Editable)
143
144     else // EAttribute
145       result.children := false
146       result.createChild := false
147       result.notify := true
148       result.propertySortChoices := false
149     end
150   end
151
152   operation visitTypeParameter(tp: ETypeParameter): GenTypeParameter is do
153     result := GenTypeParameter.new
154     result.ecoreTypeParameter := tp
155   end
156
157   operation firstLetterUpperCase(str: String): String is do
```

```
    var letter: String init str.substring(0,1)
    var remainder: String init str.substring(1, str.size)
    result := letter.toUpperCase + remainder.toLowerCase
  end


  operation getBasePkgName(p: EPackage): String is do
    if p.eSuperPackage.isVoid
    then
      result := ""
    else
      if p.eSuperPackage.eSuperPackage.isVoid
      then
        result := p.eSuperPackage.name.toLowerCase
      else
        result := getBasePkgName(p.eSuperPackage) + "." + p.name.toLowerCase
      end
    end
  end

  operation getModelName(URI: String): String is do
    var modelName: String init URI
    from var pos: Integer init modelName.indexOf("/")
    until pos == -1
    loop
      var firstElement : String
      modelName := modelName.substring(pos+1,modelName.size)
      pos := modelName.indexOf("/")
    end

    var pos: Integer init modelName.indexOf(".")
    if pos == -1
    then
      result := modelName
    else
      result := modelName.substring(0, pos)
    end
  end
}
```

Listing 5. ModelOperations/E2GEN.kmt

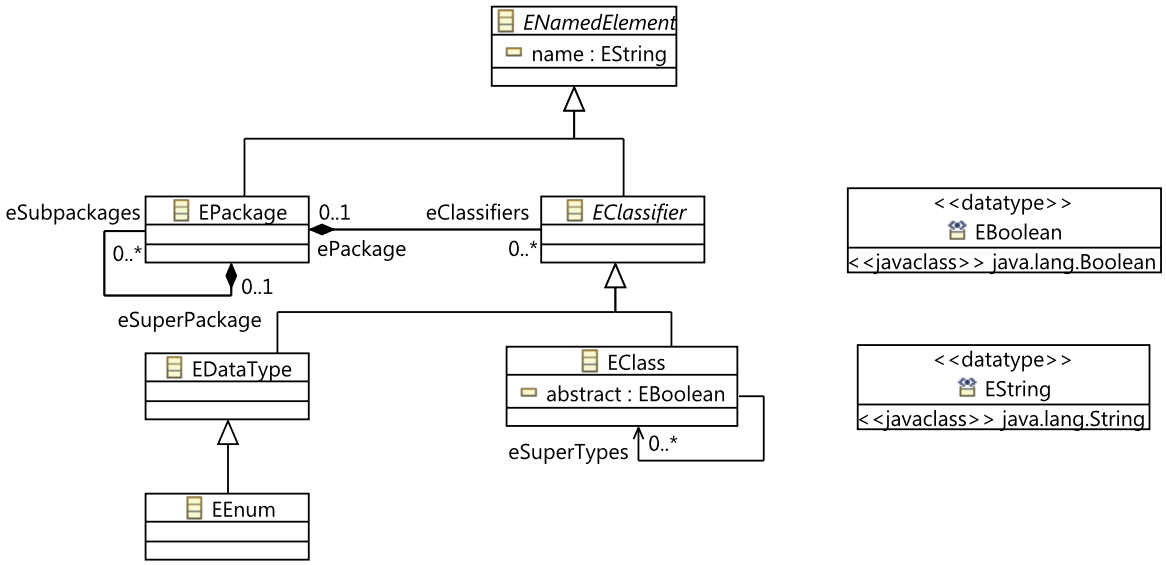

Figure 4. Static Metamodel Footprint of E2KV

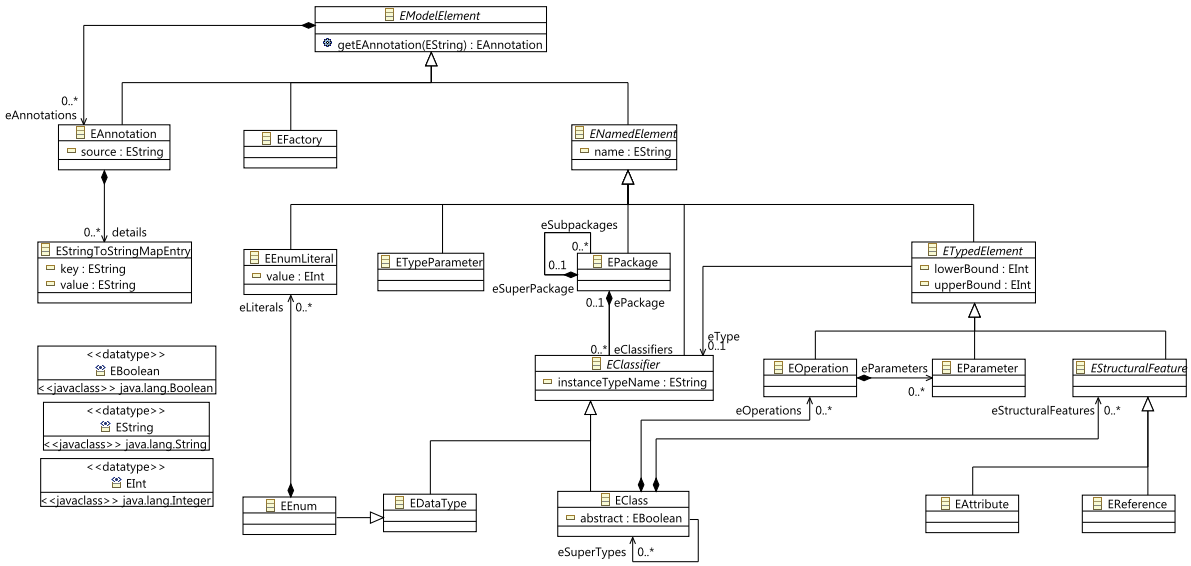

Figure 5. Static Metamodel Footprint of E2HTML

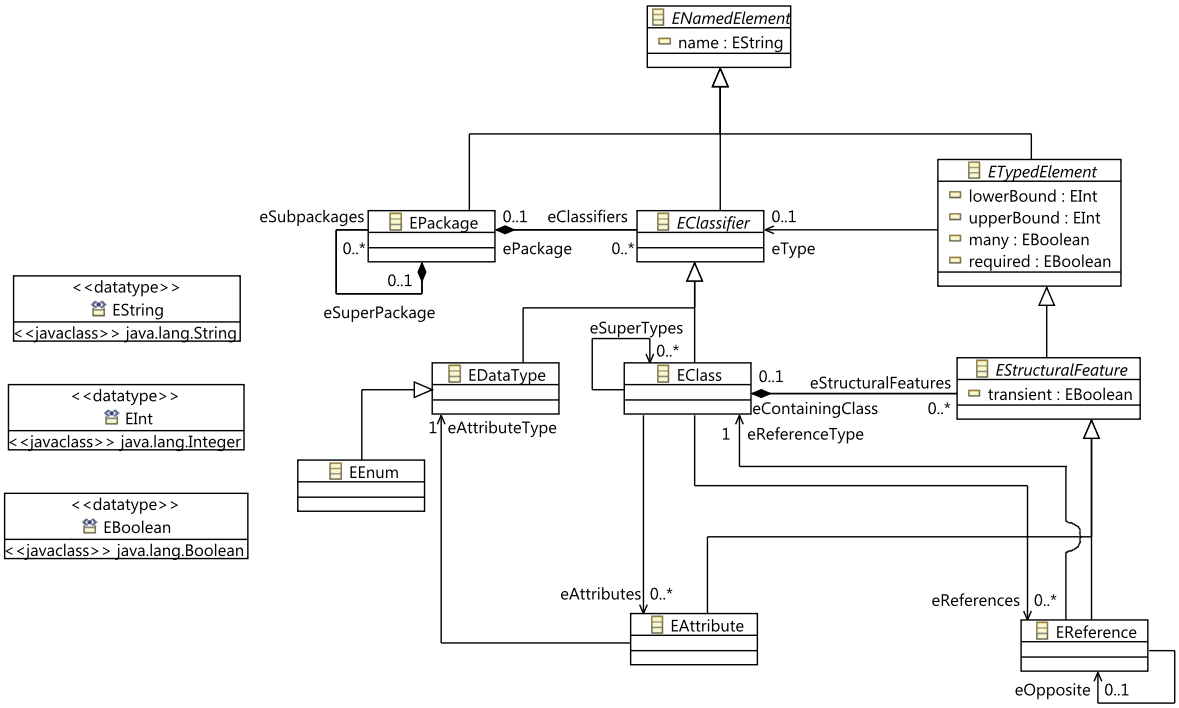

Figure 6. Static Metamodel Footprint of E2SQL

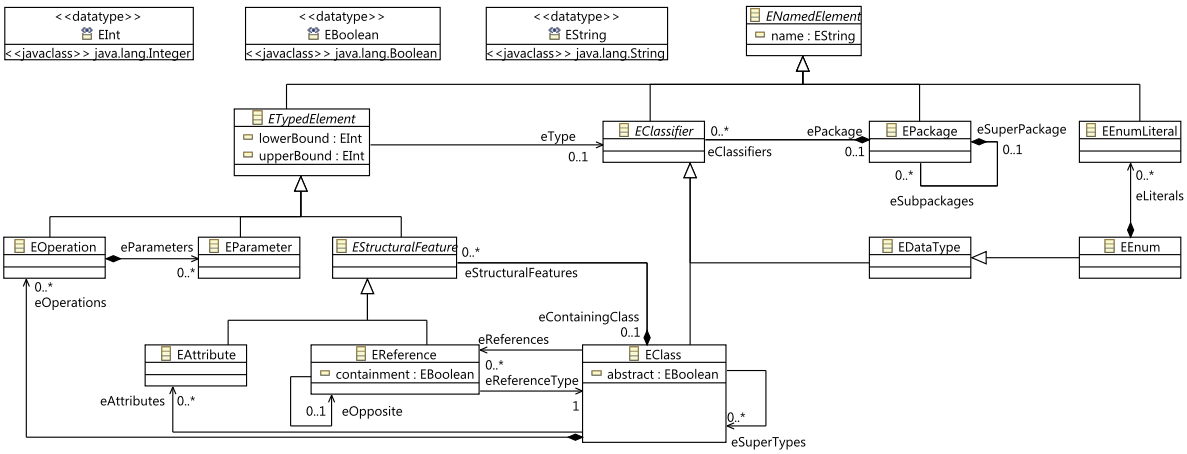

Figure 7. Static Metamodel Footprint of E2DOT

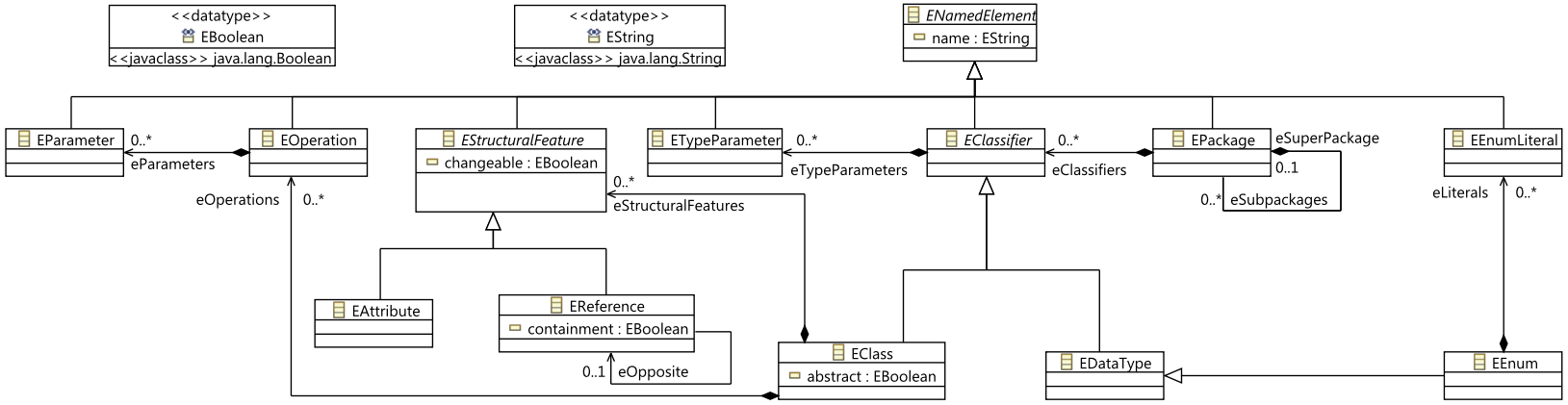

Figure 8. Static Metamodel Footprint of E2GEN

Appendix B.
Implementation of Dynamic Footprinting

```
1  package ch.uzh.ifi.rerg.cs.dynanalysis;
2
3  import java.util.HashMap;
4  import java.util.HashSet;
5  import java.util.Set;
6
7  import org.eclipse.emf.ecore.EObject;
8  import org.eclipse.emf.ecore.ETypedElement;
9
10 public class RecordTable {
11
12   private HashMap<EObject, Set<ETypedElement>> targets;
13   private HashMap<ETypedElement, Set<EObject>> features;
14
15   public RecordTable()
16   {
17     super();
18     targets = new HashMap<EObject, Set<ETypedElement>>();
19     features = new HashMap<ETypedElement, Set<EObject>>();
20   }
21
22   public void record(EObject target, ETypedElement feature)
23   {
24     Set<ETypedElement> feats = targets.get(target);
25     if(feats == null)
26     {
27       feats = new HashSet<ETypedElement>();
28       targets.put(target, feats);
29     }
30     feats.add(feature);
31
32     Set<EObject> targs = features.get(feature);
33     if(targs == null)
34     {
35       targs = new HashSet<EObject>();
36       features.put(feature, targs);
37     }
38     targs.add(target);
39   }
40
41   public Set<EObject> getTargets(ETypedElement feature)
42   {
43     return features.get(feature);
44   }
45
46   public Set<ETypedElement> getFeatures(EObject target)
47   {
48     return targets.get(target);
49   }
```

```
50
51   public Set<EObject> getTargets()
52   {
53     return targets.keySet();
54   }
55
56   public Set<ETypedElement> getFeatures()
57   {
58     return features.keySet();
59   }
60
61   public void record(EObject target)
62   {
63     Set<ETypedElement> feats = targets.get(target);
64     if(feats == null)
65     {
66       feats = new HashSet<ETypedElement>();
67       targets.put(target, feats);
68     }
69   }
70
71   public boolean exists(EObject target, ETypedElement feature)
72   {
73     Set<ETypedElement> features = targets.get(target);
74     if(features != null)
75     {
76       return features.contains(feature);
77     }
78     else
79     {
80       return false;
81     }
82   }
83 }
```

Listing 6. DynamicFootprinting/RecordTable.java

```
1 package ch.uzh.ifi.rerg.cs.dynanalysis;
2
3 import java.io.IOException;
4 import java.util.HashSet;
5 import java.util.Set;
6
7 import org.eclipse.emf.common.util.EList;
8 import org.eclipse.emf.common.util.URI;
9 import org.eclipse.emf.ecore.EAttribute;
10 import org.eclipse.emf.ecore.EClass;
11 import org.eclipse.emf.ecore.EDataType;
12 import org.eclipse.emf.ecore.EEnum;
13 import org.eclipse.emf.ecore.EObject;
14 import org.eclipse.emf.ecore.EOperation;
15 import org.eclipse.emf.ecore.EReference;
16 import org.eclipse.emf.ecore.EStructuralFeature;
```

```
17 import org.eclipse.emf.ecore.ETypedElement;
18 import org.eclipse.emf.ecore.resource.Resource;
19 import org.eclipse.emf.ecore.resource.ResourceSet;
20 import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;
21 import org.eclipse.emf.ecore.util.EcoreUtil;
22
23 import ch.uzh.ifi.rerg.cs.metrics.MetamodelSize;
24 import ch.uzh.ifi.rerg.cs.metrics.ModelSize;
25 import ch.uzh.ifi.rerg.cs.metrics.Size;
26 import ch.uzh.ifi.rerg.kextra.CallEvent;
27 import ch.uzh.ifi.rerg.kextra.Event;
28 import ch.uzh.ifi.rerg.kextra.ExecutionTrace;
29 import ch.uzh.ifi.rerg.kextra.KExtraPackage;
30 import ch.uzh.ifi.rerg.kextra.OperationCallEvent;
31 import ch.uzh.ifi.rerg.kextra.PropertyCallEvent;
32
33 public class DynAnalysis {
34
35   private ResourceSet resourceSet;
36   private Resource traceResource;
37   private RecordTable recordTable;
38   private ExecutionTrace trace;
39   private Set<EClass> classes;
40   private Set<EReference> references;
41   private Set<EAttribute> attributes;
42   private Set<EOperation> operations;
43   private Set<EDataType> datatypes;
44   private Set<EEnum> enums;
45
46   private DynAnalysis()
47   {
48     resourceSet = new ResourceSetImpl();
49     operations = new HashSet<EOperation>();
50     attributes = new HashSet<EAttribute>();
51     references = new HashSet<EReference>();
52     enums = new HashSet<EEnum>();
53     classes = new HashSet<EClass>();
54     datatypes = new HashSet<EDataType>();
55     recordTable = new RecordTable();
56   }
57
58   public DynAnalysis(URI traceURI)
59   {
60     this();
61     trace = loadTrace(traceURI);
62     proceedTrace();
63   }
64
65   private ExecutionTrace loadTrace(URI uri)
66   {
67     traceResource = resourceSet.createResource(uri);
68
69     try {
```

```
70        traceResource.load(null);
71        return (ExecutionTrace)
             EcoreUtil.getObjectByType(traceResource.getContents(),
             KExtraPackage.eINSTANCE.getExecutionTrace());
72      } catch (IOException e) {
73        e.printStackTrace();
74      }
75      return null;
76    }
77
78    private void proceedTrace() {
79      for(Event evt: trace.getEvents())
80      {
81        proceedEvent(evt);
82      }
83    }
84
85    @SuppressWarnings("unchecked")
86    private void proceedEvent(Event evt) {
87      if (evt instanceof CallEvent)
88      {
89        CallEvent ce = (CallEvent) evt;
90
91        EObject target = ce.getTarget();
92        if(target.eIsProxy())
93        {
94          //System.err.println("Warning: Unresolved Proxy: " + target);
95          return;
96        }
97        if(!isObjectInModel(target))
98        {
99          //System.err.println("EObject not in model: " + target);
100         return;
101       }
102       EClass targetClass = target.eClass();
103
104       if (evt instanceof OperationCallEvent)
105       {
106         OperationCallEvent oce = (OperationCallEvent) evt;
107         EOperation op = findOperationInEcore(oce.getFeature(), targetClass);
108
109         if(op != null)
110         {
111
112           classes.add(targetClass);
113           classes.add(op.getEContainingClass());
114           recordTable.record(target, op);
115           operations.add(op);
116         }
117       }
118       else if (evt instanceof PropertyCallEvent)
119       {
120         PropertyCallEvent pce = (PropertyCallEvent) evt;
```

```
121         EStructuralFeature feat = findPropertyInEcore(pce.getFeature(),
               targetClass);
122
123         if(feat != null)
124         {
125           classes.add(targetClass);
126           classes.add(feat.getEContainingClass());
127
128           recordTable.record(target, feat);
129
130           if (feat instanceof EAttribute) {
131             EAttribute att = (EAttribute) feat;
132             attributes.add(att);
133
134             EDataType attType = att.getEAttributeType();
135
136             if(attType instanceof EEnum)
137             {
138               enums.add((EEnum) attType);
139             }
140             else
141             {
142               datatypes.add(attType);
143             }
144
145           }
146           else if (feat instanceof EReference)
147           {
148             EReference ref = (EReference) feat;
149             references.add(ref);
150
151             if(feat.isMany())
152             {
153               EList<EObject> linkedElements = (EList<EObject>)
                     target.eGet(feat);
154               for(EObject refObj: linkedElements)
155               {
156                 if(isObjectInModel(refObj))
157                 {
158                   recordTable.record(refObj);
159                   classes.add(refObj.eClass());
160                 }
161               }
162             }
163             else
164             {
165               EObject refObj = (EObject) target.eGet(feat);
166               if(refObj != null && isObjectInModel(refObj))
167               {
168                 recordTable.record(refObj);
169                 classes.add(refObj.eClass());
170               }
171             }
```

```
172          }
173        }
174      }
175    }
176  }
177
178  private boolean isObjectInModel(EObject target) {
179    return target.eResource() != null &&
         target.eResource().getURI().isPlatform();
180  }
181
182  private EStructuralFeature findPropertyInEcore(String name, EClass cl)
183  {
184    for(EStructuralFeature candidate : cl.getEAllStructuralFeatures())
185    {
186      if(name.equals(candidate.getName()))
187        return candidate;
188    }
189    //System.err.println("Feature " + name + " not found.");
190    return null;
191  }
192
193  private EOperation findOperationInEcore(String name, EClass cl)
194  {
195    for(EOperation candidate : cl.getEAllOperations())
196    {
197      if(name.equals(candidate.getName()))
198        return candidate;
199    }
200    //System.err.println("Operation " + name + " not found.");
201    return null;
202  }
203
204  public Size getDynamicModelFootprintSize()
205  {
206    int nObjects = recordTable.getTargets().size();
207    int nPointers = 0;
208    int nValues = 0;
209    int nOperations = 0;
210
211    for(ETypedElement feature: recordTable.getFeatures())
212    {
213      int n = recordTable.getTargets(feature).size();
214      if(feature instanceof EOperation)
215      {
216        nOperations += n;
217      }
218      else if(feature instanceof EAttribute)
219      {
220        nValues += n;
221      }
222      else if(feature instanceof EReference)
223      {
```

```
224         nPointers += n;
225       }
226     }
227
228     return new ModelSize(nObjects,nPointers,nValues,nOperations);
229   }
230
231   public Size getDynamicMetaModelFootprintSize()
232   {
233     int nClasses = classes.size();
234     int nAttributes = attributes.size();
235     int nReferences = references.size();
236     int nEnums = enums.size();
237     int nPrimitiveTypes = datatypes.size();
238     int nOperations = operations.size();
239     int nEnumLiterals = 0;
240     for(EEnum en: enums)
241     {
242       nEnumLiterals += en.getELiterals().size();
243     }
244     return new
          MetamodelSize(nClasses,nAttributes,nReferences,nEnums,nEnumLiterals,nPrimitiveTypes
245   }
246
247   public boolean isFeatureUsed(ETypedElement el)
248   {
249     return recordTable.getFeatures().contains(el);
250   }
251
252   public boolean isObjectUsed(EObject el)
253   {
254     return recordTable.getTargets().contains(el);
255   }
256
257   public boolean isFeatureUsedOnTarget(ETypedElement feat, EObject obj)
258   {
259     return recordTable.exists(obj, feat);
260   }
261
262   public int getTraceLength() {
263     return trace.getEvents().size();
264   }
265 }
```

Listing 7. DynamicFootprinting/DynAnalysis.java

Appendix C.
Implementation of Static Footprinting

```
package kermeta::language::structure;

require kermeta
using kermeta::standard

/**
 * Unfortunately, weaving/loading opposite references doesn't work yet
 * Work around: Preparatory phase
 */

aspect class Object
{
  reference usedBy: set NamedElement[0..*]

  property isUsed: Boolean
    getter is do
      result := not usedBy.isEmpty
    end

  attribute isRequired: Boolean

  @Overloadable "true"
  operation usedIn(element: NamedElement): Void is do
    usedBy.add(element)
  end
}

aspect class ClassDefinition inherits GenericTypeDefinition
{
  operation findOperation(name: String): Operation is do
    result := self.allOperation.select{o | name == o.name}.one
  end
  //reference superType: set Type[0..*]#subClasses
  reference subClasses: set ClassDefinition[0..*]#superClasses
  reference superClasses: set ClassDefinition[0..*]#subClasses

  property allSuperClasses: oset ClassDefinition[0..*]
  getter is do
    result := OrderedSet<ClassDefinition>.new
    result.addAll(superClasses)
    self.superClasses.each { t | result.addAll(t.allSuperClasses) }
  end

  property allSubClasses: oset ClassDefinition[0..*]
  getter is do
    result := OrderedSet<ClassDefinition>.new
    result.addAll(subClasses)
    self.subClasses.each { t | result.addAll(t.allSubClasses) }
  end
```

```
  property numberOfAttributes: Integer
  getter is do
    result := ownedAttribute.size
  end
}

aspect class Package
{
  reference modelingUnit: ModelingUnit [0..1]

  property allNestedPackages: set Package [0..*]
    getter is do
      result := Set<Package>.new
      result.add(self)
      nestedPackage.each{p | result.addAll(p.allNestedPackages)}
    end

  property allNestedClassDefinitions: set ClassDefinition [0..*]
    getter is do
      result := Set<ClassDefinition>.new
      ownedTypeDefinition.select{td |
         td.isInstanceOf(ClassDefinition)}.each{cd |
         result.add(cd.asType(ClassDefinition))}
      nestedPackage.each{p | result.addAll(p.allNestedClassDefinitions)}
    end

  property allNestedAttributes: set Property [0..*]
    getter is do
      result := Set<Property>.new
      allNestedClassDefinitions.each{cd | result.addAll(cd.ownedAttribute)}
    end

  property numberOfPackages: Integer
    getter is do
      result := 1
      nestedPackage.each{p | result := result + p.numberOfPackages}
    end

  property numberOfClasses: Integer
    getter is do
      result := ownedTypeDefinition.select{td |
         td.isInstanceOf(ClassDefinition)}.size
      nestedPackage.each{p | result := result + p.numberOfClasses}
    end

  property numberOfAttributes: Integer
    getter is do
      result := 0
      ownedTypeDefinition.select{td |
         td.isInstanceOf(ClassDefinition)}.each{c | result := result +
         c.asType(ClassDefinition).numberOfAttributes}
      nestedPackage.each{p | result := result + p.numberOfAttributes}
```

```
    end
}

aspect class ModelingUnit
{
  property allNestedPackages: set Package [0..*]
    getter is do
      result := Set<Package>.new
      packages.each{p | result.addAll(p.allNestedPackages)}
    end

  property allNestedClassDefinitions: set ClassDefinition [0..*]
    getter is do
      result := Set<ClassDefinition>.new
      packages.each{p | result.addAll(p.allNestedClassDefinitions)}
    end

  property allNestedAttributes: set Property [0..*]
    getter is do
      result := Set<Property>.new
      allNestedClassDefinitions.each{cd | result.addAll(cd.ownedAttribute)}
    end

  property numberOfPackages: Integer
    getter is do
      result := 0
      packages.each{p | result := result + p.numberOfPackages}
    end

  property numberOfClasses: Integer
    getter is do
      result := 0
      packages.each{p | result := result + p.numberOfClasses}
    end
  property numberOfAttributes: Integer
    getter is do
      result := 0
      packages.each{p | result := result + p.numberOfAttributes}
    end
}

aspect class Operation inherits MultiplicityElement
{
  reference subOperation: set Operation [0..*]//#superOperation
  //reference superOperation: Operation[0..1]#subOperation
}
```

Listing 8. StaticFootprinting/EnhancedKermeta.kmt

```
package kermeta;

require "kermeta"
using kermeta::language::behavior
```

```
using kermeta::language
using kermeta::language::structure
abstract class KermetaVisitor
{
  operation visitObject(arg : language::structure::Object) : Void is
    do
    end

  operation visitCallVariable(arg : language::behavior::CallVariable) : Void
      is
    do
      self.visitObject(arg)
    end

  operation visitMultiplicityElement(arg :
      language::structure::MultiplicityElement) : Void is
    do
      self.visitObject(arg)
    end

  operation visitModelTypeVariable(arg :
      language::structure::ModelTypeVariable) : Void is
    do
      self.visitObject(arg)
    end

  operation visitRequire(arg : language::structure::Require) : Void is
    do
      self.visitObject(arg)
    end

  operation visitTag(arg : language::structure::Tag) : Void is
    do
      self.visitObject(arg)
    end

  operation visitVirtualType(arg : language::structure::VirtualType) : Void is
    do
      self.visitObject(arg)
    end

  operation visitTypeLiteral(arg : language::behavior::TypeLiteral) : Void is
    do
      self.visitObject(arg)
    end

  operation visitEnumerationLiteral(arg :
      language::structure::EnumerationLiteral) : Void is
    do
      self.visitObject(arg)
    end

```

```
  operation visitLambdaParameter(arg : language::behavior::LambdaParameter) : Void is
    do
      self.visitObject(arg)
    end

  operation visitTypeVariableBinding(arg : language::structure::TypeVariableBinding) : Void is
    do
      self.visitObject(arg)
    end

  operation visitIntegerLiteral(arg : language::behavior::IntegerLiteral) : Void is
    do
      self.visitObject(arg)
    end

  operation visitModel(arg : language::structure::Model) : Void is
    do
      self.visitObject(arg)
    end

  operation visitCallFeature(arg : language::behavior::CallFeature) : Void is
    do
      self.visitObject(arg)
    end

  operation visitTypeReference(arg : language::behavior::TypeReference) : Void is
    do
      self.visitObject(arg)
    end

  operation visitJavaStaticCall(arg : language::behavior::JavaStaticCall) : Void is
    do
      self.visitObject(arg)
    end

  operation visitProperty(arg : language::structure::Property) : Void is
    do
      self.visitObject(arg)
    end

  operation visitClass(arg : language::structure::Class) : Void is
    do
      self.visitObject(arg)
    end

  operation visitLoop(arg : language::behavior::Loop) : Void is
    do
      self.visitObject(arg)
```

```
    end

  operation visitConditional(arg : language::behavior::Conditional) : Void is
    do
      self.visitObject(arg)
    end

  operation visitStringLiteral(arg : language::behavior::StringLiteral) :
      Void is
    do
      self.visitObject(arg)
    end

  operation visitRescue(arg : language::behavior::Rescue) : Void is
    do
      self.visitObject(arg)
    end

  operation visitTypeDefinition(arg : language::structure::TypeDefinition) :
      Void is
    do
      self.visitObject(arg)
    end

  operation visitCallValue(arg : language::behavior::CallValue) : Void is
    do
      self.visitObject(arg)
    end

  operation visitCallSuperOperation(arg :
      language::behavior::CallSuperOperation) : Void is
    do
      self.visitObject(arg)
    end

  operation visitOperation(arg : language::structure::Operation) : Void is
    do
      self.visitObject(arg)
    end

  operation visitUsing(arg : language::structure::Using) : Void is
    do
      self.visitObject(arg)
    end

  operation visitModelType(arg : language::structure::ModelType) : Void is
    do
      self.visitObject(arg)
    end

  operation visitRaise(arg : language::behavior::Raise) : Void is
    do
      self.visitObject(arg)
```

```
    end

  operation visitConstraint(arg : language::structure::Constraint) : Void is
    do
      self.visitObject(arg)
    end

  operation visitVoidLiteral(arg : language::behavior::VoidLiteral) : Void is
    do
      self.visitObject(arg)
    end

  operation visitFunctionType(arg : language::structure::FunctionType) : Void
      is
    do
      self.visitObject(arg)
    end

  operation visitBlock(arg : language::behavior::Block) : Void is
    do
      self.visitObject(arg)
    end

  operation visitAssignment(arg : language::behavior::Assignment) : Void is
    do
      self.visitObject(arg)
    end

  operation visitFilter(arg : language::structure::Filter) : Void is
    do
      self.visitObject(arg)
    end

  operation visitEnumeration(arg : language::structure::Enumeration) : Void is
    do
      self.visitObject(arg)
    end

  operation visitEmptyExpression(arg : language::behavior::EmptyExpression) :
      Void is
    do
      self.visitObject(arg)
    end

  operation visitObjectTypeVariable(arg :
      language::structure::ObjectTypeVariable) : Void is
    do
      self.visitObject(arg)
    end

  operation visitPackage(arg : language::structure::Package) : Void is
    do
      self.visitObject(arg)
```

```
    end

  operation visitCallResult(arg : language::behavior::CallResult) : Void is
    do
      self.visitObject(arg)
    end

  operation visitClassDefinition(arg : language::structure::ClassDefinition)
      : Void is
    do
      self.visitObject(arg)
    end

  operation visitBooleanLiteral(arg : language::behavior::BooleanLiteral) :
      Void is
    do
      self.visitObject(arg)
    end

  operation visitParameter(arg : language::structure::Parameter) : Void is
    do
      self.visitObject(arg)
    end

  operation visitLambdaExpression(arg : language::behavior::LambdaExpression)
      : Void is
    do
      self.visitObject(arg)
    end

  operation visitPrimitiveType(arg : language::structure::PrimitiveType) :
      Void is
    do
      self.visitObject(arg)
    end

  operation visitVariableDecl(arg : language::behavior::VariableDecl) : Void
      is
    do
      self.visitObject(arg)
    end

  operation visitModelingUnit(arg : language::structure::ModelingUnit) : Void
      is
    do
      self.visitObject(arg)
    end

  operation visitSelfExpression(arg : language::behavior::SelfExpression) :
      Void is
    do
      self.visitObject(arg)
    end
```

```
  operation visitType(arg : language::structure::Type) : Void is
    do
      self.visitObject(arg)
    end

  operation visitVoidType(arg : language::structure::VoidType) : Void is
    do
      self.visitObject(arg)
    end

  operation visitProductType(arg : language::structure::ProductType) : Void is
    do
      self.visitObject(arg)
    end

}
package language
{
  package behavior
  {
    aspect class CallVariable inherits CallExpression
    {
      method accept(v : KermetaVisitor) : Void from
          kermeta::language::structure::Object is
        do
          v.visitCallVariable(self)
        end

    }
    aspect class TypeLiteral inherits Literal
    {
      method accept(v : KermetaVisitor) : Void from
          kermeta::language::structure::Object is
        do
          v.visitTypeLiteral(self)
        end

    }
    aspect class LambdaParameter
    {
      method accept(v : KermetaVisitor) : Void from
          kermeta::language::structure::Object is
        do
          v.visitLambdaParameter(self)
        end

    }
    aspect class IntegerLiteral inherits Literal
    {
      method accept(v : KermetaVisitor) : Void from
          kermeta::language::structure::Object is
        do
```

```
296          v.visitIntegerLiteral(self)
297        end
298
299      }
300      aspect class CallFeature inherits CallExpression
301      {
302        method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
303        do
304          v.visitCallFeature(self)
305        end
306
307      }
308      aspect abstract class CallExpression inherits Expression
309      {
310      }
311      aspect class TypeReference inherits
           language::structure::MultiplicityElement
312      {
313        method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
314        do
315          v.visitTypeReference(self)
316        end
317
318      }
319      aspect class JavaStaticCall inherits Expression
320      {
321        method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
322        do
323          v.visitJavaStaticCall(self)
324        end
325
326      }
327      aspect class Loop inherits Expression
328      {
329        method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
330        do
331          v.visitLoop(self)
332        end
333
334      }
335      aspect class Conditional inherits Expression
336      {
337        method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
338        do
339          v.visitConditional(self)
340        end
341
342      }
```

```
343     aspect class StringLiteral inherits Literal
344     {
345       method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
346         do
347           v.visitStringLiteral(self)
348         end
349
350     }
351     aspect class Rescue
352     {
353       method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
354         do
355           v.visitRescue(self)
356         end
357
358     }
359     aspect class CallValue inherits CallExpression
360     {
361       method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
362         do
363           v.visitCallValue(self)
364         end
365
366     }
367     aspect abstract class Literal inherits Expression
368     {
369     }
370     aspect class CallSuperOperation inherits CallExpression
371     {
372       method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
373         do
374           v.visitCallSuperOperation(self)
375         end
376
377     }
378     aspect class Raise inherits Expression
379     {
380       method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
381         do
382           v.visitRaise(self)
383         end
384
385     }
386     aspect class VoidLiteral inherits Literal
387     {
388       method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
389         do
```

```
390          v.visitVoidLiteral(self)
391        end
392
393      }
394      aspect class Block inherits Expression
395      {
396        method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
397        do
398          v.visitBlock(self)
399        end
400
401      }
402      aspect class Assignment inherits Expression
403      {
404        method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
405        do
406          v.visitAssignment(self)
407        end
408
409      }
410      aspect class EmptyExpression inherits Expression
411      {
412        method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
413        do
414          v.visitEmptyExpression(self)
415        end
416
417      }
418      aspect class CallResult inherits CallVariable
419      {
420        method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
421        do
422          v.visitCallResult(self)
423        end
424
425      }
426      aspect class BooleanLiteral inherits Literal
427      {
428        method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
429        do
430          v.visitBooleanLiteral(self)
431        end
432
433      }
434      aspect class LambdaExpression inherits Expression
435      {
436        method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
```

```
437         do
438           v.visitLambdaExpression(self)
439         end
440
441     }
442     aspect class VariableDecl inherits Expression
443     {
444       method accept(v : KermetaVisitor) : Void from
            kermeta::language::structure::Object is
445         do
446           v.visitVariableDecl(self)
447         end
448
449     }
450     aspect class SelfExpression inherits Expression
451     {
452       method accept(v : KermetaVisitor) : Void from
            kermeta::language::structure::Object is
453         do
454           v.visitSelfExpression(self)
455         end
456
457     }
458     aspect abstract class Expression inherits
          language::structure::TypeContainer
459     {
460     }
461   }
462
463   package structure
464   {
465     aspect abstract class TypeVariable inherits TypeContainer, Type,
          NamedElement
466     {
467     }
468     aspect class MultiplicityElement inherits TypedElement
469     {
470       method accept(v : KermetaVisitor) : Void from
            kermeta::language::structure::Object is
471         do
472           v.visitMultiplicityElement(self)
473         end
474
475     }
476     aspect class ModelTypeVariable inherits TypeVariable
477     {
478       method accept(v : KermetaVisitor) : Void from
            kermeta::language::structure::Object is
479         do
480           v.visitModelTypeVariable(self)
481         end
482
483     }
```

```
484     aspect class Require
485     {
486       method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
487         do
488           v.visitRequire(self)
489         end
490
491     }
492     aspect class Tag
493     {
494       method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
495         do
496           v.visitTag(self)
497         end
498
499     }
500     aspect class VirtualType inherits ObjectTypeVariable
501     {
502       method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
503         do
504           v.visitVirtualType(self)
505         end
506
507     }
508     aspect abstract class TypeDefinitionContainer inherits NamedElement
509     {
510     }
511     aspect class EnumerationLiteral inherits NamedElement
512     {
513       method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
514         do
515           v.visitEnumerationLiteral(self)
516         end
517
518     }
519     aspect class TypeVariableBinding inherits TypeContainer
520     {
521       method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
522         do
523           v.visitTypeVariableBinding(self)
524         end
525
526     }
527     aspect class Model
528     {
529       method accept(v : KermetaVisitor) : Void from
             kermeta::language::structure::Object is
530         do
```

```
531          v.visitModel(self)
532        end
533
534    }
535    aspect class Property inherits MultiplicityElement
536    {
537      method accept(v : KermetaVisitor) : Void from
           kermeta::language::structure::Object is
538        do
539          v.visitProperty(self)
540        end
541
542    }
543    aspect class Class inherits ParameterizedType
544    {
545      method accept(v : KermetaVisitor) : Void from
           kermeta::language::structure::Object is
546        do
547          v.visitClass(self)
548        end
549
550    }
551    aspect class TypeDefinition inherits NamedElement
552    {
553      method accept(v : KermetaVisitor) : Void from
           kermeta::language::structure::Object is
554        do
555          v.visitTypeDefinition(self)
556        end
557
558    }
559    aspect abstract class DataType inherits Type, TypeDefinition
560    {
561    }
562    aspect class Operation inherits MultiplicityElement
563    {
564      method accept(v : KermetaVisitor) : Void from
           kermeta::language::structure::Object is
565        do
566          v.visitOperation(self)
567        end
568
569    }
570    aspect class Using
571    {
572      method accept(v : KermetaVisitor) : Void from
           kermeta::language::structure::Object is
573        do
574          v.visitUsing(self)
575        end
576
577    }
578    aspect abstract class ParameterizedType inherits Type
```

```
579     {
580     }
581     aspect class ModelType inherits Type, TypeDefinition
582     {
583       method accept(v : KermetaVisitor) : Void from
            kermeta::language::structure::Object is
584         do
585           v.visitModelType(self)
586         end
587
588     }
589     aspect class Constraint inherits NamedElement
590     {
591       method accept(v : KermetaVisitor) : Void from
            kermeta::language::structure::Object is
592         do
593           v.visitConstraint(self)
594         end
595
596     }
597     aspect class FunctionType inherits TypeContainer, Type
598     {
599       method accept(v : KermetaVisitor) : Void from
            kermeta::language::structure::Object is
600         do
601           v.visitFunctionType(self)
602         end
603
604     }
605     aspect abstract class GenericTypeDefinition inherits TypeDefinition
606     {
607     }
608     aspect abstract class TypeContainer
609     {
610     }
611     aspect abstract class NamedElement
612     {
613     }
614     aspect class Filter
615     {
616       method accept(v : KermetaVisitor) : Void from
            kermeta::language::structure::Object is
617         do
618           v.visitFilter(self)
619         end
620
621     }
622     aspect class Enumeration inherits DataType
623     {
624       method accept(v : KermetaVisitor) : Void from
            kermeta::language::structure::Object is
625         do
626           v.visitEnumeration(self)
```

```
627         end
628
629     }
630     aspect abstract class TypedElement inherits TypeContainer, NamedElement
631     {
632     }
633     aspect class Object
634     {
635       @Overloadable "true"
636       operation accept(v : KermetaVisitor) : Void is
637         do
638           v.visitObject(self)
639         end
640
641     }
642     aspect class ObjectTypeVariable inherits TypeVariable
643     {
644       method accept(v : KermetaVisitor) : Void from
            kermeta::language::structure::Object is
645         do
646           v.visitObjectTypeVariable(self)
647         end
648
649     }
650     aspect class Package inherits NamedElement, TypeDefinitionContainer
651     {
652       method accept(v : KermetaVisitor) : Void from
            kermeta::language::structure::Object is
653         do
654           v.visitPackage(self)
655         end
656
657     }
658     aspect class ClassDefinition inherits GenericTypeDefinition, TypeContainer
659     {
660       method accept(v : KermetaVisitor) : Void from
            kermeta::language::structure::Object is
661         do
662           v.visitClassDefinition(self)
663         end
664
665     }
666     aspect class Parameter inherits MultiplicityElement
667     {
668       method accept(v : KermetaVisitor) : Void from
            kermeta::language::structure::Object is
669         do
670           v.visitParameter(self)
671         end
672
673     }
674     aspect class PrimitiveType inherits DataType, TypeContainer
675     {
```

```
      method accept(v : KermetaVisitor) : Void from
          kermeta::language::structure::Object is
        do
          v.visitPrimitiveType(self)
        end

    }
    aspect class ModelingUnit
    {
      method accept(v : KermetaVisitor) : Void from
          kermeta::language::structure::Object is
        do
          v.visitModelingUnit(self)
        end

    }
    aspect class Type
    {
      method accept(v : KermetaVisitor) : Void from
          kermeta::language::structure::Object is
        do
          v.visitType(self)
        end

    }
    aspect class VoidType inherits Type
    {
      method accept(v : KermetaVisitor) : Void from
          kermeta::language::structure::Object is
        do
          v.visitVoidType(self)
        end

    }
    aspect class ProductType inherits TypeContainer, Type
    {
      method accept(v : KermetaVisitor) : Void from
          kermeta::language::structure::Object is
        do
          v.visitProductType(self)
        end

    }
  }

}
```

Listing 9. StaticFootprinting/VisitableKermeta.kmt

```
@mainClass "ch::uzh::ifi::rerg::smmfp::Main"
@mainOperation "main"

package ch::uzh::ifi::rerg::smmfp;
```

```
5
6  require kermeta
7  require "EnhancedKermeta.kmt"
8  require "PrepareTransformation.kmt"
9  require "AnalyzeTransformation.kmt"
10 require "DisplayStaticMetamodelFootprint.kmt"
11 require "CompleteStaticMetamodelFootprint.kmt"
12 require "CreateStaticFootprintMetamodel.kmt"
13
14 using kermeta::language::structure
15 using kermeta::language::behavior
16 using kermeta::persistence
17 using kermeta::utils
18 using kermeta::standard
19
20 class Main
21 {
22
23   attribute qClassNameRegistry: Hashtable<String, ClassDefinition>
24   attribute qPackageURIRegistry: Hashtable<String, Package>
25
26   operation main(transformationURI: String, inputMetamodelNSURI: String,
        staticMetamodelFileURI: String, staticMetamodelNSURI: String): Void is
        do
27     var transformationUnit: ModelingUnit init
          loadTransformation(transformationURI)
28
29     qClassNameRegistry := Hashtable<String, ClassDefinition>.new
30     qPackageURIRegistry := Hashtable<String, Package>.new
31
32     // Prepare Transformation for Analysis
33     prepareTransformation(transformationUnit)
34
35     var inputMetamodel: Package init
          qPackageURIRegistry.getValue(inputMetamodelNSURI)
36     var inputMetamodelUnit: ModelingUnit init inputMetamodel.modelingUnit
37
38     // Find main operation
39     var mainOperation: Operation init findMainOperation(transformationUnit)
40
41     // Analyze it for class/op/prop usage
42     analyzeTransformation(mainOperation)
43
44     // Complete the static metamodel footprint
45     completeStaticMetamodelFootprint(transformationUnit)
46
47     // Display the results
48     displayStaticMetamodelFootprint(inputMetamodelUnit)
49
50     // Create the static footprint metamodel
51     createStaticFootprintMetamodel(inputMetamodelUnit, staticMetamodelNSURI)
52
53     // Save the static footprint metamodel
```

```
    saveMetamodel(inputMetamodelUnit, staticMetamodelFileURI)

  end

  operation loadTransformation(muURI: String ): ModelingUnit is do
    var rep : EMFRepository init EMFRepository.new
    var res : EMFResource
    res ?= rep.getResource(muURI)
    result ?= res.one
  end

  operation saveMetamodel(mu: ModelingUnit, mmURI: String): Void is do
    do
      var rep : EMFRepository init EMFRepository.new
        var res : EMFResource
        // note: Metametamodel doesn't change
      res ?= rep.createResource(mmURI,
          "http://www.kermeta.org/kermeta/1_2_0//kermeta")
      res.add(mu)
      // diagnoseDanglingElements(res)
      res.save
    rescue (se: kermeta::exceptions::ResourceSaveException)
      //Because of the type annotations, there are some dangling elements.
      //A simple type check will fix it later
    end

  end

  operation prepareTransformation(mu: ModelingUnit): Void is do
    var tpv: PrepareTransformation init PrepareTransformation.new
    tpv.classes := qClassNameRegistry
    tpv.packages := qPackageURIRegistry
    mu.accept(tpv)
  end

  operation findMainOperation(mu: ModelingUnit): Operation is do
    var qMainClassName: String init mu.tag.select{t|
        t.name=="mainClass"}.one.~value
    var mainOpName: String init mu.tag.select{t|
        t.name=="mainOperation"}.one.~value
    result :=
        qClassNameRegistry.getValue(qMainClassName).findOperation(mainOpName)
  end

  operation analyzeTransformation(op: Operation): Void is do
    var uav: AnalyzeTransformation init AnalyzeTransformation.new
    op.accept(uav)
  end

  operation displayStaticMetamodelFootprint(mu: ModelingUnit): Void is do
    var urv: DisplayStaticMetamodelFootprint init
        DisplayStaticMetamodelFootprint.new
    mu.accept(urv)
```

```
  end

  operation completeStaticMetamodelFootprint(mu: ModelingUnit): Void is do
    var rpv: CompleteStaticMetamodelFootprint init
        CompleteStaticMetamodelFootprint.new
    mu.accept(rpv)
    mu.referencedModelingUnits.each{m | m.accept(rpv)}
  end

  operation createStaticFootprintMetamodel(mu: ModelingUnit, pkgNSURI:
      String): Void is do
    var mpv: CreateStaticFootprintMetamodel init
        CreateStaticFootprintMetamodel.new
    mpv.baseURI := pkgNSURI
    mu.accept(mpv)
  end

  operation diagnoseDanglingElements(res: EMFResource): Void is do
    var report: Set<DanglingDiagnostic> init res.findDanglingModelElements()
    report.each{d | stdio.writeln(d.toString)}
  end
}
```

Listing 10. StaticFootprinting/Main.kmt

```
package ch::uzh::ifi::rerg::smmfp;

require kermeta
require "VisitableKermeta.kmt"
require "EnhancedKermeta.kmt"


using kermeta::language::structure
using kermeta::utils
using kermeta

/**
 * Unfortunately, weaving/loading opposite references doesn't work yet
 * Work around: Preparatory phase
 */

class PrepareTransformation inherits KermetaVisitor
{
  reference classes: Hashtable<String, ClassDefinition>
  reference packages: Hashtable<String, Package>
  reference currentMU: ModelingUnit

  method visitModelingUnit(m: ModelingUnit) : Void is do
    currentMU := m
    m.packages.each{p | p.accept(self)}
    m.referencedModelingUnits.each{m | m.accept(self)}
  end

```

```
  method visitPackage(p: Package) : Void is do
    packages.put(p.uri,p)
    p.modelingUnit := currentMU
    p.ownedTypeDefinition.each{td | td.accept(self) }
    p.nestedPackage.each{p | p.accept(self)}
  end

  method visitClassDefinition(c: ClassDefinition) : Void is do
    classes.put(c.qualifiedName,c)
    c.superType.each{t |
        c.superClasses.add(t.asType(Class).typeDefinition.asType(ClassDefinition))}
    c.ownedOperation.each{o | o.accept(self)}
  end

  method visitOperation(o: Operation) : Void is do
    if not o.superOperation.isVoid
    then
      o.superOperation.subOperation.add(o)
    end
  end
}
```

Listing 11. StaticFootprinting/PrepareTransformation.kmt

```
package ch::uzh::ifi::rerg::smmfp;

require kermeta
require "VisitableKermeta.kmt"
require "EnhancedKermeta.kmt"

using kermeta
using kermeta::language::structure
using kermeta::language::behavior

class AnalyzeTransformation inherits KermetaVisitor
{
  reference elementUnderAnalysis: NamedElement

  /**
   * Structural Elements
   * (mark them - and their refinement/specialization - as they are visited)
   */

  method visitConstraint(c : Constraint) : Void is do
    if c.language == ConstraintLanguage.kermeta
    then
      var previousEUA: NamedElement init elementUnderAnalysis
      elementUnderAnalysis := c
      c.body.accept(self)
      elementUnderAnalysis := previousEUA
    else
      stdio.writeln("Warning: Unhandled non-OCL constraint")
    end
```

```
  end

  method visitClassDefinition(cd : ClassDefinition) : Void is do
    // Don't do the work twice ;)
    // Assumption: x.isUsed <=> x.hasBeenVisited
    if cd.isUsed
    then
      cd.usedIn(elementUnderAnalysis)
    else
      cd.usedIn(elementUnderAnalysis)
      cd.~inv.each{i | i.accept(self)}
      cd.typeParameter.each{p | p.accept(self)}
      if not "kermeta::language::structure::Object".equals(cd.qualifiedName)
      then
        cd.subClasses.each{c | c.accept(self)}
      end
    end
  end

  method visitProperty(p : Property) : Void is do
    // Don't do the work twice ;)
    // Assumption: x.isUsed <=> x.hasBeenVisited
    if p.isUsed
    then
      p.usedIn(elementUnderAnalysis)
    else
      p.usedIn(elementUnderAnalysis)

      var previousEUA: NamedElement init elementUnderAnalysis
      elementUnderAnalysis := p

      p.type.accept(self)

      if not p.getterBody.isVoid
      then
        p.getterBody.accept(self)
      end

      if not p.setterBody.isVoid
      then
        p.setterBody.accept(self)
      end

      elementUnderAnalysis := previousEUA
    end
  end

  method visitOperation(o: Operation) : Void is do
     // Don't do the work twice ;)
     // Assumption: x.isUsed <=> x.hasBeenVisited
    if o.isUsed
    then
      o.usedIn(elementUnderAnalysis)
```

```
    else
      o.usedIn(elementUnderAnalysis)
      var previousEUA: NamedElement init elementUnderAnalysis
      elementUnderAnalysis := o

      o.ownedParameter.each{p | p.accept(self)}
      o.type.accept(self)
      o.~pre.each{c|c.accept(self)}
      o.~post.each{c|c.accept(self)}

      if not o.body.isVoid
      then
        o.body.accept(self)
      end
      elementUnderAnalysis := previousEUA
      o.subOperation.each{o| o.accept(self)}
    end
  end

  method visitParameter(p: Parameter): Void is do
    p.type.accept(self)
  end

  method visitEnumeration(e : Enumeration) : Void is do
    e.usedIn(elementUnderAnalysis)
  end

  method visitPrimitiveType(p : PrimitiveType) : Void is do
    p.usedIn(elementUnderAnalysis)
    p.instanceType.accept(self)
  end

  /**
   * Control Flow elements
   * (mark their type and navigate through control flow)
   */

  method visitBlock(b: Block): Void is do
    b.statement.each{s | s.accept(self)}
    b.rescueBlock.each{b | b.accept(self)}
  end

  method visitLoop(l: Loop): Void is do
    l.initialization.accept(self)
    l.stopCondition.accept(self)
    l.body.accept(self)
  end

  method visitConditional(c: Conditional): Void is do
    c.condition.accept(self)
    c.thenBody.accept(self)
    if not(c.elseBody.isVoid)
    then
```

```
      c.elseBody.accept(self)
    end
  end

  method visitRescue(r : Rescue) : Void is do
    r.exceptionType.accept(self)
    r.body.accept(self)
  end

  /**
   * Expressions
   * (mark their type and visit subexpressions)
   */

  method visitRaise(r : Raise) : Void is do
    r.expression.accept(self)
    r.staticType.accept(self)
  end

  method visitVariableDecl(v: VariableDecl): Void is do
    //v.type.accept(self)
    if not(v.initialization.isVoid)
    then
      v.initialization.accept(self)
    end
  end

  method visitAssignment(a: Assignment): Void is do
    a.~value.accept(self)
    a.target.accept(self)
  end

  method visitSelfExpression(s: SelfExpression): Void is do
    s.staticType.usedIn(elementUnderAnalysis)
  end

  method visitCallVariable(c: CallVariable): Void is do
    c.parameters.each{p | p.accept(self)}
  end

  method visitCallResult(c: CallResult): Void is do
    // Nothing to do
  end

  method visitCallSuperOperation(c: CallSuperOperation): Void is do
    c.parameters.each{p | p.accept(self)}
    elementUnderAnalysis.asType(Operation).superOperation.accept(self)
  end

  method visitJavaStaticCall(j : JavaStaticCall) : Void is do
    // Nothing to do
  end

```

```
  method visitCallFeature(c: CallFeature): Void is do
    if not c.target.isVoid
    then
      c.target.accept(self)
      c.target.staticType.accept(self)
    end

    if not(c.staticOperation.isVoid)
    then
      c.staticOperation.accept(self)
    end

    if not (c.staticEnumLiteral.isVoid)
    then
      c.staticEnumLiteral.accept(self)
    end

    if not (c.staticProperty.isVoid)
    then
      c.staticProperty.accept(self)
    end
    c.parameters.each{p | p.accept(self)}
  end

  method visitLambdaExpression(l: LambdaExpression): Void is do
    l.body.accept(self)
    l.parameters.each{p | p.accept(self)}
  end

  method visitLambdaParameter(lp: LambdaParameter): Void is do
    if not lp.type.isVoid
    then
      lp.type.accept(self)
    end
  end

  /**
   * Literals
   */

  method visitTypeLiteral(tl : TypeLiteral) : Void is do
    tl.typeref.accept(self)
  end

  method visitIntegerLiteral(l: IntegerLiteral) : Void is do
    //nothing to do
  end

  method visitStringLiteral(l: StringLiteral) : Void is do
    //nothing to do
  end

  method visitBooleanLiteral(l: BooleanLiteral) : Void is do
```

```
    //nothing to do
  end

  method visitVoidLiteral(l: VoidLiteral) : Void is do
    //nothing to do
  end

  method visitEnumerationLiteral(el : EnumerationLiteral) : Void is do
    //nothing to do
  end

  /**
   * Types
   * (Visit the structural elements they encompass or navigate through
     variables)
   */

  method visitProductType(p : ProductType) : Void is do
    p.type.each{t | t.accept(self)}
  end

  method visitFunctionType(f : FunctionType) : Void is do
    f.left.accept(self)
    f.right.accept(self)
  end

  method visitClass(c : Class) : Void is do
    c.typeDefinition.accept(self)
    c.typeParamBinding.each{b | b.accept(self)}
  end

  method visitTypeReference(t : TypeReference) : Void is do
    t.type.accept(self)
  end

  method visitVoidType(v : VoidType) : Void is do
    // Nothing to do
  end

  method visitModelType(m : ModelType) : Void is do
    m.includedTypeDefinition.each{t| t.accept(self)}
  end

  method visitTypeVariableBinding(t : TypeVariableBinding) : Void is do
    t.type.accept(self)
  end

  method visitObjectTypeVariable(o : ObjectTypeVariable) : Void is do
    //nothing to do
  end

  method visitModelTypeVariable(m : ModelTypeVariable) : Void is do
    m.virtualType.each{v | v.accept(self)}
```

```
  end

  method visitVirtualType(v : VirtualType) : Void is do
    v.typeParamBinding.each{b | b.accept(self)}
  end

  method visitObject(o: Object): Void is do
    if not o.isVoid
    then
      stdio.writeln("Warning: Unhandled " +
          o.getMetaClass.typeDefinition.qualifiedName + " instance: " +
          o.toString)
    end
  end
}
```

Listing 12. StaticFootprinting/AnalyzeTransformation.kmt

```
package ch::uzh::ifi::rerg::smmfp;

require kermeta
require "VisitableKermeta.kmt"
require "EnhancedKermeta.kmt"

using kermeta
using kermeta::standard
using kermeta::language::structure
using kermeta::language::behavior
using kermeta::utils

class CreateStaticFootprintMetamodel inherits KermetaVisitor
{
  attribute baseURI: String

  method visitModelingUnit(mu: ModelingUnit): Void is do
    mu.referencedModelingUnits.clear
    mu.packages.select{x | not x.isRequired}.each{x | mu.packages.remove(x)}
    mu.packages.each{p | p.accept(self)}

  end

  method visitPackage(p: Package) : Void is do

    p.nestedPackage.select{x | not x.isRequired}.each{x |
        p.nestedPackage.remove(x)}
    p.ownedTypeDefinition.select{x | not x.isRequired}.each{x |
        p.ownedTypeDefinition.remove(x)}

    if p.nestingPackage.isVoid
    then
      p.uri := baseURI
    else
      p.uri := p.nestingPackage.uri + "/" + p.name
```

```
    end

    p.nestedPackage.each{p | p.accept(self)}
    p.ownedTypeDefinition.each{td | td.accept(self)}

  end

  method visitOperation(o: Operation) : Void is do
    documentUsageInTags(o)
  end

  method visitProperty(p: Property): Void is do
    documentUsageInTags(p)
    if not p.opposite.isVoid
    then
      if not p.opposite.isRequired
      then
        var op: Property init p.opposite
        // BUG: Do not unlink opposite... Why?
        // WORKAROUND: Clone the property
        cloneProperty(p)
      end
    end

  end

  method visitClassDefinition(c: ClassDefinition) : Void is do
    c.ownedAttribute.select{x | not x.isRequired}.each{x |
        c.ownedAttribute.remove(x)}
    c.ownedOperation.select{x | not x.isRequired}.each{x |
        c.ownedOperation.remove(x)}

    c.ownedAttribute.each{p | p.accept(self)}
    c.ownedOperation.each{o | o.accept(self)}

    fixInheritance(c)
    documentUsageInTags(c)
  end

  method visitEnumeration(e: Enumeration): Void is do
    documentUsageInTags(e)
  end

  method visitPrimitiveType(t: PrimitiveType): Void is do
    documentUsageInTags(t)
    if not t.instanceType.isVoid
    then
      var  it: Type init t.instanceType
      if not it.typeContainer.equals(t)
      then
        if it.isInstanceOf(Class)
        then
          var nIT: Class init Class.new
```

```
          nIT.typeDefinition := it.asType(Class).typeDefinition
          t.instanceType := nIT
          t.containedType.add(nIT)
        end
      end
    end
  end

  operation cloneProperty(p: Property): Property is do

    var np: Property init Property.new
    np.name := p.name

    var cd: ClassDefinition init p.owningClass
    cd.ownedAttribute.remove(p)
    cd.ownedAttribute.add(np)

    np.isComposite := p.isComposite
    np.isID := p.isID
    np.default := p.default
    np.isReadOnly := p.isReadOnly

    np.isDerived := p.isDerived
    np.isGetterAbstract := p.isGetterAbstract
    np.isSetterAbstract := p.isSetterAbstract
    np.getterBody := p.getterBody
    np.setterBody := p.setterBody

    np.upper := p.upper
    np.lower := p.lower
    np.isOrdered := p.isOrdered
    np.isUnique := p.isUnique

    np.containedType.addAll(p.containedType)
    np.type := p.type

    np.tag.addAll(p.tag)
    np.ownedTags.addAll(p.ownedTags)

    np.usedBy.addAll(p.usedBy)
    np.isRequired := p.isRequired
    //We don't clone opposite ;)
    result := np
  end

  operation fixInheritance(c: ClassDefinition): Void is do
    var supClasses: OrderedSet<ClassDefinition>  init
       c.allSuperClasses.select{t | t.isRequired}.asOrderedSet

    c.superType.clear
    c.containedType.clear
    c.superClasses.clear

```

```
    supClasses.each{s |
      if not c.allSuperClasses.contains(s)
      then
        var cl: Class init Class.new
        cl.typeDefinition := s
        c.superType.add(cl)
        c.containedType.add(cl)
        c.superClasses.add(s)
      end
      }
  end

  operation documentUsageInTags(o: Object): Void is do
    if(o.isUsed)
    then
      var tag: Tag init Tag.new
      tag.name := "UsedBy"
      var sb: StringBuffer init StringBuffer.new

      o.usedBy.each{ne |
        if sb.length != 0 then sb.append("\n") end
        if not ne.isVoid
        then
          if ne.isInstanceOf(Operation) then sb.append("operation ") end
          if ne.isInstanceOf(Property) then sb.append("property ") end
          if ne.isInstanceOf(Constraint)
          then
            var c: Constraint init ne.asType(Constraint)
            if c.stereotype != ConstraintType.~inv then sb.append("invariant
                ") end
            if c.stereotype != ConstraintType.~pre  then
                sb.append("precondition ") end
            if c.stereotype != ConstraintType.~post  then
                sb.append("postcondition ") end
          end

          sb.append(ne.qualifiedName)

        end }

      tag.~value := sb.toString()
      o.tag.add(tag)
      o.ownedTags.add(tag)
    end
  end
}
```

Listing 13. StaticFootprinting/CreateStaticFootprintMetamodel.kmt

```
package ch::uzh::ifi::rerg::smmfp;

require kermeta
require "VisitableKermeta.kmt"
```

```
5  require "EnhancedKermeta.kmt"
6
7  using kermeta
8  using kermeta::language::structure
9  using kermeta::language::behavior
10
11 class DisplayStaticMetamodelFootprint inherits KermetaVisitor
12 {
13   attribute indent: String
14   reference currentOperation: Operation
15
16   method visitModelingUnit(mu: ModelingUnit): Void is do
17     stdio.writeln("--------------------------------")
18     stdio.writeln("Model: " + mu.containingResource.uri)
19     stdio.writeln("Metamodel: " + mu.containingResource.metaModelURI)
20     stdio.writeln("--------------------------------")
21     indent := ""
22     mu.packages.each{p | p.accept(self)}
23   end
24
25   method visitPackage(p: Package) : Void is do
26     if printObject(p)
27     then
28       stdio.write(indent + "pa: " + p.name)
29       writeStatus(p)
30
31       var oldIndent: String init indent
32       indent := oldIndent + "  "
33       p.nestedPackage.each{p | p.accept(self)}
34       p.ownedTypeDefinition.each{td | td.accept(self) }
35
36       indent := oldIndent
37     end
38   end
39
40   method visitOperation(o: Operation) : Void is do
41     if printObject(o)
42     then
43       stdio.write(indent + "op: " + o.name)
44       writeStatus(o)
45     end
46   end
47
48   method visitProperty(p: Property): Void is do
49     if printObject(p)
50     then
51       stdio.write(indent + "at: " + p.name)
52       writeStatus(p)
53     end
54   end
55
56   method visitClassDefinition(c: ClassDefinition) : Void is do
57     if printObject(c)
```

```
    then
      stdio.write(indent + "cd: " + c.name)
      writeStatus(c)

      var oldIndent: String init indent
      indent := oldIndent + "  "
      c.ownedOperation.each{o | o.accept(self)}
      c.ownedAttribute.each{p | p.accept(self)}
      indent := oldIndent
    end
  end

  method visitEnumeration(e: Enumeration): Void is do
    if printObject(e)
    then
      stdio.write(indent + "en: " + e.name)
      writeStatus(e)

      var oldIndent: String init indent
      indent := oldIndent + "  "
      e.ownedLiteral.each{l | l.accept(self)}
      indent := oldIndent
    end
  end

  method visitEnumerationLiteral(e: EnumerationLiteral): Void is do
    if printObject(e)
    then
      stdio.write(indent + "el: " + e.name)
      writeStatus(e)
    end
  end

  method visitPrimitiveType(t: PrimitiveType): Void is do
    if printObject(t)
    then
      stdio.write(indent + "pt: " + t.name)
      writeStatus(t)
    end
  end

  operation writeStatus(o: Object): Void is do
    if o.isRequired != void
    then
      if o.isRequired and (not o.isUsed)
      then
        stdio.write(" [ADDITIONAL]")
      end
    end
    stdio.writeln("")
  end


```

```

  operation printObject(o: Object): Boolean is do
    result := o.isUsed or o.isRequired
  end
}
```

Listing 14. StaticFootprinting/DisplayStaticMetamodelFootprint.kmt

```
package ch::uzh::ifi::rerg::smmfp;

require kermeta
require "VisitableKermeta.kmt"
require "EnhancedKermeta.kmt"

using kermeta
using kermeta::language::structure
using kermeta::language::behavior

class CompleteStaticMetamodelFootprint inherits KermetaVisitor
{

  method visitModelingUnit(mu: ModelingUnit): Void is do
    mu.packages.each{p | p.accept(self)}
    mu.usings.each{u | u.accept(self)}
    mu.requires.each{r | r.accept(self)}
    mu.isRequired := true
  end

  method visitUsing(u: Using): Void is do
    u.isRequired := true // TODO: Check for not required packages
  end

  method visitRequire(r: Require): Void is do
    r.isRequired := true // TODO: Check for empty MU...
  end

  method visitPackage(p: Package) : Void is do
    p.nestedPackage.each{x | x.accept(self)}
    p.ownedTypeDefinition.each{td | td.accept(self) }

    p.isRequired := p.nestedPackage.exists{x | x.isRequired} or
        p.ownedTypeDefinition.exists{x | x.isRequired}

  end

  method visitOperation(o: Operation) : Void is do
    o.isRequired := o.isUsed
    o.ownedParameter.each{p | p.isRequired := true}
  end

  method visitProperty(p: Property): Void is do
    p.isRequired := p.isUsed

```

```
    //To create effective metamodel, include opposites...
    //To create effective metamodel, include mandatory properties...
  end

  method visitClassDefinition(c: ClassDefinition) : Void is do
    c.ownedOperation.each{o | o.accept(self)}
    c.ownedAttribute.each{p | p.accept(self)}
    c.isRequired := c.isUsed
      or c.ownedAttribute.exists{x | x.isRequired}
      or c.ownedOperation.exists{x | x.isRequired}
      or c.allSuperClasses.exists{x | x.isUsed and (x.qualifiedName !=
          "kermeta::language::structure::Object")}
    //To create effective metamodel, include  super classes...
  end

  method visitEnumeration(e: Enumeration): Void is do
    //If the enum (or one of its literal) is marked as used
    //We keep it entirely
    e.isRequired := e.isUsed or e.ownedLiteral.exists{l | l.isUsed}
    if e.isUsed
    then
      e.ownedLiteral.each{l | l.isRequired := true}
    else
      e.ownedLiteral.each{l | l.isRequired := false}
    end
  end

  method visitEnumerationLiteral(e: EnumerationLiteral): Void is do
    // Nothing to do
  end

  method visitPrimitiveType(t: PrimitiveType): Void is do
    t.isRequired := t.isUsed
  end
}
```

Listing 15. StaticFootprinting/CompleteStaticMetamodelFootprint.kmt

```
package ch.uzh.ifi.rerg.mfpe;

import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;

import org.eclipse.emf.ecore.EAnnotation;
import org.eclipse.emf.ecore.EAttribute;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.EClassifier;
import org.eclipse.emf.ecore.EDataType;
import org.eclipse.emf.ecore.EEnum;
import org.eclipse.emf.ecore.EEnumLiteral;
import org.eclipse.emf.ecore.EModelElement;
import org.eclipse.emf.ecore.ENamedElement;
```

```
16 import org.eclipse.emf.ecore.EPackage;
17 import org.eclipse.emf.ecore.EReference;
18 import org.eclipse.emf.ecore.EStructuralFeature;
19
20 public class MetamodelMatcher {
21   private static final String USED_BY_ANNOTATION_ID = "UsedBy";
22
23   private Collection<EClass> usedClasses;
24   private Collection<EClass> rootClasses;
25   private Collection<EClass> requiredClasses;
26   private Collection<EStructuralFeature> requiredFeatures;
27
28   private HashMap<ENamedElement, ENamedElement> c2fMapping;
29   private HashMap<ENamedElement, ENamedElement> f2cMapping;
30
31   public MetamodelMatcher(EPackage complete, EPackage pruned)
32   {
33     usedClasses = new HashSet<EClass>();
34     rootClasses = new HashSet<EClass>();
35     requiredClasses = new HashSet<EClass>();
36     requiredFeatures = new HashSet<EStructuralFeature>();
37
38     c2fMapping = new HashMap<ENamedElement, ENamedElement>();
39     f2cMapping = new HashMap<ENamedElement, ENamedElement>();
40
41     matchedWith(complete, pruned);
42     findRootClasses();
43   }
44
45   public boolean isClassRequired(EClass cl)
46   {
47     return requiredClasses.contains(cl);
48   }
49
50   public boolean isClassUsed(EClass cl)
51   {
52     if(usedClasses.isEmpty())
53     {
54       return requiredClasses.contains(cl);
55     }
56     else
57     {
58       return usedClasses.contains(cl);
59     }
60   }
61
62   public boolean isFeatureRequired(EStructuralFeature feat)
63   {
64     return requiredFeatures.contains(feat);
65   }
66
67   public boolean isRootClass(EClass cl)
68   {
```

```
    return rootClasses.contains(cl);
  }

  private void matchedWith(EClass c, EClass f)
  {
    if(f != null)
    {
      requiredClasses.add(c);
      c2fMapping.put(c, f);
      f2cMapping.put(f, c);

      if(hasUsageAnnotation(f))
        usedClasses.add(c);

      for(EStructuralFeature next : c.getEStructuralFeatures())
      {
        if (next instanceof EAttribute) {
          EAttribute nextC = (EAttribute) next;
          EAttribute nextP = findAttributeInClass(f, nextC.getName());
          matchedWith(nextC, nextP);
        }
        else if(next instanceof EReference)
        {
          EReference nextC = (EReference) next;
          EReference nextP = findReferenceInClass(f, nextC.getName());
          matchedWith(nextC, nextP);
        }
      }
    }
  }

  private void matchedWith(EReference c, EReference f)
  {
    if(f != null)
    {
      requiredFeatures.add(c);
      c2fMapping.put(c, f);
      f2cMapping.put(f, c);
    }
  }


  private void matchedWith(EPackage c, EPackage f)
  {
    if(f != null)
    {
      c2fMapping.put(c, f);
      f2cMapping.put(f, c);

      for(EClassifier next : c.getEClassifiers())
      {
        if (next instanceof EClass) {
          EClass nextC = (EClass) next;
```

```
          EClass nextP = findClassInPackage(f, nextC.getName());
          matchedWith(nextC, nextP);
        }

        if (next instanceof EDataType) {
          EDataType nextC = (EDataType) next;
          EDataType nextP = findDataTypeInPackage(f, nextC.getName());
          matchedWith(nextC, nextP);
        }
      }
    }
  }

  private void matchedWith(EAttribute c, EAttribute f)
  {
    if(f != null)
    {
      requiredFeatures.add(c);
      c2fMapping.put(c, f);
      f2cMapping.put(f, c);
    }
  }

  private void matchedWith(EDataType c, EDataType f)
  {
    if(f != null)
    {
      c2fMapping.put(c, f);
      f2cMapping.put(f, c);

      if (c instanceof EEnum)
      {
        EEnum enumC = (EEnum) c;
        EEnum enumP = (EEnum) f;

        for (EEnumLiteral nextC : enumC.getELiterals())
        {
          EEnumLiteral nextP = enumP.getEEnumLiteral(nextC.getName());
          matchedWith(nextC, nextP);
        }
      }
    }
  }

  private void matchedWith(EEnumLiteral c, EEnumLiteral f)
  {
    if(f != null)
    {
      c2fMapping.put(c, f);
      f2cMapping.put(f, c);
    }
  }

```

```
175   private boolean hasUsageAnnotation(EModelElement elem)
176   {
177     if(elem == null)
178       return false;
179     EAnnotation an = elem.getEAnnotation("kermeta");
180     if(an == null)
181       return false;
182     return an.getDetails().containsKey(USED_BY_ANNOTATION_ID);
183   }
184
185   private EClass findClassInPackage(EPackage pkg, String name)
186   {
187     for(EClassifier next : pkg.getEClassifiers())
188     {
189       if(next instanceof EClass && name.equals(next.getName()))
190         return (EClass) next;
191     }
192     return null;
193   }
194
195   private EDataType findDataTypeInPackage(EPackage pkg, String name)
196   {
197     for(EClassifier next : pkg.getEClassifiers())
198     {
199       if(next instanceof EDataType && name.equals(next.getName()))
200         return (EDataType) next;
201     }
202     return null;
203   }
204
205   private EAttribute findAttributeInClass(EClass cl, String name)
206   {
207     for(EAttribute next : cl.getEAttributes())
208     {
209       if(name.equals(next.getName()))
210         return next;
211     }
212     return null;
213   }
214
215   private EReference findReferenceInClass(EClass cl, String name)
216   {
217     for(EReference next : cl.getEReferences())
218     {
219       if(name.equals(next.getName()))
220         return next;
221     }
222     return null;
223   }
224
225   private void findRootClasses()
226   {
227     rootClasses.addAll(requiredClasses);
```

```
228     for(EStructuralFeature feat: requiredFeatures)
229     {
230       if (feat instanceof EReference) {
231         EReference ref = (EReference) feat;
232         if(ref.isContainment())
233         {
234           EClass contained = ref.getEReferenceType();
235           EClass container = ref.getEContainingClass();
236
237           for(EClass reqClass: requiredClasses)
238           {
239             if(contained != container)
240             {
241               rootClasses.remove(contained);
242             }
243             if(reqClass.getEAllSuperTypes().contains(contained))
244             {
245               if(reqClass != container)
246               {
247                 rootClasses.remove(reqClass);
248               }
249             }
250           }
251         }
252       }
253     }
254     if(!usedClasses.isEmpty())
255     {
256       rootClasses.retainAll(usedClasses);
257     }
258   }
259
260   public void diagnose()
261   {
262     StringBuffer buf = new StringBuffer();
263     for(EClass cl: requiredClasses)
264     {
265       buf.append(cl.getName());
266       if(!isClassUsed(cl))
267       {
268         buf.append(" [NOT USED]");
269       }
270       if(isRootClass(cl))
271       {
272         buf.append(" [ROOT]");
273       }
274
275       EClass pClass = (EClass) c2fMapping.get(cl);
276       if(!pClass.getName().equals(cl.getName()))
277       {
278         buf.append(" (--> " + pClass.getName() +")");
279       }
280       buf.append('\n');
```

```

      for(EStructuralFeature feat: cl.getEStructuralFeatures())
      {
        if(isFeatureRequired(feat))
        {
          if(feat instanceof EReference)
            buf.append("  ref: ");
          else
            buf.append("  att: ");
          buf.append(feat.getName());
          buf.append('\n');
        }
      }
    }
    System.out.println(buf.toString());
  }

  public ENamedElement getFootprintCorrespondance(ENamedElement en)
  {
    return c2fMapping.get(en);
  }

  public ENamedElement getCompleteCorrespondance(ENamedElement en)
  {
    return f2cMapping.get(en);
  }

}
```

Listing 16. StaticFootprinting/MetamodelMatcher.java

```
package ch.uzh.ifi.rerg.mfpe;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;

import org.eclipse.emf.common.util.EList;
import org.eclipse.emf.common.util.TreeIterator;
import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.EAttribute;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.EObject;
import org.eclipse.emf.ecore.EPackage;
import org.eclipse.emf.ecore.EReference;
import org.eclipse.emf.ecore.EStructuralFeature;
import org.eclipse.emf.ecore.EcorePackage;
import org.eclipse.emf.ecore.resource.Resource;
import org.eclipse.emf.ecore.resource.ResourceSet;
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;
import org.eclipse.emf.ecore.util.EcoreUtil;

```

```
23
24 public class ModelFilter {
25
26   private ResourceSet resourceSet;
27   protected MetamodelMatcher matcher;
28
29   private HashMap<EObject, EObject> f2pTrace; // From Complete to Footprint
30   private HashMap<EObject, EObject> p2fTrace; // From Footprint to Complete
31
32
33
34   public ModelFilter()
35   {
36     resourceSet = new ResourceSetImpl();
37   }
38
39   public ModelFilter(EPackage complete, URI pruned)
40   {
41     this();
42     //registerMetamodel(complete);
43     EPackage pMM = loadMetamodel(pruned);
44     setMetamodels(complete, pMM);
45   }
46
47   protected void setMetamodels(EPackage c, EPackage p)
48   {
49     matcher = new MetamodelMatcher(c,p);
50     //matcher.diagnose();
51   }
52
53   protected EPackage loadMetamodel(URI uri)
54   {
55     Resource res = resourceSet.createResource(uri);
56
57     try {
58       res.load(null);
59     } catch (IOException e) {
60       e.printStackTrace();
61     }
62
63     return (EPackage) EcoreUtil.getObjectByType(res.getContents(),
         EcorePackage.eINSTANCE.getEPackage());
64   }
65
66   public void saveModel(Collection<EObject> model, URI uri)
67   {
68     Resource res = resourceSet.createResource(uri);
69
70     for (EObject eObject : model)
71     {
72       if(eObject.eContainer() == null)
73         res.getContents().add(eObject);
74     }
```

```
75      try {
76        res.save(null);
77      } catch (IOException e) {
78        e.printStackTrace();
79      }
80    }
81
82    protected void registerMetamodel(EPackage mm)
83    {
84      resourceSet.getPackageRegistry().put(mm.getNsURI(), mm);
85    }
86
87    public Collection<EObject> loadModel(URI uri)
88    {
89      Resource res = resourceSet.createResource(uri);
90
91      try {
92        res.load(null);
93      } catch (IOException e) {
94        e.printStackTrace();
95      }
96      Collection<EObject> model = new ArrayList<EObject>();
97
98      for(TreeIterator<EObject> it = res.getAllContents();it.hasNext();)
99      {
100       model.add(it.next());
101     }
102     return model;
103   }
104
105   public Collection<EObject> loadModel(EObject root)
106   {
107     Collection<EObject> model = new ArrayList<EObject>();
108
109     for(TreeIterator<EObject> it = root.eAllContents();it.hasNext();)
110     {
111       model.add(it.next());
112     }
113     return model;
114   }
115
116   public void proceedModelFile(URI completeModelURI, URI prunedModelURI)
117   {
118     Collection<EObject> model = loadModel(completeModelURI);
119     model = sliceModel(model);
120     saveModel(model, prunedModelURI);
121   }
122
123
124   public Collection<EObject> sliceModel(Collection<EObject> model)
125   {
126     resetTrace();
127     return setFeatures(createObjects(model));
```

```
128   }
129
130
131   private Collection<EObject> createObjects(Collection<EObject> model)
132   {
133     Collection<EObject> result = new ArrayList<EObject>();
134
135     for (EObject cObj : model)
136     {
137       if(keepObject(cObj))
138       {
139         EClass pClass = (EClass)
               matcher.getFootprintCorrespondance(cObj.eClass());
140         EPackage pPkg = pClass.getEPackage();
141         EObject pObj = pPkg.getEFactoryInstance().create(pClass);
142         result.add(pObj);
143         f2pTrace.put(cObj, pObj);
144         p2fTrace.put(pObj, cObj);
145       }
146     }
147
148     return result;
149   }
150   @SuppressWarnings("unchecked")
151   private Collection<EObject> setFeatures(Collection<EObject> model)
152   {
153     for(EObject fObj : model)
154     {
155       EObject cObj = p2fTrace.get(fObj);
156
157       for(EStructuralFeature pFeat :
             fObj.eClass().getEAllStructuralFeatures())
158       {
159         if(!pFeat.isDerived() && pFeat.isChangeable() && !fObj.eIsSet(pFeat))
160         {
161           if (pFeat instanceof EAttribute)
162           {
163             EAttribute pAtt = (EAttribute) pFeat;
164             EAttribute cAtt = (EAttribute)
                   matcher.getCompleteCorrespondance(pAtt);
165
166             Object cVal = cObj.eGet(cAtt);
167
168             fObj.eSet(pAtt,cVal);
169           }
170           else if (pFeat instanceof EReference)
171           {
172
173             EReference pRef = (EReference) pFeat;
174             EReference cRef = (EReference)
                   matcher.getCompleteCorrespondance(pFeat);
175             if(pRef.isMany())
176             {
```

```
177                 EList<EObject> cList = (EList<EObject>) cObj.eGet(cRef);
178                 EList<EObject> pList = (EList<EObject>) fObj.eGet(pRef);
179
180                 for(EObject cVal : cList)
181                 {
182                   EObject pVal = f2pTrace.get(cVal);
183                   if(pVal != null)
184                   {
185                     pList.add(pVal);
186                   }
187                   else if(cVal.eIsProxy())
188                   {
189                     System.err.println("WARNING: Proxy met.");
190                   }
191                 }
192               }
193
194               else
195               {
196                 EObject cVal = (EObject) cObj.eGet(cRef);
197                 EObject pVal = f2pTrace.get(cVal);
198
199                 if(cVal != null && pVal == null)
200                 {
201                   // EMF returns an object that is not part of the model
202                   // Typically, something about generic types...
203                   System.err.println("WARNING: Weird EMF behavior.");
204                 }
205
206                 fObj.eSet(pRef, pVal);
207
208               }
209             }
210           }
211
212         }
213       }
214
215       return model;
216     }
217
218     private void resetTrace()
219     {
220       f2pTrace = new HashMap<EObject, EObject>(); // From Complete to Prune
221       p2fTrace = new HashMap<EObject, EObject>(); // From Prune to Complete
222     }
223
224     private boolean keepObject(EObject obj)
225     {
226       return objectIsUsed(obj);
227     }
228
229     private boolean objectIsUsed(EObject obj)
```

```
  {
    return matcher.isClassUsed(obj.eClass());
  }
}
```

Listing 17. StaticFootprinting/ModelFilter.java