Verifying Ecore Models using Lightning

Christophe Kamphaus
Laboratory for Advanced Software Systems
University of Luxembourg
6, rue R. Coudenhove-Kalergi
L-1359 Luxembourg

TR-LASSY-14-01

Verifying Ecore Models using Lightning

Author:
Christophe Kamphaus

Supervisor:
Prof. Dr. Pierre Kelsen

University of Luxembourg
Faculty of Science, Technology and Communication

Bachelor Thesis
Bachelor en Sciences et Ingénierie (Acadmémique) - Filière Informatique
February 2014

Declaration of Authorship

I, Christophe KAMPHAUS, declare that this thesis titled, 'Verifying Ecore Models using Lightning' and the work presented in it are my own. I confirm that:

- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Abstract

The Lightning tool provides an environment for developing modeling languages based on Alloy. The object of this thesis is to equip the tool with the capability to import Ecore models and export Ecore model instances. This permits the verification of the many existing Ecore models with Alloy's automatic instance generation.

First we work out the model transformation from Ecore models to Alloy models making use of UML2Alloy, exploring three different methods to transform Ecore to UML and then mapping the EMF UML objects to the UML2Alloy internal UML object representation.

The second model transformation from Alloy instance to Ecore instance dynamically loads the Ecore model which generated the Alloy model to reuse the mapping information and use it to create Ecore instance objects.

Then those two transformations are integrated into Lightning and a standalone UI is introduced to help with batch transformations of larger number of models and for testing purposes.

Finally the effectiveness of the transformations is measured with test models. This shows a successful transformation for most models, but that constraints still pose problems. Another limitation is that multiple inheritance is not supported.

Acknowledgements

First I would like to thank my supervisor Professor Pierre Kelsen for giving me this interesting topic. His interest in my progress also drove me to achieve my best. His course about Model-Driven Software Development was also quite helpful for this thesis.

I would like to thank Christian Glodt and Loïc Gammaitoni for their advice and stimulating conversations and for taking the time to debug parts of the program and explain Lightning respectively.

Finally I would also like to thank my family for their support.

Contents

List of Figures

List of Tables

List of Listings

Abbreviations

A2E	**A**lloy to **E**core
API	**A**pplication **P**rogramming **I**nterface
ASM	**A**bstract **S**yntax **M**odel
E2A	**E**core to **A**lloy
E2U	**E**core to **U**ML
EMF	**E**clipse **M**odeling **F**ramework
GUI	**G**raphical **U**ser **I**nterface
MDA	**M**odel **D**riven **A**rchitecture
MDD	**M**odel **D**riven **D**evelopment
MDE	**M**odel **D**riven **E**ngineering
MDSD	**M**odel **D**riven **S**oftware **D**evelopment
MOF	**M**eta **O**bject **F**acility
NS	**N**ame**S**pace
OCL	**O**bject **C**onstraint **L**anguage
OMG	**O**bject **M**anagement **G**roup
QVTo	**Q**uery/**V**iew/**T**ransformation **o**perational
UI	**U**ser **I**nterface
UML	**U**nified **M**odeling **L**anguage
UUID	**U**niversally **U**nique **Id**entifier
XMI	**X**ML **M**etadata **I**nterchange
XML	e**X**tensible **M**arkup **L**anguage

Chapter 1

Introduction

1.1 Problem Definition

"Verifying Ecore Models using Lightning"

Lightning is a first step towards an Alloy language workbench. The aim of this thesis is to first extend Lightning to be able to import Ecore models by transforming them into Alloy models and to be able to export generated model instances into Ecore model instances and to second be able to verify Ecore models by having the ability to inspect generated model instances in Lightning and to validate the exported Ecore model instances in Eclipse.

1.2 Motivation

Ecore is used in the industry and by many researchers to describe models. It has been proven that humans have difficulties finding errors in models, unless they can examine instances of them and compare them to their own mental model. Because instance generation is a difficult problem (it is undecidable), it is necessary to limit the scope of possible instances, to be able to automatically generate instances. Alloy provides this important functionality. Our work aims to make this capability accessible to the many existing Ecore models by integrating an import/export function for Ecore models in Lightning.

1.3 Existing Work

While there exists some work to formalize UML using Alloy and other languages, of which UML2Alloy is the most relevant for this thesis, no such work exists in the area of model transformation from Ecore to Alloy or back.

1.3.1 Licenses

We did an analysis of the licenses of Alloy and UML2Alloy and their dependent packages (see Appendix B). It was noted that all licenses are more or less open. All licenses permit modification and disclaim liability. Some permit commercial use, while the others only permit limited commercial use.

Therefore the usage of these programs is non-problematic for non-commercial purposes.

The license for the code developed during the course of this thesis is the same one as the one for Lightning [1, sec. 7.1] (see Appendix A).

1.4 Contribution

This thesis contributes to the field of model driven software development. We extend Lightning's functionality to import Ecore models and export Ecore instances. We also create a standalone tool to perform batch transformations of a large number of models and to test the effectiveness of the transformations.

1.5 Structure

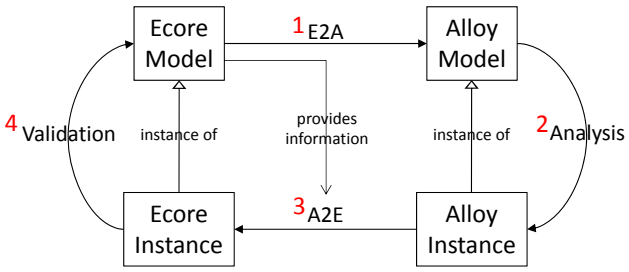

FIGURE 1.1: The model transformation/generation/transformation/validation approach

The verification of Ecore models using Lightning works as shown in figure 1.1 similar to the approach in "From UML to Alloy and Back Again" [2]. The verification of Ecore models using Lightning is composed as follows:

1. Import of Ecore models into Lightning →Ecore to Alloy (E2A) transformation
2. Generating Alloy instances using Lightning
3. Export of Alloy instance from Lightning →Alloy to Ecore (A2E) transformation
4. Verification of Ecore instances by validating them in Eclipse

Towards that goal the following part of the thesis is structured as follows:
In *Chapter 2* we present some technological foundations for this thesis. Then *Chapter 3 & 4* are dedicated to the model transformations E2A & A2E respectively. *Chapter 5* describes the integration of these model transformations into Lightning. In *Chapter 6* the effectiveness of the transformations is evaluated. Finally in *Chapter 7* we give a summary of our work and discuss limitations and possibilities for future work.

Chapter 2

Technological Foundations

2.1 Ecore

“Ecore is the core (meta-)model at the heart of EMF (Eclipse Modeling Framework). It allows expressing other models by leveraging its constructs. Ecore is also its own metamodel (ie: Ecore is defined in terms of itself).

According to Ed Merks, EMF project lead, "Ecore is the defacto reference implementation of OMG’s EMOF" (Essential Meta-Object Facility). Still according to Merks, EMOF was actually defined by OMG as a simplified version of the more comprehensive ’C’MOF by drawing on the experience of the successful simplification of Ecore’s original implementation.” [3]

2.1.1 Usage of Ecore

“Using Ecore as a foundational meta-model allows a modeler to take advantage of the entire EMF ecosystem and tooling - in as much as it’s then reasonably easy to map application-level models back to Ecore. This isn’t to say that it’s best practice for applications to directly leverage Ecore as their metamodel; rather they might consider defining their own metamodels based on Ecore.” [3]

2.1.2 XMI

"The XML Metadata Interchange (XMI) is an Object Management Group (OMG) standard for exchanging metadata information via Extensible Markup Language (XML).

It can be used for any metadata whose metamodel can be expressed in Meta-Object Facility (MOF)." [4]

Ecore models and any model instances are serialized as XMI.

"Several versions of XMI have been created: 1.0, 1.1, 1.2, 2.0, 2.1, 2.1.1, 2.4 and 2.4.1. The 2.x versions are radically different from the 1.x series. The version 2.4.1 was issued in August 2011." [4]

2.1.3 XPath

"XPath, the XML Path Language, is a query language for selecting nodes from an XML document. In addition, XPath may be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML document. XPath was defined by the World Wide Web Consortium (W3C).

The XPath language is based on a tree representation of the XML document, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria." [5]

2.2 Alloy

Daniel Jackson the developer of Alloy described the approach to software design enabled by Alloy as "agile modeling" [6] or a "lightweight formal method" [7].

"In computer science and software engineering, Alloy is a declarative specification language for expressing complex structural constraints and behavior in a software system. Alloy provides a simple structural modeling tool based on first-order logic. The mathematical underpinnings of the language were heavily influenced by the Z notation, although the syntax of Alloy owes more to languages such as Object Constraint Language. Alloy is targeted at the creation of micro-models that can then be automatically checked for correctness. Alloy specifications can be checked using the Alloy Analyzer.

The first version of the Alloy language appeared in 1997. It was a rather limited object modeling language. Succeeding iterations of the language "added quantifiers, higher arity relations, polymorphism, subtyping, and signatures". Although Alloy is designed with automatic analysis in mind, Alloy differs from many specification languages designed for model-checking in that it permits the definition of infinite models. The Alloy Analyzer is designed to perform finite scope checks even on infinite models." [8]

2.2.1 Model structure

Alloy models are relational in nature, and are composed of several different kinds of statements: [9]

- Signatures define the vocabulary of a model by creating new sets

  sig Object{} defines a signature *Object*

  sig List{ head : **lone** Node } defines a signature *List* that contains a field *head* of type *Node* and multiplicity *lone* - this establishes the existence of a relation between *Lists* and *Nodes* such that every *List* is associated with no more than one head *Node*

- Facts are constraints that are assumed to always hold
- Predicates are parameterized constraints, and can be used to represent operations
- Functions are expressions that return results
- Assertions are assumptions about the model that can be checked using the Alloy Analyzer

Because Alloy is a declarative language the meaning of a model is unaffected by the order of statements. [8]

Alloy identifiers may be alphanumerical including underscores, may not start with a number and may not be a keyword contained in table 2.1 [10].

2.2.2 The Alloy Analyzer

"The Alloy Analyzer is a software tool which can be used to analyze specifications written in the Alloy specification language. The Analyzer can generate instances of model invariants,

abstract	all	and	as	assert
but	check	disj	else	exactly
extends	fact	for	fun	iden
iff	implies	in	int	let
lone	module	no	none	not
one	open	or	pred	run
set	sig	some	sum	univ

TABLE 2.1: Alloy Keywords

simulate the execution of operations defined as part of the model, and check user-specified properties of a model. The Alloy Analyzer supports the analysis of partial models. As a result, it can perform incremental analysis of models as they are constructed, and provide immediate feedback to users." [11]

2.3 The Unified Modeling Language (UML)

"Unified Modeling Language (UML) is a standardized (ISO/IEC 19501:2005), general-purpose modeling language in the field of software engineering. The Unified Modeling Language includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.

The Unified Modeling Language was developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software in the 1990s. It was adopted by the Object Management Group (OMG) in 1997, and has been managed by this organization ever since. In 2000 the Unified Modeling Language was accepted by the International Organization for Standardization (ISO) as industry standard for modeling software-intensive systems. The current version of the UML is 2.4.1 published by the OMG in August 2011.

[...]

UML models can be exchanged among UML tools by using the XML Metadata Interchange (XMI) interchange format." [12]

2.3.1 Use of UML in the context of this thesis

In the context of this thesis we use the package UML2Alloy which implements a model transformation from UML to Alloy. UML2Alloy uses a simplified schema of UML, that is

just the UML class diagram with some additional restrictions.

To be able to easier transform to the internal object representation of UML of UML2Alloy we first transform the Ecore model into Eclipse UML.

2.3.2 The Object Constraint Language (OCL)

"The Object Constraint Language (OCL) is a declarative language for describing rules that apply to Unified Modeling Language (UML) models developed at IBM and now part of the UML standard. Initially, OCL was only a formal specification language extension to UML. OCL may now be used with any Meta-Object Facility (MOF) Object Management Group (OMG) meta-model, including UML [and Ecore]." [13]

2.4 Model transformations

"Model transformations can be thought of as programs that take models as input. There is a wide variety of kinds of model transformation and uses of them, which differ in their inputs and outputs and also in the way they are expressed. A model transformation usually specifies which models are acceptable as input, and if appropriate what models it may produce as output, by specifying the metamodel to which a model must conform." [14]

2.5 Design patterns

This thesis supposes that the reader knows what design patterns are and is familiar with the most common patterns. Nonetheless here is a short summary of the pattern used and its meaning.

2.5.1 Visitor pattern

Design Patterns [15] describes the Visitor pattern as follows:

Visitor represents an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

A visitor gathers related operations and separates unrelated ones in distinct visitor classes.

Visitor can traverse an object structure. Visitor can visit the objects in a structure as it traverses them by calling their operations. On the contrary to Iterator the classes on which a visitor operates do not need to have a common parent class.

Visitors can accumulate state as they visit each element in the object structure.

A downside to Visitor is that it assumes that the interface of the elements it operates on is powerful enough to let visitors do their job. As such it can lead to more public operations for those elements that access an element's internal state, which may compromise its encapsulation.

Chapter 3

Transforming Ecore models into Alloy models

3.1 Bridging the model gap

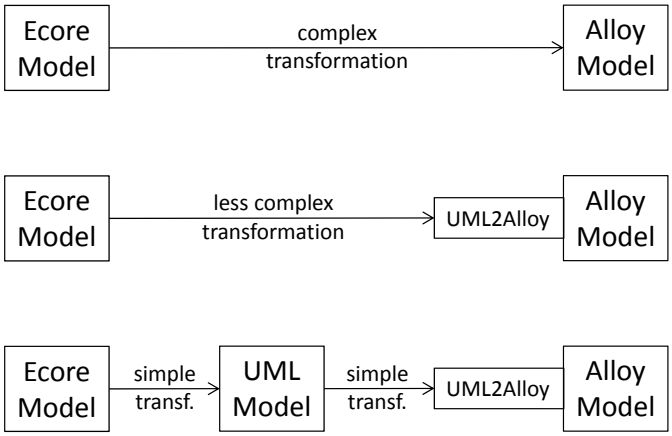

FIGURE 3.1: Reducing the complexity of the transformation by using preexisting solutions and multiple simpler transformations

By using two simple model transformations, one from Ecore to UML and one from UML to UML2Alloy objects, and using preexisting solutions, i.e. UML2Alloy, (as shown in fig. 3.1), we reduce the complexity of the overall transformation, making it easier to program and less error-prone.

3.2 Transforming Ecore models into UML models

As this was our starting point we first tried out different methods to convert Ecore models into UML: Henshin, QVTo and UMLUtil. Each method had problems. UMLUtil had the fewest drawbacks and for that reason was our final choice.

3.2.1 Using Henshin

Henshin is a model transformation language. The Henshin interpreter is invoked programmatically to apply a model transformation defined in Henshin to a model.

3.2.1.1 Integrating EMF & Henshin

To use EMF in a standalone application [16] its following packages need to be included as libraries:

- org.eclipse.emf.common
- org.eclipse.emf.ecore
- org.eclipse.emf.ecore.change
- org.eclipse.emf.ecore.xmi

As indicated on the Henshin webpage [17] for Henshin it is the following packages:

- org.eclipse.emf.henshin.model
- org.eclipse.emf.henshin.interpreter

3.2.1.2 Using Henshin

The Henshin example package contains an Ecore to UML transformation [18] that can be used as shown in 3.1.

```java
public static void runEcore2UML(String path, String ecore, boolean saveResult) {
  // Create a resource set with a base directory:
  HenshinResourceSet resourceSet = new HenshinResourceSet(path);
  if (!resourceSet.initPackageImplementation("org.eclipse.uml2.uml.UMLPackage")) {
    throw new RuntimeException("UML2 package not found. Make sure that org.eclipse.uml2.uml is in the classpath.");
  }
  resourceSet.registerXMIResourceFactories("uml");
  // Load the module:
  Module module = resourceSet.getModule("ecore2uml.henshin", false);
  // Load the example model into an EGraph:
  EGraph graph = new EGraphImpl(resourceSet.getResource(ecore));
  graph.addAll(EcorePackage.eINSTANCE.getEClassifiers()); // we need the EDataTypes in the graph
  // Create an engine and a unit application:
  Engine engine = new EngineImpl();
  UnitApplication app = new UnitApplicationImpl(engine, graph, module.getUnit("main"), null);
  // Execute the transformation:
  if (!app.execute(new LoggingApplicationMonitor())) {
    throw new RuntimeException("Error executing transformation");
  }
  // Saving the result:
  if (saveResult) {
    // Collect all generated UML packages:
    List<EObject> umlPackages = new ArrayList<EObject>();
    for (EObject obj : graph.getRoots()) {
      if (obj.eClass().getName().equals("Package")) { // Ecore packages are instances of "EPackage"
        umlPackages.add(obj);
      }
    }
    Resource resource = resourceSet.createResource(URI.createFileURI(ecore.replaceAll(".ecore", ".uml"))); // TODO: use better extension replacement
    resource.getContents().addAll(umlPackages);
    try {
      resource.save(null);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

Listing 3.1: Using a Henshin transformation on a model

3.2.1.3 Problems with Henshin

The following problems were encountered:

- This Henshin transformation completely ignores the multiplicity.
- Annotations and thus constraints are not supported.

3.2.2 Using QVTo

QVT (Query/View/Transformation) is set of model transformation languages defined by the Object Management Group. QVT-Operational is an imperative language designed for writing unidirectional transformations. [19]

3.2.2.1 Integrating QVTo

To use QVTo the following packages need to be included as libraries [20]

- org.eclipse.ocl
- org.eclipse.ocl.ecore
- org.eclipse.m2m.qvt.oml
- org.eclipse.m2m.qvt.oml.common
- org.eclipse.m2m.qvt.oml.cst.parser
- org.eclipse.m2m.qvt.oml.emf.util
- org.eclipse.m2m.qvt.oml.ecore.imperativeocl

3.2.2.2 Using QVTo

In the QVTo examples [21] it is shown how to transform Ecore to UML using QVTo like in listing 3.2.

```
1  public static void convertEcore2UML(Resource ecoreResource, Resource umlResource) {
2    File f = new File("src/Ecore2UML.qvto");
3    URI u = URI.createFileURI(f.getAbsolutePath());
4    //System.out.println(u.toString());
5    TransformationExecutor executor = new TransformationExecutor(u);
6
7    ModelExtent input = new BasicModelExtent(ecoreResource.getContents());
8    ModelExtent output = new BasicModelExtent();
9
10   ExecutionContextImpl context = new ExecutionContextImpl();
11   ExecutionDiagnostic diagnostic = executor.execute(context, input, output);
12
13   if(diagnostic.getSeverity() == Diagnostic.OK) {
14     umlResource.getContents().addAll(output.getContents());
15     System.out.println("ecore2uml success");
16   } else {
17     IStatus status = BasicDiagnostic.toIStatus(diagnostic);
18     System.out.println(status.toString().replaceAll("], ",  "],\n"));
19     //InvokeActivator.getDefault().getLog().log(status);
20   }
21 }
```

Listing 3.2: Using a QVTo transformation on a model

3.2.2.3 Problems with QVTo

The following problems were encountered:

- The unlimited upper value multiplicity of references is expressed as an object of type LiteralUnlimitedNatural and of value 1. This presents a problem because UML2Alloy does not distinguish the type and only takes the value. For unlimited values UML2Alloy expects the value -1.
- Annotations and thus constraints are not supported.

3.2.3 Using UMLUtil

UMLUtil is a part of EMF and can be used to transform Ecore into UML.

3.2.3.1 Integrating UMLUtil

To use UMLUtil the following packages need to be included as libraries

- org.eclipse.uml.uml2
- org.eclipse.uml.common
- org.eclipse.uml.types

3.2.3.2 Using UMLUtil

As shown in listing 3.3 the result is added to an UML Resource.

```
1 import org.eclipse.uml2.uml.util.UMLUtil;
2 // ...
3 uml.getContents().addAll(UMLUtil.convertFromEcore((EPackage) ecore.getContents().
      get(0), options_convert));
```

Listing 3.3: Using UMLUtil.convertFromEcore

3.2.3.3 Problems with UMLUtil

The following problems were encountered:

- Annotations and thus constraints are not transformed by default.
- Generated UML uses XPath to reference objects.
- The type attribute is not in the xmi namespace.

3.2.3.4 Using IDs for references instead of XPath

As some transformations, like mdeServices' XMI converter, need all object elements to be uniquely identified and referenced by ID it is important to change EMF's default behavior to use unique IDs instead of XPath.

Listing 3.4 shows how it is done [22, p. 491] (changed as to use uml extension).

```
ResourceSet rs = new ResourceSetImpl();
rs.getResourceFactoryRegistry().getExtensionToFactoryMap().put(
  "uml",
  new ResourceFactoryImpl()
  {
    public Resource createResource(URI uri)
    {
      return new XMIResourceImpl(uri)
      {
        protected boolean useUUIDs() { return true; }
      };
    }
  });
```

Listing 3.4: Using UUIDs

3.2.3.5 Using xmi types

mdeServices' XMI converter also requires that the type attribute to be in the xmi namespace.

Listing 3.5 shows how it is done [22, p. 491] and [23].

```
HashMap options_save = new HashMap();
options_save.put(XMIResourceImpl.OPTION_USE_XMI_TYPE, Boolean.TRUE);
uml.save(options_save);
```

Listing 3.5: Saving UML Resource with types in XMI namespace

3.2.3.6 Handling of Annotations

Because constraints are encoded as annotations and UMLUtil ignores annotations by default and even if configured to include them ignores annotations in the EMF namespace, the following configurations have to be applied:

- Configure the options to include annotations (see listing 3.6)
- Change the namespace of every eAnnotation away from EMF (see listing 3.7)
- After the conversion change the namespace of every eAnnotation back to EMF

```
1 HashMap options_convert = new HashMap();
2 options_convert.put(UMLUtil.Ecore2UMLConverter.OPTION__ECORE_TAGGED_VALUES, UMLUtil
      .OPTION__PROCESS);
3 options_convert.put(UMLUtil.Ecore2UMLConverter.OPTION__BODY_ANNOTATIONS, UMLUtil.
      OPTION__PROCESS);
4 options_convert.put(UMLUtil.Ecore2UMLConverter.OPTION__ANNOTATION_DETAILS, UMLUtil.
      OPTION__PROCESS);
```

LISTING 3.6: Setting the options to include Annotations in the E2U conversion

```
1  private static final String ECORE_REPLACEMENT_NS = "http://www.uni.lu/eclipse/org/
       emf/2002/Ecore";
2  private static final String ECORE_NS = "http://www.eclipse.org/emf/2002/Ecore";
3  // ...
4  for(TreeIterator<EObject> iter = EcoreUtil.getAllProperContents(pack, true); iter.
       hasNext(); ) {
5    EObject element = iter.next();
6    if((element instanceof org.eclipse.emf.ecore.EAnnotation)) {
7      EAnnotation a = (EAnnotation) element;
8      if(a.getSource().equals(ECORE_NS))
9        a.setSource(ECORE_REPLACEMENT_NS);
10   }
11 }
```

LISTING 3.7: Changing Namespace of eAnnotations

3.2.4 Decision for UMLUtil

All three methods had problems with annotations. Where Henshin and QVTo had problems with the multiplicity of relations UMLUtil had problems with the form of the generated references and type attributes.

Comparing the problems of Henshin, QVTo and UMLUtil we found that the problems of UMLUtil were the least severe and the easiest to resolve. Therefore we decided to use UMLUtil.

3.2.5 Handling of Oclxmi

Some models have their constraints persisted in a separate file.

Christian Glodt wrote a program called oclxmi2ecore [24] to transform Ecore models with separate OCLXMI constraint files into Ecore models with the constraints included as Annotations.

3.2.5.1 Usage of OCLXMI

The API of oclxmi2ecore (listing 3.8) is as simple as it gets.

In the transformation chain we test first if its application is needed and if the user desires it to be included. Then we create a temporary file to receive the output of oclxmi2ecore and use this file for the rest of the transformation chain. At the end we delete the temporary file. This is shown in listing 3.9.

```
1 public class OCLXMIToEcoreConverter {
2   public static void convertOCLXMIToEcoreWithAnnotations(File oclxmiFile, File
      destEcoreFile) throws IOException;
3 }
```

Listing 3.8: API of oclxmi2ecore

```
1  File oclxmi = new File(path + File.separator + Utilities.renameFileExtension(from,
       "oclxmi"));
2  File file_ecore = null;
3  if(oclxmi.exists() && options.useOCLXMI2Ecore==UseOCLXMI2Ecore.YES) {
4    System.out.println("oclxmi conversion");
5    from = from + "_wconstrainst.ecore";
6    stats.setUsed_oclxmi2ecore(true);
7    file_ecore = new File(path + File.separator + from);
8    OCLXMIToEcoreConverter.convertOCLXMIToEcoreWithAnnotations(oclxmi, file_ecore);
9  }
10 // ...
11 if(file_ecore!=null)
12   file_ecore.delete();
```

Listing 3.9: Using oclxmi2ecore

N.B. In the case of an Exception the temporary file will persist as the file may help in finding the error.

3.3 Transforming the UML

Unfortunately the XMI produced by saving the Resource containing the UML from UMLUtil is incompatible with the XMI parser from UML2Alloy. This is the case because EMF saves in the XMI version 2.0 and UML2Alloy expects XMI version 1.3 with Novosoft UML.

To resolve this there are two possibilities. Either to convert the Eclipse XMI into Novosoft XMI or write a mapping from Eclipse UML objects to UML2Alloy UML objects.

3.3.1 Using the XMI converter

The XMI converter from mdeServices [25] can import and export different XMI formats.

Listings 3.10 and 3.11 show its relevant API and usage respectively.

```
1 public class XMIGenerator extends MDEServices {
2   public void generate(File input, String importFromTool, File output, String
      exportForTool, String user) throws TransformationNotApplicable,JDOMException,
      IOException, ImportXMIException, ExportXMIException;
3 }
```

Listing 3.10: API of XMI converter

```
1 import mdeServices.XMIGenerator;
2 // ...
3 XMIGenerator xg = new XMIGenerator();
4 xg.generate(new File(from), "EclipseUML2", new File(to), "Novosoft", "TestUser");
```

Listing 3.11: Using the XMI converter

3.3.1.1 Limitations

While XMI converter can read EMF XMI (XMI v.2) and can write the version which UML2Alloy reads (XMI v.1.3), XMI converter cannot write the exact dialect which is Novosoft UML. To enable a successful conversion we would have to write an exporter for XMI converter for Novosoft UML.

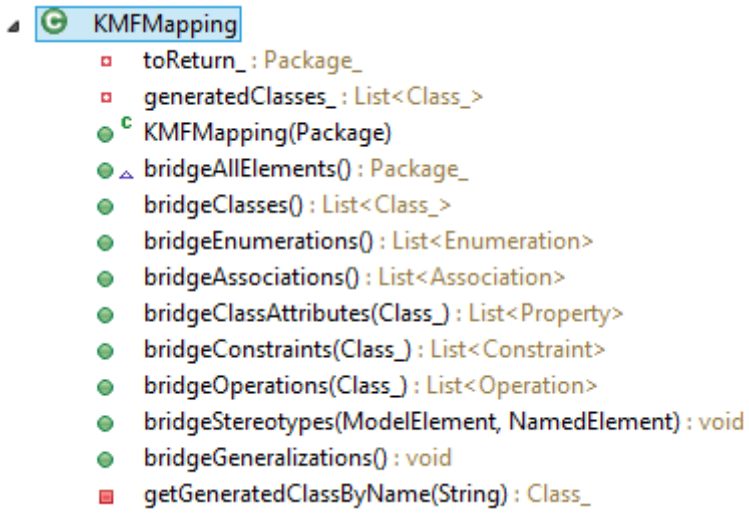

FIGURE 3.2: Structure of KMFMapping

3.3.1.2 Conclusion

We decided that it would be preferable to pursue the alternative option, which is EMFMapping 3.3.2, because of higher chances for success. Reasons for this are the lack of knowledge on the concrete format of XMI that UML2Alloy needs as it seems to be a proprietary one and the uncertainty if the UML2Alloy XMI parser would then process the resulting XMI file correctly.

Those issues are sidestepped by directly transforming from the format EMF uses to represent UML objects to the internal UML representation of UML2Alloy. Additionally there is more documentation on the EMF representation of UML objects then for the mdeServices' XMI converter or the Novosoft UML file format.

3.3.2 Using the EMFMapping

The basic structure for EMFMapping (fig. 3.2) comes from the UML2Alloy class KMFMapping which transforms from Kmfstudio UML to the internal UML2Alloy UML representation.

The supporting class EMFUtil is also based on a class from UML2Alloy, i.e. KMFUtil.

3.3.2.1 Adapting EMFMapping

To adapt those classes for their intended use we first changed the types from those in uk.ac.ukc.cs.kmf.kmfstudio.uml to org.eclipse.uml2.uml, then we ensured that other differences are considered.

These differences are the way Eclipse and Kmfstudio contain the objects. Eclipse saves them in a tree structure, whereas Kmfstudio saves them in a set. This means that the functions that iterate over all elements in the package, i.e. getEMFClasses, getEMFEnums and getAssociations in EMFUtil, need to be adapted like shown in listing 3.12.

```
1 for(TreeIterator<EObject> iter = EcoreUtil.getAllProperContents(p_, true); iter.
      hasNext(); ) {
2   EObject element = iter.next();
3   // ...
4 }
```

Listing 3.12: Iterating over all package (p_) elements

Both have different methods to access owned attributes, operations and parameters. A comparison is shown in table 3.1.

uk.ac.ukc.cs.kmf.kmfstudio.uml	**org.eclipse.uml2.uml**
Class_.getFeature() if(f instanceof Attribute)	Classifier.getOwnedAttributes()
Class_.getFeature() if(f instanceof Operation)	Classifier.getOwnedOperations()
Operation.getParameter()	Operation.getOwnedParameters()

Table 3.1: Comparison KMF / EMF for acessing attributes, operations & parameters

3.3.2.2 Handling of Identifiers

The first additional step in the EMFMapping transformation is to make all names acceptable Alloy identifiers (see p. 6 and 26) by replacing unacceptable characters by an underscore. Keywords are prefixed by an underscore. UML2Alloy does not perform this step.

N.B. There is no check for name collision, i.e. that a name occurs twice or more often in a signature or its super-types.

3.3.2.3 Handling of Constraints

In Ecore constraints are saved as eAnnotations. To bridge them into Alloy we have to iterate over all classes and their properties and for every class and property do the following: check if their is an annotation in the OCL namespace (NS) and check if their is one in the Ecore NS. For the Ecore NS it is possible to contain a key-value pair with the key "constraint" and the value the constraint body text. Alternatively if both OCL and Ecore annotations exist and their is a key "constraints" in the Ecore NS, then the corresponding value is a space-delimited array where every item should correspond to a key in the OCL NS, the corresponding value being the body of the constraint.

This algorithm is illustrated in listing 3.13.

It is interesting to note, that while it is possible to generate facts with the same name with this algorithm, the Alloy Analyzer will not issue any warning or error, unlike with signature attributes of the same name and scope.

```
1  public List<Constraint> bridgeConstraints(org.eclipse.uml2.uml.NamedElement el) {
2      List<Constraint> toReturn = new ArrayList<Constraint>();
3      List<EAnnotation> lst = el.getEAnnotations();
4      EAnnotation ecore = null;
5      EAnnotation ocl = null;
6      for(EAnnotation an : lst) {
7        if(an.getSource().equals(OCL_URI))
8          ocl = an;
9        else if(an.getSource().equals(ECORE_URI)) {
10         ecore = an;
11         String conString = an.getDetails().get("constraint");
12         if(conString!=null) {
13           org.uml2alloy.umlmmsimplifiedprofile.Constraint umlcon = new
     ConstraintImpl(el.getName(), conString);
14           toReturn.add(umlcon);
15           increment_constraints();
16         }
17       } else if(an.getDetails().containsKey("constraint") || an.getDetails().
     containsKey("constraints"))
18         this.stats.appendWarning("constraints out of ecore_uri");
19     }
20     if(ecore!=null && ocl!=null) {
21       EMap<String, String> edetails = ecore.getDetails();
22       EMap<String, String> odetails = ocl.getDetails();
23       String constraints = edetails.get("constraints");
24       if(constraints!=null) {
25         String[] keys = constraints.split(" ");
26         for(String key : keys) {
27           String constraint_body = odetails.get(key);
28           if(constraint_body!=null) {
29             org.uml2alloy.umlmmsimplifiedprofile.Constraint umlcon = new
     ConstraintImpl(el.getName()+"_"+key, constraint_body);
30             toReturn.add(umlcon);
31             increment_constraints();
32           }
33         }
34       }
35     }
36     return toReturn;
37   }
```

Listing 3.13: Mapping constraints in eAnnotations to UML2Alloy constraints

3.3.2.4 Handling of Aggregations

UMLUtil maps Ecore References to UML associations and containment to composition.

Because UML2Alloy does not support aggregations and compositions [26, p. 5], compositions have to be handled as normal associations while introducing additional constraints.

For this we use all the constraints as defined by Gogolla and Richters [27, p. 100f] shown in table 3.2 except for the existential dependency constraint, because existential dependency of the part is irrelevant in the static environment of Ecore and Alloy.

UML feature	Requirement in comparison to association
Association	-
Aggregation	Forbidding instance reflexivity
Composition	Forbidding instance reflexivity
	Existential dependency for the part
	Strong form of forbidding sharing

TABLE 3.2: Table with constraints for aggregations

For the forbidding reflexivity constraint we test if the constraint is necessary by checking if the reflexive transitive closure of all super types of the class A of an association end called *ref* contains the end type. If this is the case we add the constraint that for every element of class A that element may not be included in the transitive closure of *ref*.

N.B. This only forbids direct reflexivity. It is still possible for an element to contain itself across multiple containment relations.

Also for every association end encountered during mapping of associations that is a composite it is added to a list in a map with as key the class of the end type. This is so it is possible to iterate over all the composition relations and add the relevant constraint (see listing 3.14).

Now, because the Alloy Analyzer casts a warning if a is disjoint to b in $\forall p : P \forall a : A \forall b : B | (p \in a.rel1 \wedge p \in b.rel2) \Rightarrow a = b$ we choose to use the following form for the strong form of forbidding sharing: $\forall p : P \forall a : A \forall b : B | \neg(p \in a.rel1 \wedge p \in b.rel2)$. This could be a problem if it should be possible for p to be contained in two different relations of the same object a, because the second form only allows for containment in one relation even if a=b.

```
1  private Map<Class_, List<Relation>> compositions;
2  // ...
3  Set<Class_> allcls = compositions.keySet();
4  for(Class_ allcl : allcls) {
5    List<Relation> lst = compositions.get(allcl);
6    for(int c_a=0; c_a<lst.size(); c_a++) {
7      Relation a = lst.get(c_a);
8      for(int c_b=c_a; c_b<lst.size(); c_b++) {
9        Relation b = lst.get(c_b);
10       if(a.equals(b))
11         // No weak form of forbidding sharing
12       else
13         // No strong form of forbidding sharing
14     }
15   }
16 }
```

Listing 3.14: Iterating over composition relations

3.3.2.5 Implementation of Constraints

There are two ways to create constraints, both illustrated in listing 3.15 and 3.16 representing the forbidding reflexivity constraint.

1. The constraints are implemented by creating an *org.uml2alloy.umlsimplifiedprofile .ConstraintImpl* object with name and constraint string as arguments. This object is then added as a constraint to its owning classes (see listing 3.15). The downside to this method is that the constraint has to be parsed by UML2Alloy which during testing did not prove very reliable. This is the method used for user constraints.

2. The other way is to directly create Alloy objects (see listing 3.16). The advantage of this method is that it always works. The disadvantage is the increased development time to assemble a constraint. It is because of the advantage that we choose this method for all additional constraints and rewrote any that were not already in this form.

```
1  c.addConstraints(new ConstraintImpl(assoc_name+"_isNotReflexive", "not((self->
       closure("+assoc_name+"))->includes(self))"));
```

Listing 3.15: Creating constraints as ConstraintImpl objects

```
1  Fact f = FactFactory.createEmptyFact(c.getName()+"_"+assoc_name+"_isNotReflexive");
2  Formula i = FormulaFactory.createQuantifiedFormula(Quantifier.ALL,
3      DeclarationFactory.createDecls("self", c.getName()),
4      FormulaFactory.createNegationFormula(
5        FormulaFactory.createInElemFormular(ExprFactory.createVarExpr("self"),
6          new BinaryExpr(ExprFactory.createVarExpr("self"), BinaryExprOp.JOIN_DOT,
7            new UnaryExpr(UnaryExprOp.TRANS_CLOSURE,
8              new BinaryExpr(ExprFactory.createSigExpression(c.getName()),
9                BinaryExprOp.DOMAINRESTRICT,
10               ExprFactory.createVarExpr(assoc_name)
11   ))))));
12 f.setFormula(FormulaFactory.createFormulaSeq( i ));
13 Converter.static_addFact(f);
```

Listing 3.16: Creating constraints directly as Alloy objects

3.3.2.6 Additional Alloy keywords

During testing we found out that the newest Alloy version (v4.2) has additional keywords to the ones specified in the Alloy reference [10], see table 2.1.

Spread across the new features page [28] are the following keywords listed:
exh part private seq

In the release notes [29] is written that enum has been added as a keyword.

Nowhere listed but still unsuitable as an identifier name is the type name String. Variables of type String make it so that the Alloy Analyzer can find no instance of the model.

A summary of all additional keywords is the table 3.3.

enum	exh	part
private	seq	string

Table 3.3: Additional Alloy Keywords

3.4 Using UML2Alloy

3.4.1 API of UML2Alloy

The transformation of UML2Alloy can be invoked with a call to org.uml2alloy.UML2AlloyTransformer.convert(Package_ p). The package p and all its

content is from the Java package org.uml2alloy.umlmmsimplifiedprofile.

Other than an example Java file [30] and the diagram of the simplified UML model [26, p. 22] UML2Alloy has no API documentation. Any additional information had to come from decompiling the UML2Alloy jar file and inspecting that code.

3.4.1.1 Usage of API

In the example Java file [30] it is shown that UML2Alloy is used as shown in listing 3.17.

```
1  import org.uml2alloy.AbstractUML2AlloyTransformer;
2  import org.uml2alloy.UML2AlloyTransformer;
3  import org.uml2alloy.alloy4.AlloyModel;
4  import org.uml2alloy.umlmmsimplifiedprofile.Package_;
5  // ...
6  Package_ p = the_target_package;
7  AbstractUML2AlloyTransformer u2a = new UML2AlloyTransformer();
8  try {
9    AlloyModel am = u2a.convert(p);
10   System.out.println(am.getAlloyModelAsString());
11   // And save the generated model to a file
12   am.writeToFile("C:\\testAlloyModel.als");
13 } catch (...) {
14   e.printStackTrace();
15 } catch (Exception e) {
16   e.printStackTrace();
17 }
```

Listing 3.17: Using UML2Alloy

3.4.2 Extending UML2Alloy

3.4.2.1 Improving formatting of the resulting Alloy file

Because the standard Alloy output is not very pretty we extended the UML2Alloy class *PPVisitor4Trace* which handles generating the string output from Alloy objects. To do this the class makes use of the Visitor pattern (see 2.5.1). The changes are these:

- No space in front of signatures unless it is to separate the multiplicity from "sig".
- Added a space in front of the opening curly brackets of signatures.

- The declarations in signatures are now tab-indented with the closing curly bracket on a new line.
- Added a newline after the module definition.
- Reduced the space between predicates.

Listing 3.18 shows how to save Alloy files like this and figure 3.3 gives a side-by-side comparison of the two formattings.

```
1 am.writeToFile(alloy, new PrettyAlloyPPVisitor());
```

Listing 3.18: Saving the Alloy file with pretty formatting

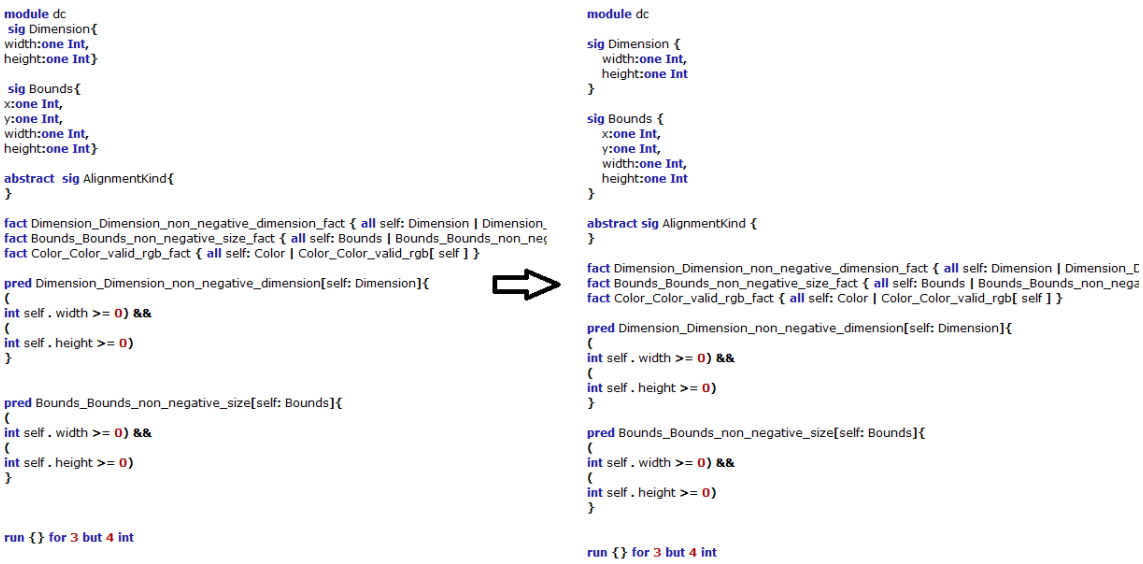

Figure 3.3: Comparison UML2Alloy & new formatting

3.4.2.2 Improving OCL transformation rules

To permit transitive closure for the forbidding reflexivity constraint of containment relations (see 3.3.2.4) following the first method of transforming constraints (see 3.3.2.5) it was necessary to add the capability for transitive closure in the UML2Alloy OCL transformation rules.

For this we decompiled the following UML2Alloy classes:

- UML2AlloyTransformer

- SiTraClass2Signature
- SiTraInvariant2Predicate
- SiTraSubClass2ExtendsSignature
- SiTraArrowSelectionExp2Expr
- SiTraOperationCall2OperationCall

After that we changed SiTraArrowSelectionExp2Expr so as to accept closure as a valid function, SiTraOperationCall2OperationCall as to support the closure function (see 3.19). In the others we changed any reference to these classes so as to refer to our own version. This change of reference is expressed twice in listing 3.20. The first change (line 8) affects the rule registry. The second change (line 16) is any reference to the class by name.

```
1 if (arrowExpr.equals("closure")) {
2   Expr leftExpr = (Expr) nodeLeft;
3   Node nodeRight = handleArguements(source, t);
4   Expr rightExpr = (Expr) nodeRight;
5
6   if(get_class_name()!=null)
7     rightExpr = new BinaryExpr(ExprFactory.createSigExpression(_class_name),
      BinaryExprOp.DOMAINRESTRICT, rightExpr);
8   return new BinaryExpr(leftExpr, BinaryExprOp.JOIN_DOT, new UnaryExpr(UnaryExprOp.
      TRANS_CLOSURE, rightExpr));
9 }
```

Listing 3.19: Code added to SiTraOperationCall2OperationCall

```
1  import org.uml2alloy.transformer.rules.ocl.SiTraArrowSelectionExp2Expr;
2  // ...
3  public class CKUML2AlloyTransformer extends AbstractUML2AlloyTransformer {
4    public CKUML2AlloyTransformer() {
5      // Replacing some transformation rules with our own
6      @SuppressWarnings("rawtypes")
7      List<Class<? extends UML2AlloyRule>> lst = getSiTraTransformer4UML2Alloy().
       getRuleTypes();
8      lst.set(lst.indexOf(SiTraArrowSelectionExp2Expr.class), lu.uni.lightning.util.
       rules.ocl.SiTraArrowSelectionExp2Expr.class);
9      // ...
10   }
11   // ...
12   public void mapClasses(Package_ p) throws RuleNotFoundException,
13       UML2AlloyTransformationException {
14     ArrayList classes = p.getClasses();
15     if (classes != null) {
16       Class c2sclass = lu.uni.lightning.util.rules.SiTraClass2Signature.class;
17       List listsig = getSiTraTransformer4UML2Alloy().transformAll(
18           c2sclass, classes);
19       // ...
20     }
21   }
22 }
```

Listing 3.20: Changing UML2Alloy transformation rules for increased functionality

3.4.3 Alternative solution to rewriting the transformation rules

Alternatively to rewriting the UML2Alloy transformation rules it is possible to directly create Alloy objects in EMFMapping like described in 3.3.2.5. All facts created like this are then added to a list of facts which is added to the resulting Alloy model after UML2Alloy's transformation is run. This is shown in listing 3.21.

This evades any problems with UML2Alloy's parsing of OCL constraints. This method is used for all additional constraints (i.e. forbidding reflexivity, weak/strong form of forbidding sharing and enforce abstract).

```
Fact f;
listofacts.add(f);
// ...
AlloyModel am = u2a.convert(p);
am.addFacts(listofacts);
```

Listing 3.21: Adding facts to the Alloy model after conversion

3.5 Putting the whole conversion together

All the different parts of the conversion from Ecore to Alloy are put together in the *Conversion* class. This class defines the possible options and contains all the code involved in transforming from Ecore to Eclipse UML and from Eclipse UML to Alloy as well as invoking both of these transformations. The API that is exposed is shown in listing 3.22.

```
package lu.uni.lightning.util;
public class Converter {
  public static enum UseOCLXMI2Ecore { YES, NO, TRY }
  public static enum UseConstraints { YES, NO, TRY }
  public static class Config;
  public static boolean runEcore2Alloy(String ecore, Stats stats, Config options);
}
```

Listing 3.22: The API of the Converter class

3.5.1 Configuration options

The possible configuration options with their default value are shown in listing 3.23 with a description in table 3.4.

```
public static class Config {
  public UseOCLXMI2Ecore useOCLXMI2Ecore = UseOCLXMI2Ecore.TRY;
  public UseConstraints useConstraints = UseConstraints.TRY;
  public boolean replace_keyword_names = true;
  public boolean enforce_abstract = true;
}
```

Listing 3.23: Possible configuration options for E2A transformation

Option	Description
useOCLXMI2Ecore	This specifies whether to use OCLXMI2Ecore (see 3.2.5). The option TRY means that if a first transformation with OCLXMI2Ecore fails a second transformation without OCLXMI2Ecore will provide the result.
useConstraints	This specifies whether to use constraints (see 3.3.2.3). The option TRY means that if at first transformations with constraints fail (with or without OCLXMI2Ecore) another transformation without constraints will provide the result.
replace_keyword_names	This specifies whether any identifiers that are keywords are replaced (see 3.3.2.2).
enforce_abstract	This specifies that if any Alloy signatures are abstract they should follow the expected behavior from Ecore and other languages (Java, C, ...) (see 3.6.2).

TABLE 3.4: Description of configuration options for E2A transformation

3.6 Problems encountered

3.6.1 Multiple inheritance

UML2Alloy does not support multiple inheritance [26, p. 4] and as such throws an Exception when it encounters it.

Alloy itself has no simple support for multiple inheritance [9, p. 96].

As it would be beyond the scope of this thesis it is a future exercise to enable the E2A transformation of classes with multiple inheritance.

3.6.2 Abstract signatures

If a signature is declared abstract, we would expect that there could not be any instances of this type. However in Alloy if there are no sub-types the Alloy Analyzer assumes that there is an error in the model and disregards the abstract keyword [9, p. 272f].

To keep the expected behavior an additional constraint needs to be added in that case. This constraint is shown in listing 3.24.

If the “enforce_abstract” option is set to true we check in the bridgeClasses method in EMFMapping if an abstract class has any descendants if not we add the constraint.

```
1 Fact f = FactFactory.createEmptyFact(umlcl.getName()+"_isAbstract");
2 Formula i = IntFactory.createElemIntExpr(IntFactory.createCardinalityExpr(
      ExprFactory.createSigExpression(umlcl.getName())),
3     IntCompOp.INT_EQUALS, new LiteralIntExpr(0)
4   );
5 f.setFormula(FormulaFactory.createFormulaSeq( i ));
```

Listing 3.24: The enforce abstract constraint

3.6.3 Enum literals have conflicting names

UML2Alloy transforms every enumeration literal into a signature with exactly one instance of the same name. Because of this if there are a few enumerations with literals of the same name, this would lead to name collision. To minimize the likelihood of this happening we prepend the enumeration name to the literal’s name (see bridgeEnumerations() in EMFMapping).

Although Alloy has the enum keyword, UML2Alloy does not use it. As the enum keyword would provide a more readable notation, changing UML2Alloy to use it could be a possible future improvement, even if the enum keyword itself does not eliminate the danger of name collision.

3.6.4 Strings and other datatypes

UML2Alloy does not translate datatypes to signatures. Therefore we had to create a signature with the name of the datatype for every datatype listed in table 3.5 that is used. The basis for this list is the Ecore datatypes and some other commonly used ones.

A remaining problem is if the datatype is not in the list it will not be translated. A possible solution would be to transform every used datatype into its own signature regardless if it is an Ecore datatype or a user-defined one.

Bool	Boolean	EBigDecimal	EBigInteger
EBoolean	EBooleanObject	EByte	EByteArray
EByteObject	EChar	ECharacterObject	EDate
EDouble	EDoubleObject	EFloat	EFloatObject
EIntegerObject	EJavaClass	EJavaObject	ELong
ELongObject	EShort	EShortObject	EString
String	_String		

Table 3.5: All importable datatypes

3.6.5 Key collision causes ambiguity

Because at first every property was saved in a Map with a key derived from its type and name, it was possible for more than one class to have a property of the same name and type, which would then reuse the earlier property. While the consequence of this on the declaration of signature attributes was nonexistent, the generated facts related to the relation could have more conditions than needed. This is the case for unidirectional relations where the non-navigable property has been reused and the reused property is navigable. This would lead to the creation of a symmetry fact and make it so that the association fact had an additional term even though no corresponding field existed in the signature. This presence of additional constraints and the absence of corresponding fields would lead the Alloy Analyzer to produce ambiguity errors if more than one other signature have fields of the same name.

The solution to this problem was not to use the Map as the iteration over every association and their member ends ensures that every property is only passed once and therefore there is no need to reuse earlier properties.

3.6.6 Inability to resolve types outside Ecore package

Because EMF is sometimes unable to resolve types outside the model package or the Ecore package, this leads to classes that by default have no type. This leads to NullPointerExceptions in the UML conversion or in the UML2Alloy conversion when the type is accessed for its name.

Possible solutions in the future would be any of the following or a combination thereof:

- Raise an error or warning
- Replace all unresolvable types with a dummy object
- Translate all packages into distinct Alloy modules

3.6.7 Array index out of bounds

EMF needs that all references with an eOpposite have their opposite also contain a symmetric eOpposite. If that is not the case the resulting EMF UML model will have an association with just one association end. This will lead to an array index out of bounds exception during the mapping to UML2Alloy objects.

Chapter 4

Transforming Alloy instances into Ecore instances

This transformation takes the Alloy instance generated by the Alloy Analyzer and making use of EMF's dynamic capabilities loads the Ecore model which generated the Alloy model and generates a corresponding Ecore instance.

4.1 Using the Alloy Analyzer

The API of the Alloy Analyzer that is used is shown in table 4.1. The information is from the Alloy API documentation [31].

4.1.1 Generating the model instance

The instance generation is encapsulated in the class *lu.uni.lightning.core.AlloyMetaModel* part of Lightning and written by Loic Gommaitoni. For testing purposes we copied this class with a more simplified interface shown in listing 4.1. For both versions the original version and the testing one an A4Solution object can be obtained by calling runModel(). This A4Solution object contains all the information about the instance objects created.

CompModule	parseEverything_fromFile(A4Reporter rep, java.util.Map<java.lang.String,java.lang.String> loaded, java.lang.String filename) Read everything from "file" and parse it; if it mentions submodules, open them and parse them too.
A4Solution	TranslateAlloyToKodkod.execute_commandFromBook(A4Reporter rep, java.lang.Iterable<Sig> sigs, Command cmd, A4Options opt) Based on the specified "options", execute one command and return the resulting A4Solution object.
java.lang.Iterable<ExprVar>	A4Solution.getAllAtoms() Returns an unmodifiable copy of the list of all atoms if the problem is solved and is satisfiable; else returns an empty list.
SafeList<Sig>	A4Solution.getAllReachableSigs() Returns an unmodifiable copy of the list of all sigs in this solution's model; always contains UNIV+SIGINT+ SEQIDX+STRING+NONE and has no duplicates.
A4TupleSet	A4Solution.eval(Sig.Field field) Return the A4TupleSet for the given field (if solution not yet solved, or unsatisfiable, or field not found, then return an empty tupleset)
A4Solution	A4Solution.next() If this solution is UNSAT, return itself; else return the next solution (which could be SAT or UNSAT).
SafeList<Sig.Field>	Sig.getFields() Return the list of fields as a combined unmodifiable list (without telling you which fields are declared to be disjoint)
java.lang.String	Sig.label The label for this sig; this name does not need to be unique.
java.lang.String	Sig.Field.label The label associated with this object; it's used for pretty-printing and does not have to be unique.
java.lang.String	A4Tuple.atom(int i) Returns the i-th atom in this Tuple.
int	A4Tuple.arity() Returns the arity.

Table 4.1: API of the Alloy Analyzer used

```
1 package lu.uni.lightning.util;
2 public class A4SolutionGenerator {
3   public A4SolutionGenerator(String source);
4   public A4Solution runModel() throws Exception;
5 }
```

Listing 4.1: Interface of A4SolutionGenerator class

4.2 Reuse of the information in the Ecore model

The Ecore model which generated the Alloy model is loaded and then in an iteration over all elements in the Ecore model the relevant information for the transformation is saved in maps. A distinction is made according to the type of the element. The key for the maps is the name of the Alloy signature that corresponds to that element.

The A2E transformation assumes that the option "replace_keyword_names" (see table 3.4) was used for the E2A transformation. This assumption is safe because if the option was not used and there was an identifier that was a keyword, then a syntax error would have been cast, no instance could have been generated and the A2E transformation could not have been used.

- For **Classes** information is saved in three different maps. The classes map contains the mapping name to class. The attributes map contains for every name of a class a map of all its structural features' names to the corresponding structural feature. The map cls_factories contains for every name of a class the factory with which it can be created.
- For **Enumerations** every literal is saved in the literals map.
- For **DataTypes** it is assured that the datatype has an instance class if they are an Ecore datatype, named like an Ecore datatype without the leading 'E' or the name contained in a table (case-insensitive) (see table 4.2). After that the datatype is saved in the types map with its name as key.

Datatype name	Corresponding Ecore datatype
boolean bool	EBoolean
integer int	EInt
double real	EDouble

TABLE 4.2: Correspondence Datatype name & Ecore Datatype

- Finally an iteration over all **Attributes** inspects their type and makes sure that all datatypes even those contained in other packages are saved in the types map, after making sure they have an instance class like described in the previous paragraph.

4.3 Mapping Alloy instance objects to the corresponding Ecore types

For every object instance created by the Alloy Analyzer called atoms the corresponding classifier is looked up in the maps and then an instance created. For classes the factory necessary to create an instance has also to be looked up. For literals it is enough to return the literal and for data types a distinction is made for String types. String types return the full Alloy instance name. Any other data type returns the default value. The result is then saved in the result Map called "objects" with the Alloy instance name as key.

The iteration over all Alloy atoms, with call to the Ecore object creation function and saving to the result map is shown in listing 4.2. The details of the creation function are shown in listing 4.3.

```
1 for (ExprVar s : solution.getAllAtoms())
2   objects.put(s.toString(), createObject(s.label));
```

LISTING 4.2: Iteration over all Alloy instance objects, creating the corresponding Ecore instance object and saving the result

```
String initial = name;
name = name.split("\\$")[0];
Object toReturn = classes.get(name);
if(toReturn != null)
  return cls_factories.get(name).create(classes.get(name));
toReturn = literals.get(name);
if(toReturn != null)
  return toReturn;
toReturn = types.get(name);
if(toReturn != null) {
  EDataType dt = (EDataType) toReturn;
  if("java.lang.String".equals(dt.getInstanceClassName()))
    return initial;
  return dt.getDefaultValue();
}
Assert.isNotNull(toReturn);
```

LISTING 4.3: Creating Ecore objects based on the Alloy instance name

After the creation of the objects an iteration over all signatures and the fields in those signatures looks up all links between objects in the Alloy instance. For every link the respective source and target objects are looked up in the result map based on their name except if the target object's name does not contain a dollar, then it is a number. Finally the target is assigned to the correct structural feature of the source. All this is shown in listing 4.4.

```
for (Sig s : solution.getAllReachableSigs()) {
  for (Field f : s.getFields()) {
    A4TupleSet ts = (solution.eval(f));
    for (A4Tuple t : ts) {
      source = t.atom(0);
      target = t.atom(t.arity()-1);
      setField(f.label, objects.get(source),
          (target.contains("$") ? objects.get(target) : Integer.parseInt(target)));
    }
  }
}
```

LISTING 4.4: Iteration over all links and assigning the target object to the correct structural feature of the source object

With the name of the class of the source object and the name of the link it is possible to find the structural feature that defines the link. This is done by looking up the attributes

with the class name. Among those attributes it is the one by the name of the link. If their is not one it must be a structural feature inherited from one of the super types. This is shown in listing 4.5.

```
String source_name = rename(source.eClass().getName());
String initial_name = rename(source.eClass().getName());
EStructuralFeature f = null;
while(f==null && source_name!=null) {
  Assert.isNotNull(attributes.get(source_name));
  f = attributes.get(source_name).get(field);
  // Considering inheritance
  if(f==null) {
    List<EClass> lst = classes.get(source_name).getESuperTypes();
    if(lst.isEmpty())
      source_name = null;
    else
      source_name = rename(lst.get(0).getName()); // Only considering single
    inheritance
    if(source_name!=null && source_name.equals(initial_name)) // No recursive
    inheritance
      source_name = null;
  }
}
Assert.isNotNull(f);
```

Listing 4.5: Finding the structural feature

With the structural feature identified a distinction has to be made if more than one object can be assigned to that feature. For multiple values the list has to be first retrieved with eGet and then a new list created if none existed yet and finally the object has to be added to the list and the list set again with eSet. This is shown in listing 4.6.

```
Object obj = source.eGet(f);
if(obj != null && obj instanceof List)
  lst = (List) obj;
else
  lst = new ArrayList();
lst.add(object);
source.eSet(f, lst);
```

Listing 4.6: Adding an object to a many valued structural feature f

With a single valued structural feature it is enough to assign the object to the feature f and for the case that an error happens assign the default value. This is shown in listing 4.7. The error could happen for some DataTypes.

```
try {
  source.eSet(f, object);
} catch(ClassCastException e) {
  e.printStackTrace();
  try {
    source.eSet(f, f.getDefaultValue());
  } catch (Exception e1) {
    throw new Exception(e);
  }
}
```

Listing 4.7: Assign an object to a single valued structural feature f

4.4 Saving the Ecore model instance

To save the Ecore model instance a resource at the required destination is created. It is ensured that the resource is empty and then every object in the result map that is an Ecore object (EObject) is added to the resource. Finally the resource is saved. All this is shown in listing 4.8.

```
Resource result = rs.createResource(URI.createFileURI(xmi.getAbsolutePath()));
result.getContents().clear();
Collection<Object> values = objects.values();
for(Object obj : values)
  if(obj instanceof EObject)
    result.getContents().add((EObject) obj);
Map options = new HashMap();
options.put(XMLResource.OPTION_SCHEMA_LOCATION, Boolean.TRUE);
result.save(options);
```

Listing 4.8: Saving an Ecore instance

4.5 Final API for the A2E conversion

The API for the Alloy instance to Ecore instance conversion is shown in listing 4.9. The A4Solution object contains all the information about the Alloy instance objects. The ecore

string parameter has to be the path to the Ecore model file which generated the Alloy model that generated the A4Solution and the instanceFile string is the path to where the result should be saved.

```
1 package lu.uni.lightning.ui.content;
2 public class SolutionXMIExporter {
3   public static Resource export2XMI(A4Solution solution, String ecore, String
      instanceFile) throws Exception;
4 }
```

Listing 4.9: API of the A2E conversion

4.6 Problems encountered

4.6.1 Missing constraints

Without the constraints related to aggregations (see 3.3.2.4) the Alloy Analyzer generates instances that fail validation or generate runtime errors such as Cyclic dependency errors.

Another error that can be caused by a missing forbidding sharing constraint is if a field has a multiplicity constraint of one ore more and is a containment. If in that case the object assigned to that field is also assigned to another containment EMF automatically removes it from the first field and thus can cause a “Needs value” error.

4.6.2 Datatypes without Java classes

Some datatypes do not have a Java instance class defined. This presents a problem because it is impossible to assign an object of that type to an attribute, thus possibly failing validation.

As a solution we try to guess the needed Java class based on the name of the base type.

As such a datatype with the name String gets the same Java instance class as the EString datatype (i.e. java.lang.String), while the datatype Int gets the Java instance class from EInt (i.e. java.lang.Integer).

More information on how this is done are in section 4.2 on Datatypes.

4.6.3 Missing xsi:schemaLocation

To permit the loading and validation of the model instance from other programs than Lightning or Ecore2Alloy it is important to include the schemaLocation, i.e. where the Ecore model is saved.

The listing 4.10 shows how to activate this, as described in the dynamic EMF tutorial [32].

```
1 Map options = new HashMap();
2 options.put(XMLResource.OPTION_SCHEMA_LOCATION, Boolean.TRUE);
3 resource.save(options);
```

Listing 4.10: Using xsi:schemaLocation

Chapter 5

Integration into Lightning & Standalone UI

5.1 Strategy

We developed and tested the conversion algorithm in a standalone project to develop the API and create a UI to test the API. After that we set a dependency in Lightning on that project and used the API in Lightning.

5.2 Standalone UI and library

The library encapsulating the E2A & A2E functionality is called "Ecore2Alloy", although after a later stage of development it also contains the A2E functionality. This is to limit the number of libraries and because A2E shares many core functionality, like the Ecore ResourceSet specialization and EMFUtil classes, with E2A.

We wrote a standalone UI for the library also included in the project "Ecore2Alloy" in order to better test the library, later on to automate the validation and to do this with a large number of test cases / models. (This is comparable to a batch conversion.) For this standalone UI we reused some UI elements from Lightning (GUI, Config, Tree-Elements).

The functionality we added is the separation into three parts and the functionality of the treeview.

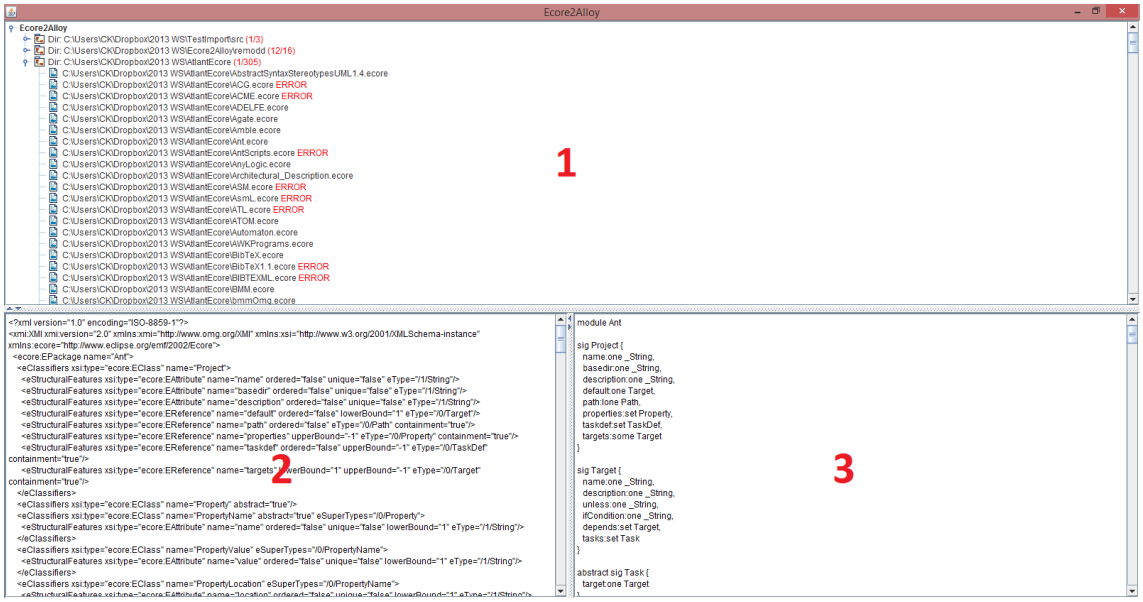

FIGURE 5.1: Screenshot of Ecore2Alloy

5.2.1 How to use the standalone UI

The UI (shown in fig. 5.1) consists of three parts:

1. The topmost part is a treeview of all the folders and files that have been imported and can be converted.
2. The bottom-left part shows the Ecore model of the selected file.
3. The bottom-right part shows the resulting Alloy file.

The commands are accessed by the context menu of the mouse on elements of the treeview. Table 5.1 shows all the commands, which element (scope) they apply to and a description. The scope is either the root element, a folder or a file.

Command	Scope	Description
Add directory	Root	Adds a folder to the program and performs the discover function on that folder.
Delete	Folder	Removes a folder from the program.
Discover	Root & Folder	Adds all files in the folder(s) to the program and detects if their are changes in those files since the last conversion. Those files with changes are added to the to be converted list.
Update	Root	Triggers the conversion for all files not yet converted.
Force update	all	Triggers the conversion for all files in the scope.
Export stats	Root & Folder	Exports the statistics for the scope to a CSV file.

Table 5.1: All Ecore2Alloy commands with scope and description

5.2.1.1 Statistics provided by the UI

The statistics provided by Ecore2Alloy are the following (shown in table 5.2):

Column	Description
A.	Number of constraints before the conversion into Alloy
B.	Number of annotations in the original Ecore file
C.	Name of the parent folder of the Ecore file
D.	Name of the Ecore file
E.	Whether oclxmi2ecore has been used
F.	Whether constraints have been used
G.	Number of signatures in Alloy (-1 means not available)
H.	Status of the conversion
I.	Status of the instance generation
J.	Whether there has been an Alloy parse error
K.	Whether there is a possible instance
L.	Whether there is an error or not in the generated instance
M.	Whether there is an error or not in the generated instance (with detail)

Table 5.2: Statistics provided by Ecore2Alloy

5.2.2 Library API

The API for the Ecore to Alloy conversion has been described in section 3.5. For the conversion Alloy to Ecore the API is described in section 4.5.

5.3 Integration into Lightning

The "Ecore2Alloy" library is called from a few relevant parts in Lightning.

5.3.1 Integration of the E2A transformation

For the E2A conversion there are two points in Lightning where an import could necessitate a conversion from Ecore to Alloy. Those being the import existing language menu item in the *GUI* class (see listing 5.1) and the import ASM context menu item for the *ASMFolderTreeNode* class (see listing 5.2).

```
1 Utilities.copy(source, target);
2 ConverterAdapter.convertLanguageFolder(target); // added this line
3 GUI.this.workspace.loadExistingLanguage(target);
```

Listing 5.1: Lightning's import functionality for language folders

```
1 if(!target.getName().endsWith(".als")) {
2   if(!ConverterAdapter.convertToAlloy(target)) {
3     JOptionPane.showMessageDialog(null, "There was an error importing the ecore
      model.", "Import failed", JOptionPane.ERROR_MESSAGE);
4     return;
5   }
6   String alloy = target.getPath();
7   alloy = Utilities.renameFileExtension(alloy, "als");
8   target = new File(alloy);
9 }
```

Listing 5.2: Lightning's import functionality for single model files which deals specifically with the conversion

Both points call a function in the *ConverterAdapter* class which groups the configuration options for E2A from the Lightning options together and calls the E2A API. ConverterAdapter also provides functions for converting a whole folder of models and distinguishes which conversion to apply based on the file extension (see listing 5.3).

```
1  package lu.uni.lightning.util;
2  public class ConverterAdapter {
3    public static void convertLanguageFolder(File target) {
4      String languageFolder = target.getAbsolutePath();
5      File ASM_Folder = new File (languageFolder+File.separator+"ASM");
6      int num_files = 0;
7      int num_successful_converts = 0;
8      File[] list = ASM_Folder.listFiles(new ecoreFileFilter());
9      num_files = list.length;
10     for(File asm_file : list)
11       if(convertToAlloy(asm_file))
12         num_successful_converts++;
13     JOptionPane.showMessageDialog(null, num_successful_converts + "/" + num_files
14         + " ecore models were successfully imported.", "Import info", JOptionPane.
       INFORMATION_MESSAGE);
15   }
16   public static boolean convertToAlloy(File subject) {
17     if(subject.getName().endsWith(".uml"))
18       return convertUMLToAlloy(subject);
19     else if(subject.getName().endsWith(".xmi"))
20       return convertUMLToAlloy(subject);
21     else if(subject.getName().endsWith(".ecore"))
22       return convertEcoreToAlloy(subject);
23     return false;
24   }
25   // convertUMLToAlloy directly invokes UML2Alloy
26   private static boolean convertUMLToAlloy(File subject);
27   private static boolean convertEcoreToAlloy(File subject) {
28     return Converter.runEcore2Alloy(subject.getAbsolutePath(), null, Config.
       getEcore2AlloyConfig());
29   }
30 }
```

Listing 5.3: Lightning's ConverterAdapter

5.3.2 Integration of the A2E transformation

In the class SolutionViewer we added a button xmi, which is only enabled if the Alloy file has an Ecore model of the same name (see listing 5.4). This button opens a save dialog box.

If there is a valid instance file chosen it extracts the A4Solution from the SolutionViewer-Model assciated to the SolutionViewer and finally calls SolutionXMIExporter.export2XMI with the A4Solution, the paths to the Ecore file and the instance file as arguments (see listing 5.5).

```
1  private JButton xmi =new JButton(new ImageIcon(this.getClass().getResource("/save.
       png")));
2  private void init_toolbar() {
3    this.xmi.setToolTipText("save this as an XMI file" );
4    this.xmi.addActionListener(new ActionListener() {
5      // ...
6    });
7    // ...
8    String alloy = SolutionViewer.this.model.getAlloy();
9    File ecore = new File(Utilities.renameFileExtension(alloy, "ecore"));
10   this.xmi.setEnabled(ecore.isFile() && ecore.canRead());
11 }
```

Listing 5.4: Adding the Save instance button to Lightning

```
String alloy = SolutionViewer.this.model.getAlloy();
String ecore = Utilities.renameFileExtension(alloy, "ecore");
JFileChooser m = new JFileChooser();
FileNameExtensionFilter filter = new FileNameExtensionFilter("XMI files", "xmi");
m.setFileFilter(filter);
m.setDialogTitle("Save this instance...");
m.showSaveDialog(null);
File f = m.getSelectedFile();
if(f == null)
  return;
String instance = f.getAbsolutePath();
if(!Utilities.getFileExtension(instance).equalsIgnoreCase("xmi"))
  instance = instance + ".xmi";
try {
  SolutionViewerModel model = SolutionViewer.this.model;
  VIEW_MODE old_view = model.getViewMode();
  A4Solution solution = null;
  try {
    model.setViewMode(VIEW_MODE.noCS);
    solution = model.getInstanceToBeShown().getSolution();
  } catch (InvalidPromotedInstanceException e1) {
    e1.printStackTrace();
  } finally {
    model.setViewMode(old_view);
  }
  SolutionXMIExporter.export2XMI(solution, ecore, instance);
} catch (Exception e1) {
  e1.printStackTrace();
  JOptionPane.showMessageDialog(null, e1.getMessage(), "Error exporting instance to
     xmi", JOptionPane.ERROR_MESSAGE);
}
```

Listing 5.5: The instance save button functionality

5.4 Improving usability of Lightning with additional features

5.4.1 About dialog

This dialog (encapsulated as the lu.uni.lightning.ui.dialogs.AboutDialog class) shows licensing information for Lightning and all dependent packages.

5.4.2 Keyboard shortcuts

• Ctrl-Q	Quit application
• Ctrl-W	Close tab/window
• Esc	Close dialog window (Settings & About dialogs)
• Ctrl-Tab	Show the next tab
• Ctrl-Shift-Tab	Show the previous tab

TABLE 5.3: The keyboard shortcuts added to Lightning

Tabel 5.3 shows the keyboard shortcuts added to Lightning. The tab shifting only works if the focus is in the tab pane. The shortcuts are added in the following classes where applicable: lu.uni.lightning.ui.GUI, lu.uni.lightning.ui.dialogs.SourceEditor, PreferenceDialog and AboutDialog.

Chapter 6

Evaluation

6.1 Methodology

To test the conversion process we used the standalone UI to convert some test models, because this allows the automated conversion and validation of a larger number of models than a manual approach via Lightning. We assume that because we use the same library functions that the Import/Export from Lightning would use that the result would be the same. We test this assumption by converting a small sample of those models with Lightning and validating the exported instances with Eclipse.

We used the models from the following model zoos in addition to three of our own models to test the transformations:

- AtlantEcore [33]
- Remodd [34]

6.2 Testing

We also used these test models with "Ecore2Alloy" for debugging, where we took the results and categorized the error types and for every type picked a model. Concentrating on these models we searched for the root causes of the different error types, which we then resolved.

6.2.1 Errors found in E2A

Errors in the conversion from Ecore model to Alloy model are all described in section 3.6. Those being: Multiple inheritance, abstract signatures, enum literals with conflicting names, String and other datatypes, key collision for a map, index out of bounds exception and a type resolution error.

6.2.2 Errors found in A2E

The errors that have appeared in the Alloy instance to Ecore instance conversion are errors due to missing constraints of aggregations, BaseTypes withouth Java classes or an inability to validate the resulting instance in Eclipse. These are described in more detail in section 4.6.

6.3 Result

The complete result dataset is shown in Appendix C with the columns as described in section 5.2.1.1.

The results are presented in tables 6.1, 6.2, 6.3 and 6.4. They show first the result of the model transformation Ecore model to Alloy model. Of those where an Alloy model was generated, they show the result of the instance generation and validation. Then there is a list of all parse errors. Finally the last table shows which validation errors there were.

#Models	Percent	Category
220	67.9 %	OK
69	21.3 %	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by the UML profile for Alloy was encountered. Details: |OCL-Assertion violated: context Class_::getSuperclasses() pre: self.getGeneralizations() -> size <= 1
15	4.6 %	java.lang.NullPointerException
10	3.1 %	java.lang.NullPointerException from Uml2Alloy
8	2.5 %	java.lang.IndexOutOfBoundsException: Index: 1, Size: 1
1	0.3 %	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException: Unresolved reference '/1/Non'. (MAS.ecore, 9, 163)
1	0.3 %	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException: Unresolved reference '/0/Solution'. (PASSI.ecore, 6, 252)
324	100 %	Total

Table 6.1: Results of the model transformation

#Models	Percent	Percent of all models	Category
1	0.5 %	0.3 %	java.lang.OutOfMemoryError: Java heap space
1	0.5 %	0.3 %	Recursive containment not allowed
5	2.3 %	1.5 %	Parse error
2	0.9 %	0.6 %	Empty object created
12	5.5 %	3.7 %	Empty result
22	10.0 %	6.8 %	Instance generated; ERROR validating
177	80.5 %	54.6 %	Instance generated; OK validating
220	100.0 %	67.9 %	Total

Table 6.2: Results of the instance generation & instance transformation

line 23, column 2, filename=C:\Users\CK\Dropbox\2013 WS\AtlantEcore\XUL-Interactorl.als|Two overlapping signatures cannot have|two fields with the same name "id":||1) one is in sig "this/Interactor"|line 23, column 2, filename=C:\Users\CK\Dropbox\2013 WS\AtlantEcore\XUL-Interactorl.als||2) the other is in sig "this/Windows"|line 9, column 2, filename=C:\Users\CK\Dropbox\2013 WS\AtlantEcore\XUL-Interactorl.als
line 37, column 16, filename=C:\Users\CK\Dropbox\2013 WS\AtlantEcore\SPL.als|The name "Direction" cannot be found.
line 38, column 45, filename=C:\Users\CK\Dropbox\2013 WS\AtlantEcore\CompanyStructure.als||This name is ambiguous due to multiple matches:|sig this/Store|field this/Company <: Store
line 48, column 11, filename=C:\Users\CK\Dropbox\2013 WS\AtlantEcore\requirement.als|The name "Date" cannot be found.
line 58, column 2, filename=C:\Users\CK\Dropbox\2013 WS\Ecore2Alloy\remodd\WorkflowMetamodel.als|Two overlapping signatures cannot have|two fields with the same name "guard":||1) one is in sig "this/Decision"|line 58, column 2, filename=C:\Users\CK\Dropbox\2013 WS\Ecore2Alloy\remodd\WorkflowMetamodel.als||2) the other is in sig "this/ActivityGroup"|line 36, column 2, filename=C:\Users\CK\Dropbox\2013 WS\Ecore2Alloy\remodd\WorkflowMetamodel.als

Table 6.3: Parsing errors

#Models	Percent	Category
4	18,2 %	The required feature 'x' of 'y' must be set
1	4,5 %	The feature 'x' of 'y' with 0 values must have at least 1 values
12	54,5 %	The feature 'x' of 'y' contains a bad value
1	4,5 %	An object may not circularly contain itself
4	18,2 %	The name 'x' is not well formed
22	100.0 %	Total

Table 6.4: Validation errors

6.3.1 Detailed look at models with constraints

Tables 6.5, 6.7, 6.8 and 6.9 show the same statistics as above just for models from Remodd, that is models with constraints in a separate OCLXMI file.

Table 6.6 gives special attention to the number of constraints present in the Ecore files that successfully generate Alloy files.

#Models	Percent	Category
5	31.3 %	OK
4	25.0 %	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by the UML profile for Alloy was encountered. Details: |OCL-Assertion violated: context Class_::getSuperclasses() pre: self.getGeneralizations() -> size <= 1
5	31.3 %	java.lang.NullPointerException
2	12.5 %	java.lang.NullPointerException from Uml2Alloy
16	100.0 %	Total

Table 6.5: Results of the model transformation (models with constraints)

#Constraints before OCLXMI2Ecore	#Constraints after OCLXMI2Ecore	Model name
0	0	CSPF.ecore
0	3	DC.ecore
0	0	er2mof.ecore
0	0	RBAC.ecore
0	0	WorkflowMetamodel.ecore
0	3	Total

Table 6.6: Number of constraints (models with constraints & OK model transformation)

#Models	Percent	Percent of all models	Category
1	20.0 %	6.3 %	Parse error
3	60.0 %	18.8 %	Instance generated; ERROR validating
1	20.0 %	6.3 %	Instance generated; OK validating
5	100.0 %	31.3 %	Total

TABLE 6.7: Results of the instance generation & instance transformation (models with constraints)

line 58, column 2, filename=C:\Users\CK\Dropbox\2013 WS\Ecore2Alloy\remodd\WorkflowMetamodel.als|Two overlapping signatures cannot have|two fields with the same name "guard":||1) one is in sig "this/Decision"|line 58, column 2, filename=C:\Users\CK\Dropbox\2013 WS\Ecore2Alloy\remodd\WorkflowMetamodel.als||2) the other is in sig "this/ActivityGroup"|line 36, column 2, filename=C:\Users\CK\Dropbox\2013 WS\Ecore2Alloy\remodd\WorkflowMetamodel.als

TABLE 6.8: Parsing error (models with constraints)

#Models	Percent	Category
2	66,6 %	The required feature 'x' of 'y' must be set
1	33,3 %	The feature 'x' of 'y' with 0 values must have at least 1 values
3	100.0 %	Total

TABLE 6.9: Validation errors (models with constraints)

6.3.2 Sample result validation in Eclipse

Table 6.10 shows the result of importing a model into Lightning, generating and exporting an instance in/from Lightning and validating that instance in Eclipse. The category from previous tests (in Ecore2Alloy) is shown as well. Idem means the result from Lightning and Eclipse matches the one from Ecore2Alloy.

Group	Model name	Category (from previous tests)	Result
CK	company.ecore	OK	idem
CK	flowchartdsl.ecore	OK	idem
Remodd	BaseIDL.ecore	UMLProfile4AlloyException	idem
Remodd	CMOFConstraints.ecore	java.lang.NullPointerException	idem
Remodd	ComponentIDL.ecore	java.lang.NullPointerException from Uml2Alloy	idem
Remodd	CSPF.ecore	ERROR validating	OK
Remodd	DC.ecore	ERROR validating	OK
Remodd	er2mof.ecore	OK	idem
Remodd	WorkflowMetamodel.ecore	Parse error	idem
AtlantEcore	ACG.ecore	UMLProfile4AlloyException	idem
AtlantEcore	ATL.ecore	java.lang.NullPointerException	idem
AtlantEcore	Book.ecore	OK	idem
AtlantEcore	Bossa.ecore	java.lang.NullPointerException from Uml2Alloy	idem
AtlantEcore	BPMN.ecore	Empty object created	idem
AtlantEcore	COBOL.ecore	ERROR validating	OK
AtlantEcore	Cocus.owl.ecore	IndexOutOfBoundsException	idem
AtlantEcore	CompanyStructure.ecore	Parse error	idem
AtlantEcore	HAL.ecore	ERROR validating	OK
AtlantEcore	KDMSimplified.ecore	ERROR validating	OK
AtlantEcore	Klaper.ecore	Recursive containment not allowed	OK
AtlantEcore	MAS.ecore	UnresolvedReferenceException	idem
AtlantEcore	MavenProject.ecore	empty result	idem
AtlantEcore	ODP-CV.ecore	ERROR validating	OK
AtlantEcore	requirement.ecore	Parse error	idem
AtlantEcore	XUL-Interactorl.ecore	Parse error	idem

Table 6.10: Results of validating model instances with Eclipse

6.3.3 Causes for errors

We found the following causes for errors:

- The UMLProfile4AlloyException is because UML2Alloy does not support multiple inheritance.
- The NullPointerExceptions have their root in a non-resolved type (see 3.6.6).
- The IndexOutOfBoundsException is due two eReferences where only one has the other as its opposite (see 3.6.7).
- The OutOfMemoryError seems to be caused by an error in EMF, but further debugging is needed to find a conclusive cause.
- The "Recursive containment not allowed" errors are caused by missing constraints for the containment relationship (see 4.6.1).
- The parsing errors "Two overlapping signatures cannot have two fields with the same name" mean that a signature has a field that overwrites a field from a super-class, which is not allowed.
- The parsing error "This name is ambiguous due to multiple matches" is due to a field and a signature having the same name. The only solution to this would be to prepend every class name by "this/". We chose not to do this because of the rarity of this case and it would make the resulting Alloy model harder to read and it can be manually added. The optimal solution would be to programmatically detect the problem and only then apply the solution.
- The parsing error "The name 'x' cannot be found" is due to a datatype that is not among the importable datatypes (see 3.6.4).
- The "Empty object created" means that an Alloy file contains no signatures. This is due to the conversion algorithm picking the wrong package to convert in Ecore models with multiple packages.
- An "Empty result" means that only an empty result would satisfy that model and constraints. A likely cause is that the default scope has been insufficient to find another solution.

Due to time constraints we were unable to identify the root causes of the UnresolvedReferenceException and the remaining ERROR validations.

Chapter 7

Conclusion

7.1 Summary

We successfully wrote a library to convert Ecore models into Alloy models and Alloy instances into Ecore instances and have integrated that functionality into Lightning. The evaluation proves that this system works, but there are still problems.

One important problem is the limitations on constraints. This becomes apparent in that only one model of all 17 models with constraints could be converted into an Alloy model with constraints. This is mainly due to the limitations of UML2Alloy, but it could also be due to errors in the OCLXMI2Ecore library. We did not test for errors in the OCLXMI2Ecore library.

While we achieved a successful conversion, instance generation and validation for 54.6 % of all models it should be possible to improve this up to 77.5 %.

This value is calculated by removing those errors that are caused by language and library restrictions (multiple inheritance, UnresolvedReferenceException and Parse errors about overwriting a field from a super-class).

$$1.0 - (69 + 2 + 2)/324 = 0.775$$

Another small problem is that little regard is given to the quality of the instance generated except that it is not empty and validates according to its model. Therefore there could be model instances with just one element.

7.2 Limitations

The constraint handling is pretty limited (no user-defined functions and a restricted function set(no iterative functions)). This is due to the restrictions of UML2Alloy.

Multiple inheritance is not supported, because UML2Alloy does not support multiple inheritance.

The algorithm has trouble resolving types not defined in the Ecore model file and using datatypes not listed in table 3.5.

The algorithm selects the first Ecore package to transform that is not named "PrimitiveTypes".

The algorithm cannot use types (like enumerations) from packages other than the one selected.

Datatypes may need the instance Java class if they are not caught among those datatypes where the Java class can be guessed by the name (see 4.2).

7.3 Future Work

If the conversion should support constraints in any greater capacity in the future, it is imperative that OCL handling will need to be improved. One improvement would be to allow user-defined functions. To do this it is necessary to extend UML2Alloy and to further test the effectiveness of OCLXMI2Ecore.

Because UML2Alloy does not support multiple inheritance and Alloy has no adhoc support for multiple inheritance a solution enabling multiple inheritance would require different handling than UML2Alloy [9, p. 96].

An important point to improve is the resolution of types not included in the Ecore package selected and the selection of the Ecore package to transform itself as this has an important impact on the resulting model.

Also another point to improve is coping with two EReferences where only one has the other as its opposite.

Another future improvement for Lightning would be to visualize the meta-model in the more familiar class diagram notation.

A better definition of the scope would be helpful so that more meaningful instances can be generated. The automatic determination of scope is an active area of research.

Appendix A

Lightning license

Appendix B

Tables with license information

Program	contained in	Licence
Alloy Analyzer 4	Alloy	MIT
Kodkod	Alloy	MIT
JavaCup	Alloy	~MIT
SAT4J	Alloy	GNU lesser general public licence v2.1
ZChaff	Alloy	Princeton University
MiniSat	Alloy	MIT

UML2Alloy	UML2Alloy	as is, no warranty
Alloy3	UML2Alloy	GNU general public licence v2
Alloy4	UML2Alloy	MIT
Dresden OCL Library	UML2Alloy	GNU lesser general public licence v3
Kent Modelling Framework	UML2Alloy	Common Public License - v 1.0
Log4j	UML2Alloy	Apache License v2.0
LoggingOutputStream Class	UML2Alloy	Apache License v1.1
OCL4Java	UML2Alloy	GNU general public licence v2
Swingworker Class	UML2Alloy	GNU lesser general public licence v2.1
Xerces	UML2Alloy	Apache License v2.0

Licence	Link with code using a different license	Release changes under a different license	Modify	Distribute	Warranty	Limited Commercial Use	Commercial Use	Sublicense	Sublicense	Hold Liable	Use Trademark	Include Copyright Notice	Include License	Include Original	State Changes	Disclose Source	Non-Static Linkage
MIT	Yes	Yes	Yes	Yes			Yes	Yes		No		Must					
Princeton University	Yes		Yes				ask			No	No	Must	Must				
GNU GPL v2			Yes	Yes	Yes	Yes			No	No				Must		Must	
GNU lesser GPL v2.1			Yes	Yes	Yes	Yes				No				Must			Must
GNU lesser GPL v3		Yes	No	Yes	Yes	Yes				No				Must			Must
Common Public License - v 1.0		Yes	No	Yes	Yes	Yes				No		Must	Must			Must	
Apache License v1.1		Yes		Yes			?			No	No	Must	Must				
Apache License v2.0		Yes	Yes	Yes	Yes		Yes	Yes		No	No	Must	Must		Must		

http://www.tldrlegal.com/license/mit-license
http://en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licenses

Appendix C

Evaluation result dataset

0	1	src	company.ecore	N	N	3	OK	N	Y	OK	Diagnostic OK source=org.
0	0	src	flowchartdsl.ecore	N	Y	6	OK	N	Y	OK	Diagnostic OK source=org.
0	0	src	flowchartdsl2.ecore	N	Y	6	OK	N	Y	OK	Diagnostic OK source=org.
0	13	remodd	BaseIDL.ecore	Y	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b				
0	0	remodd	BMethod.ecore	Y	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b				
0	1211	remodd	CMOFConstraints.ecore	Y	N	-1	java.lang.NullPointerException				
0	30	remodd	ComponentIDL.ecore	Y	N	-1	java.lang.NullPointerException from Uml2Alloy				
0	0	remodd	CSPF.ecore	Y	Y	24	OK	N	Y	ERROR	Diagnostic ERROR source=
0	0	remodd	cwm.ecore	Y	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b				
3	26	remodd	DC.ecore	Y	Y	26	OK	N	Y	ERROR	Diagnostic ERROR source=
0	181	remodd	DG.ecore	Y	N	-1	java.lang.NullPointerException				
0	1211	remodd	EMOFConstraints.ecore	Y	N	-1	java.lang.NullPointerException				
0	8	remodd	er2mof.ecore	Y	Y	20	OK	N	Y	OK	Diagnostic OK source=org.
0	51	remodd	OCLexpressions.ecore	Y	N	-1	java.lang.NullPointerException from Uml2Alloy				
0	23	remodd	OCLtypes.ecore	Y	N	-1	java.lang.NullPointerException				
0	0	remodd	RBAC.ecore	Y	Y	14	OK	N	Y	ERROR	Diagnostic ERROR source=
0	180	remodd	Sad3.ecore	Y	N	-1	java.lang.NullPointerException				
0	33	remodd	SAM.ecore	Y	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b				
0	37	remodd	WorkflowMetamodel.ecore	Y	Y	49	OK	line 58, column 2, filename= Y			
0	0	AtlantEcore	AbstractSyntaxStereotypesUML1.4.ecore	N	Y	10	OK	N	Y	OK	Diagnostic OK source=org.
0	0	AtlantEcore	ACG.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b				
0	0	AtlantEcore	ACME.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b				
0	0	AtlantEcore	ADELFE.ecore	N	Y	19	OK	N	Y	OK	Diagnostic OK source=org.
0	0	AtlantEcore	Agate.ecore	N	Y	71	OK	N	Y	OK	Diagnostic OK source=org.
0	0	AtlantEcore	Amble.ecore	N	Y	16	OK	N	Y	OK	Diagnostic OK source=org.
0	0	AtlantEcore	Ant.ecore	N	Y	49	OK	N	Y	OK	Diagnostic OK source=org.
0	0	AtlantEcore	AntScripts.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b				
0	0	AtlantEcore	AnyLogic.ecore	N	Y	20	OK	N	Y	OK	Diagnostic OK source=org.
0	0	AtlantEcore	Architectural_Description.ecore	N	Y	6	OK	N	Y		empty result
0	0	AtlantEcore	ASM.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b				
0	0	AtlantEcore	AsmL.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b				
0	0	AtlantEcore	ATL.ecore	N	N	-1	java.lang.NullPointerException				
0	0	AtlantEcore	ATOM.ecore	N	Y	15	OK	N	Y	OK	Diagnostic OK source=org.
0	0	AtlantEcore	Automaton.ecore	N	Y	4	OK	N	Y	OK	Diagnostic OK source=org.
0	0	AtlantEcore	AWKPrograms.ecore	N	Y	12	OK	N	Y	OK	Diagnostic OK source=org.
0	0	AtlantEcore	BibTeX.ecore	N	Y	46	OK	N	Y	OK	Diagnostic OK source=org.
0	0	AtlantEcore	BibTeX1.1.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b				
0	0	AtlantEcore	BIBTEXML.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b				
0	0	AtlantEcore	BMM.ecore	N	Y	32	OK	N	Y	OK	Diagnostic OK source=org.
0	0	AtlantEcore	bmmOmg.ecore	N	Y	32	OK	N	Y	OK	Diagnostic OK source=org.
0	0	AtlantEcore	Book.ecore	N	Y	3	OK	N	Y	OK	Diagnostic OK source=org.

0	0	AtlantEcore	Bossa.ecore	N	N	116	java.lang.NullPointerException from Uml2Alloy						
0	0	AtlantEcore	BPEL.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	BPMN.ecore	N	Y	0	OK	Empty object created	N	Y			
0	0	AtlantEcore	BQL.ecore	N	Y	8	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	Bugzilla.ecore	N	Y	100	OK		N	Y	ERROR	Diagnostic ERROR source=(	
0	0	AtlantEcore	BusinessEntityModel.ecore	N	Y	10	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	BusinessProcessModel.ecore	N	Y	26	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	C.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	CADM.ecore	N	Y	27	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	CDE.ecore	N	Y	10	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	CFG.ecore	N	Y	5	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	ChocoModel.ecore	N	Y	25	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	Class.ecore	N	Y	7	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	classDiagram.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	ClassicModels.ecore	N	Y	12	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	CML.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	cmt.owl.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	COBOL.ecore	N	Y	33	OK		N	Y	ERROR	Diagnostic ERROR source=(	
0	0	AtlantEcore	Cocus.owl.ecore	N	N	-1	java.lang.IndexOutOfBoundsException: Index: 1, Size: 1						
0	0	AtlantEcore	Collaborations_Interactions_UML.ecore	N	Y	13	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	CompanyStructure.ecore	N	Y	7	OK	line 38, column 45, filename	Y				
0	0	AtlantEcore	ComponentUML.ecore	N	Y	5	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	Conference.owl.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	confious.owl.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	confOf.owl.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	Contact.ecore	N	Y	4	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	ControllerUML.ecore	N	Y	9	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	CORBAComponent.ecore	N	Y	15	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	CPL.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	CPP.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	CPR.ecore	N	Y	7	OK		N	Y		empty result	
0	0	AtlantEcore	Cristal.ecore	N	Y	9	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	crs_dr.owl.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	CSharp.ecore	N	Y	12	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	CSM.ecore	N	Y	21	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	CWMCore.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	CWMRelationalData.ecore	N	Y	14	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	deployment.ecore	N	Y	6	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	DeploymentReport.ecore	N	Y	63	OK		N	Y	ERROR	Diagnostic ERROR source=(	
0	0	AtlantEcore	DiagramInterchange.ecore	N	N	-1	java.lang.NullPointerException from Uml2Alloy						
0	0	AtlantEcore	DocBook.ecore	N	Y	7	OK		N	Y	OK	Diagnostic OK source=org.(	

0	0	AtlantEcore	DoDAF-OV5.ecore	N	Y	32	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	DoDAF-SV4.ecore	N	Y	44	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	DoDAF-SV5.ecore	N	Y	36	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	DoDAF.ecore	N	Y	58	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	DOT.ecore	N	Y	28	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	DotNET_SystemReflection.ecore	N	Y	155	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	DSL.ecore	N	Y	15	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	DSLModel.ecore	N	Y	16	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	DSLtools.ecore	N	Y	15	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	DTD.ecore	N	Y	34	OK		N	Y	ERROR	Diagnostic ERROR source=(
0	0	AtlantEcore	DTMP.ecore	N	Y	5	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	DXF.ecore	N	Y	4	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	EAI.ecore	N	Y	7	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	ebXML.ecore	N	Y	26	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	EclipseLaunchConfigurations.ecore	N	Y	10	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	EclipsePlugIn.ecore	N	Y	8	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	edas.owl.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b
0	0	AtlantEcore	EG.ecore	N	Y	13	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	ekaw.owl.ecore	N	N	-1	java.lang.IndexOutOfBoundsException: Index: 1, Size: 1					
0	0	AtlantEcore	EQN.ecore	N	Y	15	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	EXPRESS.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b
0	0	AtlantEcore	EXPRESSb.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b
0	0	AtlantEcore	Extended_UML_Core_Package.ecore	N	Y	20	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	Family.ecore	N	Y	3	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	FeatureDiagrams.ecore	N	Y	8	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	FiniteStateMachine.ecore	N	Y	7	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	FlatSignalFlow.ecore	N	Y	11	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	GAIA.ecore	N	Y	20	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	Gantt.ecore	N	Y	12	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	GenericEditor.ecore	N	Y	12	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	GenericOutline.ecore	N	Y	7	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	GeoTrans.ecore	N	Y	5	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	Grafcet.ecore	N	Y	11	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	GraphML.ecore	N	Y	24	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	GraphVizDot.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b
0	0	AtlantEcore	GUI.ecore	N	Y	19	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	GWPNV0.ecore	N	Y	3	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	GWPNV1.ecore	N	Y	3	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	GWPNV2.ecore	N	Y	5	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	GWPNV3.ecore	N	Y	6	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	GWPNV4.ecore	N	Y	6	OK		N	Y	OK	Diagnostic OK source=org.(

0	0	AtlantEcore	GWPNV5.ecore	N	Y	7	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	HAL.ecore	N	Y	62	OK		N	Y	ERROR	Diagnostic ERROR source=(
0	0	AtlantEcore	HierarchicalSignalFlow.ecore	N	Y	9	OK		N	Y		empty result
0	0	AtlantEcore	HierarchicalStateMachine.ecore	N	Y	17	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	HPROF.ecore	N	Y	33	OK		N	Y	ERROR	Diagnostic ERROR source=(
0	0	AtlantEcore	HTML.ecore	N	Y	60	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	HybridAutomata.ecore	N	Y	4	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	iasted.owl.ecore	N	N	-1	java.lang.IndexOutOfBoundsException: Index: 1, Size: 1					
0	0	AtlantEcore	IEEE1471ConceptualModel.ecore	N	Y	15	OK		N	Y		empty result
0	0	AtlantEcore	IEEE1471ViewpointM2.ecore	N	Y	15	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	ifc2x3.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	IMSTransactionMessage.ecore	N	Y	16	OK		N	Y	ERROR	Diagnostic ERROR source=(
0	0	AtlantEcore	IntegrationOfOCL_ExpressionsInUML.ecore	N	Y	9	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	IRL.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	J2SE5.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	Java-20040316.ecore	N	Y	11	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	JAVA3.ecore	N	Y	13	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	JavaAbstractSyntax.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	JavaProject.ecore	N	Y	16	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	JavaSource.ecore	N	Y	6	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	Jess.ecore	N	Y	45	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	KDM.ecore	N	Y	345	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	KDMSimplified.ecore	N	Y	15	OK		N	Y	ERROR	Diagnostic ERROR source=(
0	0	AtlantEcore	Klaper.ecore	N	Y	18	OK	Recursive containment not a	N	Y		
0	0	AtlantEcore	KM3.ecore	N	Y	14	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	KML.ecore	N	Y	102	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	LaTeX.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	LQN.ecore	N	Y	34	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	M.ecore	N	Y	35	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	Make.ecore	N	Y	11	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	Mantis.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	Marte.ecore	N	N	-1	java.lang.NullPointerException					
0	0	AtlantEcore	MAS.ecore	N	N	-1	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException: Unresolved reference '/1/Non'. (M					
0	0	AtlantEcore	Matlab.ecore	N	Y	49	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	Maude.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	MavenMaven.ecore	N	Y	59	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	MavenProject.ecore	N	Y	8	OK		N	Y		empty result
0	0	AtlantEcore	Measure.ecore	N	Y	18	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	METAH.ecore	N	Y	15	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	Metrics.ecore	N	Y	8	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	MICRO.owl.ecore	N	N	-1	java.lang.IndexOutOfBoundsException: Index: 1, Size: 1					

0	0	AtlantEcore	MiningMart.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	MiningMart_SimplifiedMetamodel.ecore	N	Y	15	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	MiningMart_ViewCaseRepresentation.ecore	N	N	-1	java.lang.NullPointerException from Uml2Alloy					
0	0	AtlantEcore	MiningMart_ViewDataRepresentation.ecore	N	Y	7	OK	N	Y	OK	Diagnostic OK source=org.(	
0	646	AtlantEcore	mlhim2.ecore	N	N	-1	java.lang.NullPointerException					
0	0	AtlantEcore	MoDAF-AV.ecore	N	Y	49	OK	N	Y		empty result	
0	0	AtlantEcore	MoMM.ecore	N	Y	6	OK	N	Y		empty result	
0	0	AtlantEcore	MonitorProgram.ecore	N	Y	29	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	MSOfficeExcel_SpreadsheetMLBasicDef.ecore	N	Y	26	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	MSOfficeExcel_SpreadsheetMLPrintingSetup.ecore	N	Y	62	OK	N	Y	ERROR	Diagnostic ERROR source=(	
0	0	AtlantEcore	MSOfficeExcel_SpreadsheetMLSimplified.ecore	N	Y	18	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	MSOfficeExcel_SpreadsheetMLStyles.ecore	N	N	-1	java.lang.NullPointerException from Uml2Alloy					
0	0	AtlantEcore	MSOfficeExcel_SpreadsheetMLWorkbookProp.ecore	N	Y	35	OK	N	Y	ERROR	Diagnostic ERROR source=(	
0	0	AtlantEcore	MSOfficeExcel_SpreadsheetMLWorksheetOpt.ecore	N	Y	48	OK	N	Y	ERROR	Diagnostic ERROR source=(	
0	0	AtlantEcore	MSOfficeWord_WordprocessingMLBasicDef.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	MSOfficeWord_WordprocessingMLSimplified.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	MSOfficeWord_WordprocessingMLStyles.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	MSOfficeWord_WordprocessingMLTableElts.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	MSProject.ecore	N	Y	4	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	MSProject2.ecore	N	Y	9	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	MSVisio_DatadiagramMLBasicDef.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	MSVisio_DatadiagramMLSimplified.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	MSVisio_DatadiagramMLTextFormat.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	MSVisio_DatadiagramMLXForm.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	MTRANS.ecore	N	Y	11	OK	N	Y		empty result	
0	0	AtlantEcore	MySQL.ecore	N	Y	10	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	News.ecore	N	Y	14	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	OCCAM.ecore	N	Y	22	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	OCL_Expressions.ecore	N	Y	16	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	OCL_Operations.ecore	N	Y	16	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	OCL_Types.ecore	N	Y	16	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	OCL_Values.ecore	N	Y	19	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	ODP-CV.ecore	N	Y	35	OK	N	Y	ERROR	Diagnostic ERROR source=(	
0	0	AtlantEcore	ODP-EV.ecore	N	Y	34	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	ODP-IV.ecore	N	Y	11	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	ODP-NV.ecore	N	Y	29	OK	N	Y	ERROR	Diagnostic ERROR source=(	
0	0	AtlantEcore	ODP-TV.ecore	N	Y	4	OK	N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	OpenConf.owl.ecore	N	N	-1	java.lang.IndexOutOfBoundsException: Index: 1, Size: 1					
0	0	AtlantEcore	OpenQVT.ecore	N	Y	21	OK	N	Y		empty result	
0	0	AtlantEcore	OWL.ecore	N	N	-1	java.lang.NullPointerException from Uml2Alloy					
0	0	AtlantEcore	paperdyne.owl.ecore	N	N	-1	java.lang.IndexOutOfBoundsException: Index: 1, Size: 1					

0	0	AtlantEcore	Parameters.ecore	N	Y	3	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	Pascal.ecore	N	N	-1	java.lang.NullPointerException from Uml2Alloy					
0	0	AtlantEcore	PASSI.ecore	N	N	-1	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException: Unresolved reference '/0/Solution'					
0	0	AtlantEcore	PathExp.ecore	N	Y	5	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	PCS.owl.ecore	N	N	-1	java.lang.IndexOutOfBoundsException: Index: 1, Size: 1					
0	0	AtlantEcore	PDG.ecore	N	Y	6	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	Perceptory.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	Person.ecore	N	Y	2	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	PetriNet.ecore	N	Y	8	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	PetriNet_extended.ecore	N	Y	14	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	PIF.ecore	N	Y	14	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	PL1.ecore	N	Y	17	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	PluginEclipse.ecore	N	N	-1	java.lang.NullPointerException					
0	0	AtlantEcore	PNML_basic.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	PNML_modular.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	PNML_simplified.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	PNML_structured.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	Problem.ecore	N	Y	6	OK		N	Y	ERROR	Diagnostic ERROR source=(
0	0	AtlantEcore	Program.ecore	N	Y	29	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	ProMarte.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	Promenade.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	PRR.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	PtolemyII.ecore	N	N	-1	java.lang.NullPointerException from Uml2Alloy					
0	0	AtlantEcore	Publication.ecore	N	Y	2	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	QoS.ecore	N	Y	25	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	QoS_Characteristic.ecore	N	Y	13	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	QoS_Profile.ecore	N	Y	7	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	QoS_Statement.ecore	N	Y	8	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	QVT.ecore	N	N	-1	java.lang.NullPointerException					
0	0	AtlantEcore	QVT_SimpleRDBMS.ecore	N	Y	7	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	QVT_SimpleUML.ecore	N	Y	9	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	R2ML.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	RDFS.ecore	N	Y	22	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	Relational.ecore	N	Y	5	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	RelationalDBContent.ecore	N	Y	6	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	RelationalDBSchema.ecore	N	Y	6	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	Repository.ecore	N	Y	9	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	Reqtify.ecore	N	Y	12	OK		N	Y	OK	Diagnostic OK source=org.(
0	0	AtlantEcore	requirement.ecore	N	Y	64	OK	line 48, column 11, filename	Y			
0	0	AtlantEcore	RequisitePro.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by					
0	0	AtlantEcore	RSS-2.0.ecore	N	Y	19	OK		N	Y	ERROR	Diagnostic ERROR source=(

0	0	AtlantEcore	sbvrEclipse.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b						
0	0	AtlantEcore	sbvrOMG.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b						
0	0	AtlantEcore	SBVRvoc.ecore	N	Y	129	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	SCADE.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b						
0	0	AtlantEcore	Scilab.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b						
0	0	AtlantEcore	SDM.ecore	N	Y	168	OK		N	Y	ERROR	Diagnostic ERROR source=(	
0	0	AtlantEcore	SecureUML.ecore	N	Y	10	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	SEE_Design.ecore	N	Y	14	OK		N	Y		empty result	
0	0	AtlantEcore	SeminarSchedulingSystem.ecore	N	Y	8	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	Sharengo.ecore	N	Y	11	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	sigkdd.owl.ecore	N	N	-1	java.lang.IndexOutOfBoundsException: Index: 1, Size: 1						
0	0	AtlantEcore	SignalFlow.ecore	N	Y	11	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	SimpleAirlineDomain.ecore	N	Y	10	OK		N	Y	ERROR	Diagnostic ERROR source=(	
0	0	AtlantEcore	SimpleSBVR.ecore	N	Y	34	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	SimulinkStateFlow.ecore	N	Y	25	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	SoftwareQualityControl.ecore	N	Y	11	OK		N	Y	ERROR	Diagnostic ERROR source=(	
0	0	AtlantEcore	SPEM.ecore	N	Y	16	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	SPL.ecore	N	Y	79	OK	line 37, column 16, filename	Y				
0	0	AtlantEcore	SQLDDL.ecore	N	Y	18	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	SQLDML.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b						
0	0	AtlantEcore	Statecharts.ecore	N	Y	10	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	SVG.ecore	N	Y	40	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	SWRC.ecore	N	Y	56	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	SyncCharts.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b						
0	0	AtlantEcore	SysML.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b						
0	0	AtlantEcore	Table.ecore	N	Y	4	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	TextualPathExp.ecore	N	Y	7	OK		N	Y		empty result	
0	0	AtlantEcore	Trace.ecore	N	Y	5	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	TroposActorConcept.ecore	N	Y	7	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	TroposGoalAndPlanConcepts.ecore	N	Y	12	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	TroposIntegratingActorConcept.ecore	N	Y	4	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	UDDI_meta-model_fragment.ecore	N	Y	7	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	UEML.ecore	N	Y	23	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	UEMLExtended.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b						
0	0	AtlantEcore	UEMLExtensionCapturingAgents.ecore	N	Y	29	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	UEMLExtensionCapturingSocialEffects.ecore	N	Y	29	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	UEMLExtensionDynamicAspects.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b						
0	0	AtlantEcore	UEMLExtensionModelingAspects.ecore	N	Y	25	OK		N	Y	OK	Diagnostic OK source=org.(	
0	0	AtlantEcore	UIML-3.0.ecore	N	N	-1	java.lang.NullPointerException						
0	0	AtlantEcore	UML2.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported b						
0	0	AtlantEcore	UMLDI-ActivityGraphs.ecore	N	N	-1	java.lang.NullPointerException						

0	0	AtlantEcore	UMLDI-Collaborations.ecore	N	N	-1	java.lang.NullPointerException						
0	0	AtlantEcore	UMLDI-ModelManagement.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	UMLDI-StateMachines.ecore	N	N	-1	java.lang.NullPointerException						
0	0	AtlantEcore	UMLDI-UseCases.ecore	N	N	-1	java.lang.NullPointerException from Uml2Alloy						
0	0	AtlantEcore	UMLDI.ecore	N	Y	0	OK	Empty object created	N	Y			
0	0	AtlantEcore	UMLForOOClassModeling.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	UML_metamodel_fragment.ecore	N	Y	14	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	UML_UseCases.ecore	N	Y	13	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	UML_withReuseContracts.ecore	N	Y	9	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	UnifiedOntologyLanguage.ecore	N	Y	6	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	UnixFS.ecore	N	Y	6	OK		N	Y		empty result	
0	0	AtlantEcore	useCase.ecore	N	Y	9	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	USECASE1.ecore	N	Y	16	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	USECASE2.ecore	N	Y	15	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	UsiXML-task.ecore	N	Y	19	OK		N	Y	ERROR	Diagnostic ERROR source=e	
0	0	AtlantEcore	vb.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	WebApplications_AbstractModel.ecore	N	Y	8	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	WebApplications_ConceptualModel.ecore	N	Y	19	OK	java.lang.OutOfMemoryErro	N	Y			
0	0	AtlantEcore	WfMC.ecore	N	Y	6	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	WikiTable.ecore	N	Y	7	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	WorkDefinitions.ecore	N	Y	9	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	WSDL.ecore	N	Y	15	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	WSLink.ecore	N	Y	6	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	WTIP_SimpleClass.ecore	N	Y	7	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	WTIP_SimpleRDBMS.ecore	N	Y	4	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	XAML-Perspective.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	XAML-ResourceDictionary.ecore	N	Y	16	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	XHTML.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	XML.ecore	N	Y	6	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	XMorphLanguage_abstractSyntax.ecore	N	N	-1	java.lang.NullPointerException						
0	0	AtlantEcore	XPDL-1.14.ecore	N	Y	54	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	XQuery.ecore	N	Y	19	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	XSchema.ecore	N	N	-1	org.uml2alloy.umlmmsimplifiedprofile.UMLProfile4AlloyException: A feature not supported by						
0	0	AtlantEcore	XSLT.ecore	N	Y	17	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	XUL-Interactorl.ecore	N	Y	38	OK	line 23, column 2, filename=	Y				
0	0	AtlantEcore	yUML.ecore	N	Y	28	OK		N	Y	OK	Diagnostic OK source=org.e	
0	0	AtlantEcore	µOCCAM.ecore	N	Y	59	OK		N	Y	OK	Diagnostic OK source=org.e	

Bibliography

[1] Loïc Gammaitoni. Towards a lightweight modelling language workbench based on alloy, 2013.

[2] Kyriakos Anastasakis, Behzad Bordbar, and Seyyed Shah. From uml to alloy and back again. *Models in Software Engineering*, 6002:158–171, 2010. ISSN 0302-9743. doi: 10.1007/978-3-642-12261-3_16. URL http://link.springer.com/chapter/10.1007%2F978-3-642-12261-3_16.

[3] Wikipedia contributors. Eclipse modeling framework, 2013. URL http://en.wikipedia.org/w/index.php?title=Eclipse_Modeling_Framework&oldid=584613797. [Online; accessed 22-December-2013].

[4] Wikipedia contributors. Xml metadata interchange, 2013. URL http://en.wikipedia.org/w/index.php?title=XML_Metadata_Interchange&oldid=578688312. [Online; accessed 29-December-2013].

[5] Wikipedia contributors. Xpath, 2013. URL http://en.wikipedia.org/w/index.php?title=XPath&oldid=581422684. [Online; accessed 29-December-2013].

[6] Daniel Jackson Joseph P. Near. An imperative extension to alloy. *Lecture Notes in Computer Science*, 5977:118–131, 2010. URL http://link.springer.com/chapter/10.1007/978-3-642-11811-1_10.

[7] Daniel Jackson Wing and Jeannette. Lightweight formal methods. *IEEE Computer*, pages 21–22, 1996. URL http://www.cs.cmu.edu/~wing/publications/JacksonWing96.pdf.

[8] Wikipedia contributors. Alloy (specification language), 2013. URL http://en.wikipedia.org/w/index.php?title=Alloy_(specification_language)&oldid=580666868. [Online; accessed 22-December-2013].

[9] Daniel Jackson. *Software abstractions : logic, language, and analysis*. MIT Press, Cambridge, Mass., rev. edition, 2012. ISBN 9780262017152 (hardcover alk. paper) 0262017156 (hardcover alk. paper). 2011024317 Daniel Jackson. ill. ; 24 cm. Includes bibliographical references (p. [337]-343) and index.

[10] Daniel Jackson et al. Alloy language reference, 2012. URL http://alloy.mit.edu/alloy/documentation/book-chapters/alloy-language-reference.pdf.

[11] Wikipedia contributors. Alloy analyzer, 2013. URL http://en.wikipedia.org/w/index.php?title=Alloy_Analyzer&oldid=580588345. [Online; accessed 23-December-2013].

[12] Wikipedia contributors. Unified modeling language, 2013. URL http://en.wikipedia.org/w/index.php?title=Unified_Modeling_Language&oldid=586682636. [Online; accessed 23-December-2013].

[13] Wikipedia contributors. Object constraint language, 2013. URL http://en.wikipedia.org/w/index.php?title=Object_Constraint_Language&oldid=585780505. [Online; accessed 23-December-2013].

[14] Wikipedia contributors. Model transformation, 2013. URL http://en.wikipedia.org/w/index.php?title=Model_transformation&oldid=574249545. [Online; accessed 30-December-2013].

[15] Erich Gamma. *Design patterns : elements of reusable object-oriented software*. Addison-Wesley professional computing series. Addison-Wesley, Reading, Mass., 1995. ISBN 0201633612 (acid-free paper). 94034264 Erich Gamma ... [et al.]. ill. ; 25 cm. Includes bibliographical references (p. 375-381) and index.

[16] varied. Emf/faq - eclipsepedia, 2013. URL http://wiki.eclipse.org/EMF/FAQ#I_want_to_use_EMF.2C_SDO.2C_or_XSD_in_my_standalone_project.2C_or_include_only_a_working_subset_of_the_code._What_libraries_.28jar_files.29_do_I_need_in_my_CLASSPATH.3F.

[17] Henshin developers. Henshin interpreter - eclipsepedia, 2013. URL http://wiki.eclipse.org/Henshin_Interpreter.

[18] Henshin developers. org.eclipse.emf.henshin.examples.ecore2uml, 2012. URL http://dev.eclipse.org/svnroot/modeling/org.eclipse.emft.henshin/trunk/

plugins/org.eclipse.emf.henshin.examples/src/org/eclipse/emf/henshin/examples/ecore2uml/.

[19] Wikipedia contributors. Qvt, 2014. URL http://en.wikipedia.org/w/index.php?title=QVT&oldid=584270398. [Online; accessed 30-January-2014].

[20] QVTo developers. Qvto/new and noteworthy/helios - eclipsepedia, 2013. URL http://wiki.eclipse.org/QVTo/New_and_Noteworthy/Helios.

[21] varied. qvto examples, 2013. URL https://github.com/eclipse/qvto.

[22] Dave Steinberg. *EMF : Eclipse Modeling Framework*. The eclipse series. Addison-Wesley, Upper Saddle River, NJ, 2nd edition, 2009. ISBN 0321331885 (pbk. alk. paper) 9780321331885 (pbk. alk. paper). URL Tableofcontentsonlyhttp://www.loc.gov/catdir/toc/ecip086/2007049160.html. 2007049160 Dave Steinberg ... [et al.]. ill. ; 24 cm. Includes bibliographical references and index. Eclipse series.

[23] varied. Xmiresource (emf javadoc), 2013. URL http://download.eclipse.org/modeling/emf/emf/javadoc/2.6.0/org/eclipse/emf/ecore/xmi/XMIResource.html#OPTION_USE_XMI_TYPE.

[24] Christian Glodt. lu.uni.lassy.oclxmi2ecore, 2013. URL svn://demos.uni.lux/svn/trunk/lu.uni.lassy.oclxmi2ecore.

[25] Jordi Cabot. jordicabot / umltosql - umltophp/symfony - umltopython/django — bitbucket, 2013. URL https://bitbucket.org/jordicabot/umltosql-umltophp-symfony-umltopython-django.

[26] Kyriakos Anastasakis. Uml2alloy reference manual, 04-May-2009 2009. URL http://www.cs.bham.ac.uk/~bxb/UML2Alloy/files/uml2alloy_manual.pdf.

[27] Mark Richters Martin Gogolla. Transformation rules for uml class diagrams. *Lecture Notes in Computer Science*, 1618:92–106, 1999. URL http://link.springer.com/chapter/10.1007%2F978-3-540-48480-6_8.

[28] Daniel Jackson et al. The alloy analyzer new syntactic features in alloy 4, 2013. URL http://alloy.mit.edu/alloy/documentation/quickguide/a4.html.

[29] Daniel Jackson et al. Alloy - release notes, 2013. URL http://alloy.mit.edu/alloy/release-notes.html.

[30] Kyriakos Anastasakis. Uml2alloy java example, 2009. URL http://www.cs.bham.ac.uk/~bxb/UML2Alloy/example4/files/ExampleAPI.java.

[31] Daniel Jackson et al. Documentation edu.mit.csail.sdg.alloy4compiler.translator, 2013. URL http://alloy.mit.edu/alloy/documentation/alloy-api/index.html?edu/mit/csail/sdg/alloy4compiler/translator/package-summary.html.

[32] Nidhi Singh; Rohit Babbar. Build metamodels with dynamic emf, 2007-11-20 2007. URL http://www.ibm.com/developerworks/library/os-eclipse-dynamicemf/. © Copyright IBM Corporation 2007.

[33] varied. Ecore - atlanmod, 2011. URL http://www.emn.fr/z-info/atlanmod/index.php/Ecore.

[34] remodd_team. Repository for model driven development (remodd) overview, April 19, 2011 2013. URL http://www.cs.colostate.edu/remodd/v1/.

Index