From UML to Alloy and Back Again

Seyyed M.A. Shah, Kyriakos Anastasakis, and Behzad Bordbar

School of Computer Science, The University of Birmingham, Edgbaston, B15 2TT.
United Kingdom.
{szs|kxa|bxb}@cs.bham.ac.uk

Abstract. Model Transformations can be used to bridge the gap between design and analysis *technical spaces* by creating tools that allow a model produced by a designer to be transformed to a model suitable for conducting automated analysis. Such model transformations aim at allowing the designer to benefit from the capabilities provided by analysis tools and languages. If the designer who is not a formal method expert is to benefit from such tools, the outcome of the analysis should also be transformed to the language used in the design domain.
This paper presents a study involving UML2Alloy, a tool for transforming UML models in form of UML class diagrams which are augmented with OCL constraints, to Alloy. The conversion allows analysis of UML models via Alloy, to identify consistencies in those UML models. We present a method of automatically creating a model transformation based on the original UML2Alloy transformation. The new transformation converts Alloy instances into the UML equivalent object diagram. The current technique is presented with the help of an example, along with a prototype implementation using the QVT standard.

1 Introduction

Model Driven Architecture (MDA), among other things, allows bridging the gap between technical spaces and domains [21, 12]. For example, MDD has been extensively used to allow analysis of UML models by transferring UML models to formal languages, which in turn can be used for conducting various types of analysis [6, 18, 11, 2]. One of the motivations behind such approaches is to assist the designer who works with UML to benefit from the advantages of other languages. For example, UML2Alloy [3] transforms models involving UML class diagram and OCL to the Alloy language. Then, Alloy [8] is used to analyse the model for identifying inconsistencies in the design. For example, in case that there is an inconsistency between various parts of the design, Alloy can produce a counter-example helping to reveal the source of inconsistency. Identifying such inconsistencies allows improving the design to produce better software at earlier stages of software development. UML2Alloy has been successfully applied to various domains including Agile Manufacturing [4], Security [7], and Access control [2].

To fulfil the vision of bridging the gap between design and analysis technical spaces, there is a clear need to transfer the outcome of the analysis *back* into the design space. In other words, a design model must be transformed to a language suitable for analysis, and after conducting analysis the result must be transformed back to the design space. Therefore the designer can both produce a model in the design space and receive the feedback of analysis in the design space. In case of UML2Alloy if a counter-example is found in analysis, it must be transferred to a UML object-diagram. That object diagram is an instance of the class diagram in the design space and represents a violation of a property of the system. Otherwise, the developer must be an expert in two languages, in this case Alloy and UML.

UML2Alloy has been shown to be a complex model transformation, [3]. Defining a transformation from Alloy to UML to carry the outcome of analysis conducted by Alloy to an object diagram has proved to be challenging too. Firstly, the transformations from UML to Alloy and from Alloy to UML are not independent. The transformation from instances of the Alloy models to UML must result in an instance of the original UML model, i.e. the created object diagram must comply to the class diagram. Secondly, the reader may think that a bi-directional [19] model transformation would solve the problem. This is not the case, as the second transformation is between the instances of the models that are transformed by the original transformation from UML to Alloy. In other words, the first transformation is carried out at the M2 layer of MOF [15] hierarchy while the second transformation is carried out at the M1 layer.

In this paper we shall present a method for automatically generating the second transformation from the first transformation, i.e. the second transformation which converts the Alloy instance models (analysis) to the original UML form is created automatically from the transformation that maps UML to Alloy models in the first place. We use an off-the-shelf implementation of the Queries Views Transformations (QVT) [16] standard to implement the transformation generator, as well as for the generated transformation. This enables UML developers to harness the power of UML as well as the analytical support of Alloy, hence truly bridging the gap between the two domains.

This paper is structured as follows, in the next section, relevant background information is given. Following this, we present the motivation for this work in Section 3. An outline of our solution with the help of an example and a prototype implementation are presented in Section 4. The paper concludes with a discussion of the solution and related work.

2 Background

2.1 Alloy for the Analysis of UML Models

The Unified Modelling Language (UML) is widely accepted by the software engineering community as the *de facto* standard for the design, implementation and documentation of systems in industry. However, the UML is not intended to support analysis and reasoning of the models created using it. Analysing a model can be essential in identifying the design flaws and removing them at earlier stages of the development process. For this purpose, a number of proposals have been developed that advocate the transformation of UML models to well-established formalisms for the purpose of analysis. In particular, Evans et al. [6] propose the use of Z [26] as the underlying semantics for UML. Snook and Butler [18] suggest the use of B [1], while Kim [11] transforms UML models to Object-Z, by defining a mapping from a subset of the UML meta model to the Object-Z meta model.

In [2], we advocate the formalisation of a subset of the UML, with the help of the Alloy language [8]. Alloy is an increasingly popular declarative textual modelling language based on first-order relational logic. The Alloy language is supported by a tool, the Alloy Analyzer, which provides support for fully automated analysis of Alloy models, with the help of SAT solvers. The tool provides the capability to simulate, check assertions and to debug a model. In simulation, the tool produces an instance of the model that conforms to the constraints of the system. To check if an assertion (i.e. a statement expressed in first-order logic that captures a requirement) holds according to the specification, the tool attempts to generate an instance that invalidates the assertion. To debug over-constrained models, the tool can identify which elements of the model specification are hindering the generation of instances [17]. Alloy has been used for analysing a wide number of protocols and systems [9, 10, 5]. Specifically, Jackson and Sullivan [9] have analysed COM architecture and Khurshid and Jackson [10] have analysed consistency of the International Naming Scheme (INS). Dennis et al. [5] have used Alloy to analyse a radiation therapy machine, exposing flaws in the original design of the system.

In order to automate the transformation from UML to Alloy, tool called UML2Alloy [2], has been developed which transforms a subset of UML class diagrams and OCL constraints into the Alloy language. UML2Alloy employs the SiTra Model Transformation framework to carry out the transformation. The tool works by transforming a UML class diagram with OCL constraints into the Alloy language and interfaces with the Alloy Analyzer, using the Alloy Analyzer API to automatically produce an instance, which conforms to the model. The method presented in this paper can be applied to produce UML object diagrams from the Alloy instances, such as counterexamples produced by the Alloy Analyzer.

2.2 UML Object Diagram Standard

Modeling standards (or specifications) such as MOF, EMOF [14], EDOC [13] are defined for the creation of software as models. Models in such languages are expressed at different layers, and layers are normally described by another layer. For example, in the MOF hierarchy there are four layers. At the highest layer (M3), MOF is defined. One layer below M3 is UML language specification; M2 is the UML meta-model. UML Models are created at the M1 layer to represent a system under development. In the UML standard, the M0 layer is used for run-time instantiations of a modeled system. In this paper, we use the M0 layer to represent instances of a M1 model. This is so we can create a model transformation at that level. A similar discussion on extending the layers relationship in UML is made in [24].

3 Description of the Problem

UML2Alloy is a tool to automatically analyse UML Class Diagrams with OCL constraints. This is achieved by defining a mapping between the UML meta model and the Alloy meta model (w in Figure 1). The tool works by converting the UML model to an equivalent model in the Alloy language (x in Figure 1) and using the Alloy Analyzer API it carries out the analysis. Some analysis is produced in the form of Alloy instances, a form the user is not familiar with. Standards based tools support the UML language exclusively and are oftentimes used as an Integrated Development Environment. UML tools usually do not support analysis natively but could be used to view and modify analysis produced externally, where it is in standard UML form. Using UML2Alloy and a conversion of Alloy instances, the analysis could then be used as part of the normal development process.

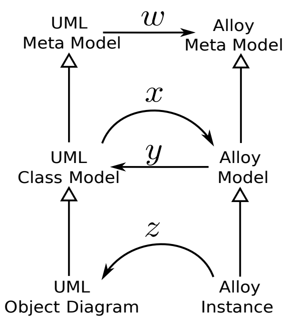


Fig. 1. Multiple layers in UML2Alloy. Horizontal arrows represent transformation, vertical arrows show instance-of relationship.

The key difficulty in converting Alloy instances, back to UML instances (z in Figure 1) is the inherent semantic differences between UML and Alloy. Instances in Alloy are not naturally instances of the originating UML model- some information is “lost in transformation”. An example of this semantic difference between UML and Alloy is where the attributes and associations of a UML model are converted to Alloy fields. Therefore, converting an Alloy instance back to an instance of the originating UML model would require knowledge of precisely how attributes and associations were converted to fields.

Although it may be possible for hand conversion of instances to infer the mappings manually (via inspection) the process would be time consuming, tedious and error prone. The instances produced by Alloy could be both large and numerous, so not suitable to manual conversion. As a result there is a need to automate the conversion of Alloy instances into UML instances, we propose a model transformation be used for the conversion, z in Figure 1. As the meta models (UML Class Model and Alloy Model in Figure 1) can change, the model transformation between them should ideally be generated automatically, based on the UML2Alloy transformation.

Another problem with manual conversion or even manually created model transformation is the accuracy of the conversion. In hand-converted models, misinterpretation of the original transformation may mean instances are converted incorrectly. Similarly, a manually created model transformation is prone to developer error, resulting in many wrongly converted instances of analysis. Our solution creates the analyses (instance) transformation automatically, based on the execution trace of the original UML2Alloy model transformation. This leads to a higher degree of confidence that there is consistency between the two transformations. A result of using model transformation throughout allows for an implementation in an MDA-compliant tool.

4 Outline of the Solution

Figure 2 depicts the enumerated steps of our solution to convert Alloy instances to UML instances. Figure 1 is related as follows, w is "UML2Alloy" and z is the instance converter "Alloy instance converter" of Figure 2. Our solution centres on an initial transformation in UML2Alloy, where given a UML model an Alloy model will be produced (Step 1 in Figure 2). The resulting Alloy model can be automatically analysed, with some of the analysis produced as Alloy instances (Step 2 in Figure 2). Using the trace of the first UML2Alloy transformation, we create another model transformation (Step 3 in Figure 2). This second transformation is used on Alloy instances, to convert them to UML instances (Step 4 in Figure 2).

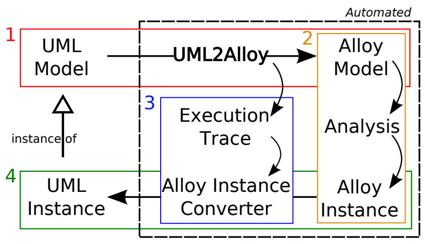


Fig. 2. Outline of the approach

Step 1: The first transformation, UML2Alloy is executed on a UML model, to produce an Alloy Model and a transformation trace. The properties of the resulting Alloy model can be automatically evaluated.

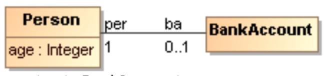
Step 2: In this step, UML2Alloy uses the Alloy Analyser API to automatically analyse the Alloy model. Two general kinds of analysis are performed to produce instances: simulation and assertion checking. Simulation produces an arbitrary instance, that conforms to the model constraints. Assertion checking will allow the user to verify a property of the model holds, with an instance produced (counter example) if the property does not hold.

Step 3: The next stage is to create the Alloy to UML instance converter. We propose that such a conversion can be achieved using MDA techniques: by creating a transformation. We use the trace of the original UML to Alloy model transformation to create the second instance transformation. In effect, the trace of the first transformation (executed in Step 1) is used as the specification of the second transformation (used in Step 4). Each trace instance, that has recorded a conversion in the first transformation is converted to a rule of the second transformation. Further details of can be found in Section 4.2.

Step 4: The second transformation (Step 4 in Figure 2) can be executed, converting Alloy instances to UML instances. The resulting instance can then be used in standard UML tool support.

4.1 Example

In this section we introduce an example UML model to illustrate the solution and used later to explain the implementation. Consider the UML Class Diagram with OCL constraints shown in Figure 3a. Using the model transformation UML2Alloy on this UML model, an Alloy model is produced shown in Figure 3b. For example, the UML class “Person” is converted to the Alloy signature “Person” (line 1, Figure 3b). Class Attributes become Fields of the Signature, for example the “age” attribute of “Person” in UML becomes the “age” field of the “Person” signature (line 2, Figure 3b). The navigable UML association between “Person” and “BankAccount” becomes the fields and facts on lines 3, 6, 8-10 in the Alloy model. The conversion of associations and attributes to fields highlights a major semantic difference between the formalisms, see section 3 for further discussion. Finally the OCL constraint of the model is converted to the facts in the Alloy model (line 12).

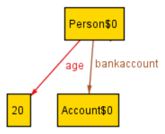


```
context BankAccount
            inv cond:self.per.age>18
```

(a) UML model

```
1   some sig Person{
2   age : one Int,
3   ba : lone BankAccount}
4
5   some sig BankAccount{
6   per : one Person}
7
8   fact{per in BankAccount lone->one Person}
9   fact{ba in Person one->lone BankAccount}
10  fact{per = ~ba}
11
12  fact{all self:BankAccount|int self.per.age>18}
```

(b) Output of UML2Alloy, using (a)



(c) Alloy instance of (b)

Fig. 3. Multi-level Model Transformation, with examples

The Alloy model can be simulated using the Alloy Analyzer to produce one (often many more and larger) Alloy instances such Figure 3c. The UML2Alloy transformation produces a trace. So in the above example, a trace is created when Class "Person" is converted to sig "Person". The trace of class to sig also refers to another trace, attribute "age" to field "age".

Using this information, the transformation Trace2MT will create the instance transformation (Step 3 in Figure 2). Based on the trace, the first rule created is PersonSig2PersonClass, which takes an instance of the "Person" signature in Alloy and converts it to an instance of the UML "Person" Class. PersonSig2PersonClass will be created to invoke another rule AgeField2AgeAttribute, which converts instances of the Alloy "age" field to instances of the UML "age" attribute. The rest of the model transformation (Step 4 in Figure 2) is created by repeating the process for every trace (produced in Step 1, Figure 2). Once the transformation has been created, Alloy instances can be automatically converted to UML instances.

4.2 Implementation

In this section we present an MDA centric implementation of our solution (Step 3 in Figure 2), by creating a prototype Model Transformation. The main artefact of the implementation is created using the QVT [16] standard, created using SmartQVT [20]. Trace2MT (Step 3 in Figure 2) converts the trace of a first transformation in UML2Alloy (Step 1 in Figure 2) into a second transformation that converts a given Alloy instance into a UML object diagram (Step 4 in Figure 2). In the implementation, the second generated transformation is also in the form of a QVT model transformation.

The rules of Trace2MT are defined between the trace meta model and the QVT Operational [16] meta model. The trace meta model is instrumental to this implementation, as it is the basis for the generated model transformation. We have defined a trace meta model based on the need to generate transformation rules. The pertinent features of our trace meta model for this purpose are preservation of order of rule invocation as well as the hierarchy of rule invocation.

The important rules of Trace2MT are "first2entryoperation" and "traceinst2rule", both invoked by the entry rule "itrace2operational", which in turn is invoked by the main entry point. The rule "first2entryoperation" converts the first of the TraceInstance, into the main entry point (EntryOperation from [16]) of the generated model transformation. The rule "traceinst2rule" of Trace2MT converts a given TraceInstace to a rule of the instance converter (MappingOperation from the QVT meta model [16]).

Figure 4 depicts the trace meta model utilised in the implementation. The ITrace class of the meta model is instantiated once in an execution of the UML2Alloy transformation. The TraceInstance class is instantiated at rule execution, whenever a UML element is converted to an Alloy element. The trace model records all transformation execution, including transformation of primitive types. The TraceInstance class records the source and destination value of a mapping. Where one rule invoked another rule, a link is also created between the two traces in TraceInstance. The information from the trace model is used to build the "Alloy instance converter" from Figure 2.

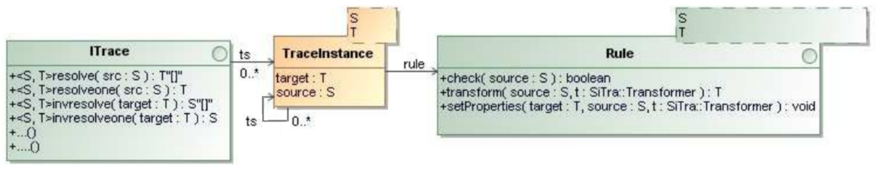

ITrace
+<S, T>resolve(src : S) : T"[]"
+<S, T>resolveone(src : S) : T
+<S, T>invresolve(target : T) : S"[]"
+<S, T>invresolveone(target : T) : S
+...()
+....()
ts
0..*
S
T
TraceInstance
target : T
source : S
ts
0..*
rule
S
T
Rule
+check(source : S) : boolean
+transform(source : S, t : SiTra::Transformer) : T
+setProperties(target : T, source : S, t : SiTra::Transformer) : void

Fig. 4. Trace meta model

5 Discussion and Related Work

In this paper we present a method to automatically generate the transformation rules to convert Alloy instances to UML Object diagrams. This allows instances generated in analysis using Alloy to be transfered back to the UML domain. The presented method uses the trace of the UML2Alloy transformation to create rules for the transformation of instances. In our method, the outcome of analysis is presented is standard UML form, so it can be used in existing tool support. The trace is used to create the model transformation and this allows us to avoid the inherent differences between UML and Alloy and thus convert Alloy instance back to UML automatically. Using trace data to generate the transformation allows for a high degree of confidence in the consistency of the converter. The generated transformation could be used to convert many, large complex Alloy instances back to UML form automatically.

If the original UML model changes, UML2Alloy can be used to automatically create an equivalent Alloy model. Any previously created transformation of instances would no longer be valid, using the presented solution the transformation can be re-created automatically. We have presented an example of how this solution could be implemented in a MDA-compliant tool and using only model transformation. We have presented our solution in terms of UML2Alloy, further research is required to understand where the reverse transformation is appropriate for any given transformation. For example how and when to apply the technique in the validation of other UML diagrams.

The work presented in this paper can be related to several different areas within model driven development, but only a few of those are particularly relevant to instance conversion for analysis. In [25, 22], discussion is on infering model transformation rules using a manually created mapping between particular models i.e. model transformation rules from an example. The current approach differs in that rules are created top-down from a higher to a lower level of abstraction and in using the trace of an existing model transformation. Also relevant to the work presented here is [23], who introduce the concept of meta-transformation. We utilise meta-transformation where input or output of model transformation is model transformation in converting traces to model transformation. The work in [23] focuses on architectural, practical and conceptual issues of creating such transformations.

Bibliography

[1] J.-R. Abrial. *The B-book: assigning programs to meanings.* Cambridge University Press, New York, NY, USA, 1996. ISBN 0-521-49619-5.

[2] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. UML2Alloy: A Challenging Model Transformation. In G. Engels, B. Opdyke, D.C. Schmidt, and F. Weil, editors, *ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems*, volume 4735 of *LNCS*, pages 436–450, Nashville, USA, 2007. Springer.

[3] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. On Challenges of Model Transformation from UML to Alloy. *Software and Systems Modeling, Special Issue on MoDELS 2007*, 2009. Accepted for publication subject to minor revisions.

[4] Behzad Bordbar and Kyriakos Anastasakis. UML2Alloy: A tool for lightweight modelling of Discrete Event Systems. In Nuno Guimarães and Pedro Isaías, editors, *IADIS International Conference in Applied Computing 2005*, volume 1, pages 209–216, Algarve, Portugal, February 2005. IADIS Press. ISBN 972-99353-6-X.

[5] Greg Dennis, Robert Seater, Derek Rayside, and Daniel Jackson. Automating commutativity analysis at the design level. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pages 165–174. ACM Press, 2004. ISBN 1-58113-820-2.

[6] Andy Evans, Robert France, and Emanuel Grant. Towards Formal Reasoning with UML Models. In *Proceedings of the OOPSLA'99 Workshop on Behavioral Semantics*, 1999.

[7] Geri Georg, Indrakshi Ray, Kyriakos Anastasakis, Behzad Bordbar, Manachai Toahchoodee, and Siv Hilde Houmb. An Aspect-Oriented Methodology for Developing Secure Applications. *Information and Software Technology. Special Issue on Model Based Development for Secure Information Systems.* Accepted for publication.

[8] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis.* The MIT Press, London, England, 2006.

[9] Daniel Jackson and Kevin Sullivan. COM revisited:tool-assisted modelling of an architectural framework. In *8th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, San Diego, CA, 2000.

[10] Sarfraz Khurshid and Daniel Jackson. Exploring the design of an intentional naming scheme with an automatic constraint analyzer. In *ASE '00: Proceedings of the 15th IEEE international conference on Automated software engineering*, page 13, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0710-7.

[11] Soon-Kyeong Kim. *A Metamodel-based Approach to Integrate Object-Oriented Graphical and Formal Specification Techniques.* PhD thesis, University of Queensland, Brisbane, Australia, 2002.

[12] I. Kurtev, J. Bézivin, and M. Aksit. Technological Spaces: an Initial Appraisal. *CoopIS, DOA*, 2002, 2002.

[13] OMG. enterprise distributed object computing (edoc), . URL http://www.omg.org/technology/documents/formal/edoc.htm.

[14] OMG. Metaobject facility (mof), . URL http://www.omg.org/mof/.

[15] OMG. *Meta Object Facility (MOF) 2.0 Core Specification.* OMG, 2004. URL www.omg.org.

[16] OMG. *MOF QVT Final Adopted Specification.* Object Modeling Group, 2007.

[17] Ilya Shlyakhter, Robert Seater, Daniel Jackson, Manu Sridharan, and Mana Taghdiri. Debugging overconstrained declarative models using unsatisfiable cores. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering, Montreal, Canada*, pages 94–105. IEEE Computer Society, 2003.

[18] Colin Snook and Michael Butler. Uml-b: Formal modelling and design aided by uml. *ACM Transactions on Software Engineering and Methodology*, 15 (1):92–122, January 2006.

[19] Perdita Stevens. Bidirectional model transformations in qvt: Semantic issues and open questions. In *MoDELS*, pages 1–15, 2007.

[20] France Telecom. Smartqvt: An open source model transformation tool implementing the mof 2.0 qvt-operational language. URL http://smartqvt.elibel.tm.fr/.

[21] OMG UML. 2.0 superstructure final adopted specification. *OMG Document reference ptc/03-08*, 2, 2003.

[22] D. Varro. Model transformation by example. *Lecture Notes in Computer Science*, 4199, 2006.

[23] D. Varro and A. Pataricza. Generic and meta-transformations for model transformation engineering. *Lecture Notes in Computer Science*, pages 290–304, 2004.

[24] D. Varró and A. Pataricza. VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML. *Software and Systems Modeling*, 2(3):187–210, 2003.

[25] Manuel Wimmer, Michael Strommer, Horst Kargl, and Gerhard Kramler. Towards model transformation generation by-example. In *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, Washington, DC, USA, 2007. IEEE Computer Society.

[26] Jim Woodcock and Jim Davies. *Using Z: Specification, Refinement, and Proof.* Prentice Hall, Upper Saddle River, NJ, USA, 1996.